# Oracle® Retail Job Orchestration and Scheduler

Implementation Guide

Release 19.0

**F23589-01**

January 2020

ORACLE®

Oracle® Retail Job Orchestration and Scheduler Implementation Guide Release 19.0

F23589-01



Primary Author: Sanal Parameshwaram

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 JOS Process Flow

## 5 Scheduler

## 6 Use Cases

## 7 Pre-Implementation Considerations



## 8 High Availability Considerations



## 9 Deployment Architecture



## 10 Performance Considerations



## 11 OAuth 2.0



## A Appendix A: Scheduler REST Endpoints

## G Appendix G: Job Admin UI Screenshots

# Send Us Your Comments

Oracle Retail Job Orchestration and Scheduler Implementation Guide, Release 19.0.

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

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

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

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

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

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

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

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

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

# Preface

This is a documentation of the Oracle Retail Job Orchestration and Scheduler Implementation Guide.

## Audience

The Implementation Guide is intended for the Oracle Retail Job Orchestration and Scheduler 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 or additional patches, read the documentation for all releases that have occurred since the base release before you begin installation. Documentation for patch releases can contain critical information related to the base release, as well as information about code changes since the base release.

## Improved Process for Oracle Retail Documentation Corrections

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

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

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

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

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

## Oracle Retail Documentation on the Oracle Technology Network

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

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

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

## Conventions

The following text conventions are used in this document:

| Convention | Meaning |
|---|---|
| **Navigate:** | This is a navigate statement. It tells you how to get to the start of the procedure and ends with a screen shot of the starting point and the statement "the Window Name window opens.". |
| **Note:** | This information is provided to improve your understanding, simplify a task, or point out special circumstances. |
| **Important:** | This information is important for the user to be aware of. For example, information that can help prevent the loss of data. |
| `code` | `This is a code sample. It is used to display examples of code.` |

# 1

# Introduction

Oracle Retail Job Orchestrator and Scheduler (JOS) is a generic tool that manages, executes, orchestrates, and schedules batch jobs for an enterprise, addressing the dependencies between tasks and processing non-interactive long-running jobs.

## Standards and Specifications

JOS is designed and built on industry standard nonproprietary Java EE 7 and Java Batch (JSR 352).

### 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 specifications 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. The Java EE specifications 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/problems of the components rather than spending time building scalable, robust infrastructure on which to run.

### Java Batch

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

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

1. Application developers have clear, reusable interfaces for constructing batch style applications.
2. Job writers have a powerful expression language for how to execute the steps of a batch execution.

3. 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 WebLogic server implements the Java EE specification and is the Java EE server vendor for JOS in this release. The WebLogic server provides many additional services beyond the standard services required by the Java EE specification.

See the WebLogic Application Server documentation for more information:

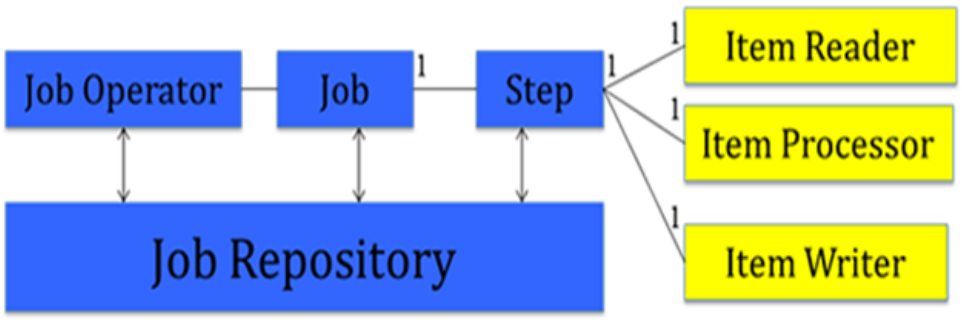
**http://docs.oracle.com/middleware/12213/wls/index.html**

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

# Java Batch Overview

Batch processing for the 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 typically bulk oriented, non-interactive, and long running.

***Figure 1–1 Batch Processing***




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

- A step contains all the necessary logic and data to perform the actual processing. A chunk-style step contains ItemReader, ItemProcessor, and ItemWriter.
- The Job Operator provides an interface to manage all aspects of job processing.
- The Job Repository holds information about jobs currently running and jobs that ran in the past.

# 2

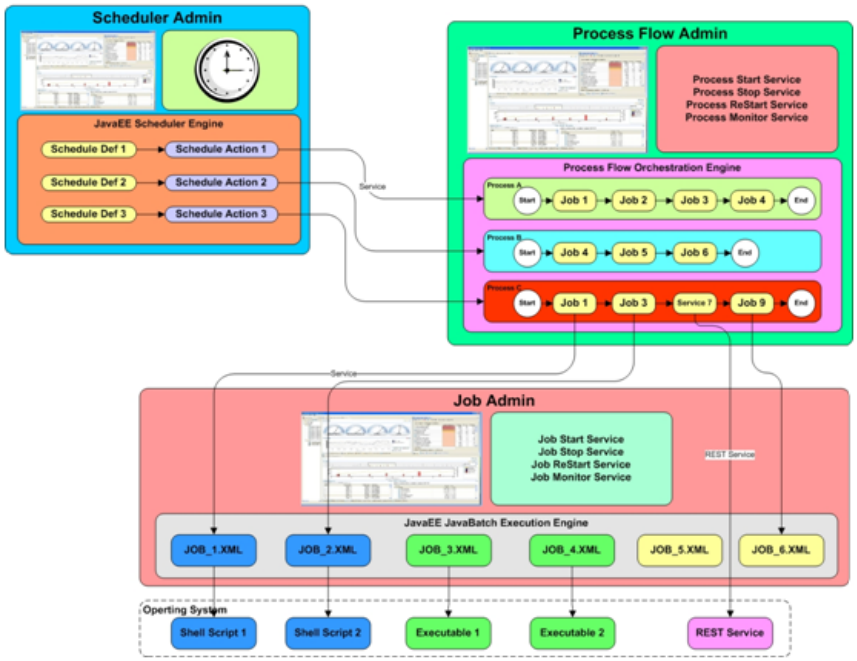
# JOS Components

The JOS components include the following:

- **Scheduler** – A generic GUI tool used to define and manage time-based scheduling work.
- **Process Flow** – Defines the workflow by connecting and orchestrating multiple executable tasks. Provides an engine to execute the workflows.
- **Job Admin** – A robust task execution engine based on standard JavaBatch (JSR352) technology. Provides a GUI to manage and monitor jobs.

## JOS Architecture

***Figure 2–1 Job Orchestration and Scheduling (JOS) Architecture***



# 3

# Job Admin

This chapter describes the Job Admin functionality.

## Job Admin Concepts

Here is a list of the Job Admin features.

- Provides management and monitoring of batch jobs.
- Services available to start/restart/monitor jobs programmatically.
- Monitors executions and logs.
- Built on JavaEE JSR352 JavaBatch standard and available on WebLogic.
- GUI to manage/start/restart batch jobs.
- JavaBatch Runtime Engine to execute job.xml files.

## Job Admin Components

The following section includes information about the Job Admin components.

### RESTFul Services

This section describes the RESTFul services.

#### Batch Service

The Batch service is a RESTful service that provides various endpoints to manage batch jobs. Here are the key endpoints in the Batch Service.

##### Start Job

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

**Sample Request**

```
http://localhost:7001/jos-batch-job-admin/resources/batch/jobs/ShellCommandRunnerJob?jobParameters=key=value
```

**Successful Response**

***Figure 3–1 Start Job Successful/Error Responses***

Successful Response

XML

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

JSON

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

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 endpoint 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/jos-batch-job-admin/resources/batch/jobs/executions/2
```

***Figure 3–2 Start Job Successful/Error Responses***

Successful Response

XML

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

JSON

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

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": ""
}
```

### 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:....."
}
```

### Inputs

jobName as path parameter

jobExecutionId as path parameter

### Sample Request

```
http://localhost:7001/jos-batch-job-admin/resources/batch/jobs/ShellComman
dRunnerJob/1
```

***Figure 3–3 Inputs Successful/Error Responses***

Successful Response

XML

```
<jobInstanceExecutionsVo targetNamespace="">
    <jobName>ShellCommandRunnerJob</jobName>
<jobInstanceId>1</jobInstanceId>
<resource>http://localhost:7001/jos-batch-job-
admin/resources/batch/jobs/ShellCommandRunnerJob/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": "ShellComandRunnerJob",
        "jobInstanceId": 1,
        "resource": "http://localhost:7001/jos-batch-job-
admin/resources/batch/jobs/ShellCommandRunnerJob/1",
        ["jobInstanceExecutionVo": {
            "executionId": 1,
            "executionStatus": "COMPLETED",
            "executionStartTime":"2016-07-11 15:45:27.356",
            "executionDuration": "10"
        }]
    }
}
```

Error Response

XML

```
<exceptionVo targetNamespace="">
    <statusCode>503</statusCode>
<status>SERVICE_UNAVAILABLE</status>
    <message>Service Unavailable</message>
    <stackTrace></stackTrace>  <!-- optional -->
</exceptionVo>
```

JSON

```
{
    "statusCode": "503",
    "Status": "SERVICE_UNAVAILABLE",
    "Message": "Service Unavailable",
    "stackTrace": ""
}
```

### 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"
   }
]
```

### System Setting Service

See Appendix D for details about System Setting Service.

### Job Metrics Service

Job Metrics service provides an end point that produces metrics for jobs based on time range.

**Path:** /telemetry/jobs

**HTTP Method:** GET

**Inputs:** fromTime as query parameter

toTime as a query parameter

**Sample Response**

```
{
     "data-requested-at": "2017-10-09T10:49:57.208-06:00",
     "data-requested-from-time": "2017-10-08T10:49:57.197-06:00",
     "data-requested-to-time": "2017-10-09T10:49:57.197-06:00",
     "jobs-server-runtime-info": {
         "id": "bdi-rms-batch-job-admin.war",
         "app-type": "BDI",
         "app-status": "RUNNING",
         "up-since": 1642,
         "total-jobs-count": 1,
         "total-executions-count": 4,
         "successful-executions-count": 0,
         "failed-executions-count": 4,
         "job": [
             {
                 "name": "ShellCommandRunnerBatchlet",
                 "slowest-run-duration": 0,
                 "fastest-run-duration": 0,
                 "avg-run-duration": 0,
                 "executions": {
                     "execution_count": 4,
                     "success_count": 0,
```

```
"failure_count": 4,
"execution": [
    {
        "execution-id": 121,
        "instance_id": 121,
        "status": "FAILED",
        "startTime": "2017-10-09T10:06:43-06:00",
        "endTime": "2017-10-09T10:06:43-06:00",
        "step": [
            {
                "step-execution-id": 121,
                "name": "shellCmd",
                "duration": 0,
                "status": "FAILED"
            }
        ]
    },
    {
        "execution-id": 122,
        "instance_id": 122,
        "status": "FAILED",
        "startTime": "2017-10-09T10:07:36-06:00",
        "endTime": "2017-10-09T10:07:36-06:00",
        "step": [
            {
                "step-execution-id": 122,
                "name": "shellCmd",
                "duration": 0,
                "status": "FAILED"
            }
        ]
    },
    {
        "execution-id": 123,
        "instance_id": 123,
        "status": "FAILED",
        "startTime": "2017-10-09T10:08:20-06:00",
        "endTime": "2017-10-09T10:08:20-06:00",
        "step": [
            {
                "step-execution-id": 123,
                "name": "shellCmd",
                "duration": 0,
                "status": "FAILED"
            }
        ]
    },
    {
        "execution-id": 124,
        "instance_id": 124,
        "status": "FAILED",
        "startTime": "2017-10-09T10:13:20-06:00",
        "endTime": "2017-10-09T10:13:20-06:00",
        "step": [
            {
                "step-execution-id": 124,
                "name": "shellCmd",
                "duration": 0,
                "status": "FAILED"
            }
```

```
                        ]
                    }
                ]
            }
        }
    ]
}
```

## 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": ""
}
```

## Job Admin UI

The Job Admin UI provides functionality to manage and monitor jobs. The Job Admin UI is described in detail in Appendix G.

## Best Practices

Best practices include:

- Use Batchlet if a job must run a script or program that resides locally.
- Use ItemReader, ItemWriter, and so on, if a job requires more control over the processing of data. Chunk processing allows data to be processed in chunks, and, if a job fails, it can only process the remaining chunks during a restart.
- Use job parameters to dynamically pass parameters to a job.
- Use PartitionMapper to specify the number of partitions and the threads for chunk processing so that all data is processed concurrently.

## Job Admin Security

Both Job Admin UI and REST Services are secured with SSL and basic authentication. The following roles are defined to restrict access to operations in Job Admin.

- JobAdminRole
- JobOperatorRole
- JobMonitorRole

Batch jobs can be run from the Job Admin UI or through the Batch REST service. Here are the operations that can be performed by the users based on their roles.

***Table 3–1 Job Admin Functions and Roles***

| 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 |

## Job Admin Customization

During the deployment of Job Admin, seed data is loaded to various tables. Seed data files are located in the jos-<app>-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 3–2 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 |

*Table 3–2 (Cont.) BDI Group*

| Column | Type | Comments |
|---|---|---|
| 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 3–3 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 |
| MEMBER_NAME | VARCHAR2(255) | Name of the job |
| MEMBER_TYPE | VARCHAR2(255) | Type of the member ex - job |

*Table 3–4 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 the 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).

## Job Admin Troubleshooting

This section describes the Job Admin errors and troubleshooting.

### Deployment Error

**Issue:** Job Admin deployment can encounter this error if the 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 the application, setting up new credentials and verifying that the deployment has been successful.

### 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 policies 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 the BDI Installation guide and verify that the WSM- policy manager is configured for the admin server URL.

Open the WebLogic domain console and target the wsm-pm application to Admin Server. Bounce Admin server and verify that the wsm-pm application is in an active state.

### 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:** Examine the WebLogic server log files to identify the root cause of the job failure. A possible root cause is the improper data source configuration.

### Job Admin UI Throwing Error: Job XML Not Found

**Issue:** Log files contain this exception:

Caused By: javax.ejb.EJBException: EJB Exception: : java.lang.RuntimeException: Could not find jobName(ShellCommandRunnerBatchlet) 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.

**Solution:** The job has been deleted from the jos-job-home before redeployment. Either add the missing job XML or delete the execution records of the job from the batch database.

#### Job Admin UI Throwing Error: Table or View Doesn't Exists

Issue: Log files contain this exception:

<Error> <javax.enterprise.resource.webcontainer.jsf.application> <BEA-000000> <Error Rendering View[/index.xhtml] javax.el.ELException: /index.xhtml @15,84 value="#{batchSummaryRequestBean.refreshPage}": javax.ejb.EJBException: EJB Exception: : com.ibm.jbatch.container.exception.PersistenceException: java.sql.SQLSyntaxErrorException: ORA-00942: table or view does not exist

**Solution:** BatchInfrastructure database is not pointing to a valid schema. Check if the schema has the following tables: CHECKPOINTDATA, STEPEXECUTIONINSTANCEDATA, STEPSTATUS, EXECUTIONINSTANCEDATA, JOBINSTANCEDATA, JOBSTATUS. If not, then run the DDL:

```
$Oracle_Home/oracle_common/common/sql/wlsservices/batch/oracle/jbatch.sql
```

#### IO Exception or Permissions Issue On Running A Shell Runner Job

**Issue:** java.io.IOException: Cannot run program "./TestShell1.sh" (in directory "/u00/webadmin/19.0.0/Scripts"): error=13, Permission denied

**Solution:** Check if the script the job is running is present in the specified location or not.

Check if the required permissions are provided for running the script.

#### Missing Credentials Access Permission

**Issue:** Caused by: java.lang.RuntimeException: Cannot get the Credential Map with the specified appLevelKeyPartitionName(DEFAULT_KEY_PARTITION_NAME).at com.oracle.retail.integration.common.security.credential.CredentialStoreManager.getAllUserNameAliases(CredentialStoreManager.java:1171) ~[retail-public-security-api-19.0.0.jar:?]Caused by: java.security.AccessControlException: access denied ("oracle.security.jps.service.credstore.CredentialAccessPermission" "context=SYSTEM,mapName=DEFAULT_KEY_PARTITION_NAME,keyName=*" "read")

**Solution:** Verify weblogic.policy file. The credential access permissions should be added for the domain where the applications are deployed. Add the permissions and restart the Admin and managed server.

### Missing System Credentials

**Issue:** Caused by: java.lang.IllegalArgumentException: alias(processCallBackServiceUrlUserAlias) not found in the credential store.

**Solution:** Add the system credentials using the UI or REST service.

### Missing System Options

**Issue:** 2017-03-31T02:49:42,628 [Thread-132] DEBUG Logger$debug - Starting job: null/resources/batch/jobs/DiffGrp_Fnd_ExtractorJob 2017-03-31T02:49:42,658 [Thread-132] ERROR Logger$error$0 - Error calling activity. java.lang.NullPointerException: Cannot invoke method getBytes() on null object

**Solution:** Add the system options using the UI or ReST service.

# 4

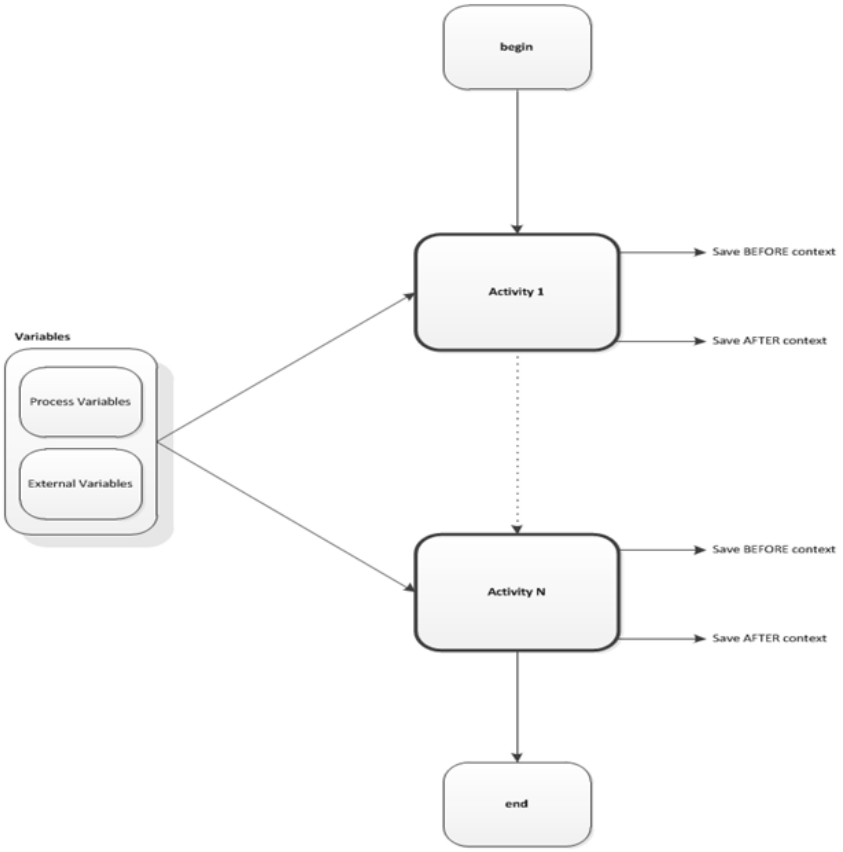
# JOS Process Flow

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 up a process from start to finish.

## Process Flow

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

***Figure 4–1 Process Flow***




## Process Flow Concepts

This section includes the following:

- DSL (Domain Specific Language)
- Process Variables

## DSL (Domain Specific Language)

Process flow definition is specified in a Domain Specific Language (DSL) built on the top of Groovy. Since Groovy is build on the top of Java Virtual Machine (JVM), Groovy can understand Java and Groovy language constructs. So the process flow DSL can understand the DSL, Groovy, and Java language constructs.

A process is a list of activities. The keywords "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 the process flow definition appears as the first activity. There should be only one begin 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. The name attribute is mandatory and should be used to give a unique name for the activity.

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 process flow definition appears as the last activity. There should be only one end 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 the DSL code.

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

```
// Set Variable

processVariables["VariableName"] = "Some Value"

// Use a variable value

def  anotherVariable = processVariables["VariableName"]
```

### External Variables

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

For example:

```
externalVariables["rxmJobAdminBaseUrlUserAlias"]
```

# Process Flow DSL

This section includes the following:

- Process Flow DSL Characteristics
- DSL Keywords
- Process Flow API
- Process Flow Variables

## Process Flow DSL Characteristics

The section describes the characteristics of the process flow DSL.

- Every process flow must have a name. The process flow name must match the filename that the process flow is saved into.
- Process flows are written in DSL and saved as .flo files.
- Process flow is made up of two special activities called "begin" and "end" and a group of user-defined activity nodes.
- The begin and end activities 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 the moveTo function.
  - Every activity has an action block that does the real work. Small amounts of Groovy/Java code can be put 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 such as $activityName and $name.
  - Errors can be thrown using the 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 can call service with just one line.
  - Services available to start/restart/monitor process flows programmatically.
  - Can handle chaining of process flows.
  - Service credential management framework built in.
  - 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

The tables below describe the following:

- DSL Keywords
- DSL Blacklisted Keywords
- Process Flow API
- Process Flow Variables

***Table 4–1 DSL Keyword Descriptions***

| 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 be 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 (examples - java, groovy, thread etc.)

***Table 4–2 Process Flow API Descriptions***

| 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(externalVariables["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. |

***Table 4–2 (Cont.) Process Flow API Descriptions***

| DSL API | Usage | Description |
|---|---|---|
| waitForJobCompletedOrFailed(def targetActivity, def url, String credentials, int waitMinutes=1) | waitForJobCompletedOrFailed("JobAbcActivity",externalVariables["url"] + "/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. |
| waitForProcessInstancesToReachStatus(def processInstanceList, def targetStatus=PROCESS_COMPLETED, def logicalAndOrOr = LOGICAL_AND, int waitSeconds=60) | waitForProcessInstancesToReachStatus(["P~1", "Q~1"], PROCESS_COMPLETED, LOGICAL_AND) | Method to wait for other process instances to reach a status. |
| waitForProcessNamesToReachStatus(Map, processNameToNumberOfExecutionsAfterStartMarkerTime, LocalDateTime startMarkerTime, def targetStatus = PROCESS_COMPLETED, def logicalAndOrOr = LOGICAL_AND, def whichExecutionStatus = LAST_EXECUTION_STATUS, int waitMinutes = 1) | waitForProcessNamesToReachStatus([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. |
| persistGlobalUserData(String 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(String key) | findGlobalUserData("key") | Gets value from BDI_SYSTEM_OPTIONS table for a given key. |
| Map findAllGlobalUserData(String key) | findAllGlobalUserData() | Returns a Map with all user data. |
| removeGlobalUserData(String key) | removeGlobalUserData("key") | 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. |

*Table 4–2 (Cont.) Process Flow API Descriptions*

| DSL API | Usage | Description |
|---|---|---|
| POST | POST[externalVariables.url]^externalVariables.urlUserAlias<br>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.<br>externalVariables.url - URL system option key configured in System Options table<br>customHttpHeaders - [a:"b", c:"d"]<br>Use "+" to provide custom http headers<br>Use "&" to provide response media type<br>Use "^" to provide basic authentication alias. User name and password will be Base64 encoded by the API.<br>Use "\|" to provide entity media type<br>Use "<<" to post data. The data will be in the format provided in entity media type. |
| GET | GET[externalVariables.url]^externalVariables.urlUserAlias<br>def response = (GET[externalVariables.url] + customHttpHeaders & MediaType.APPLICATION_JSON_TYPE ^ BasicAuth.alias1) as String | Method to make a GET call to a URL.<br>externalVariables.url - URL system option key configured in System Options table<br>customHttpHeaders - [a:"b", c:"d"]<br>Use "+" to provide custom http headers<br>Use "&" to provide response media type<br>Use "^" to provide basic authentication alias. User name and password will be Base64 encoded by the API |

*Table 4–2 (Cont.) Process Flow API Descriptions*

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

*Table 4–3 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" ])<br>//get value<br>def aVar = processVariables['myVar1']<br>//put new value<br>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. |

*Table 4–3 (Cont.) Process Flow Variables*

| Variables | Implicit or Explicit | Usage Examples | Description |
|---|---|---|---|
| Local variables | Explicit | action{<br><br>def a = "xyz"<br>def i = 7<br><br>i++<br><br>} | 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 activity. |
| Global external variables | Explicit | persistGlobalUserData("key1", "value1")<br><br>def xyz = findGlobalUserData("key1")<br><br>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.

## Sub-Processes

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 processes 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 the activity and the other for after the activity.

***Table 4–4 Process Schema Table Descriptions***

| 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 two 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, and so on, at the activity level |
| BDI_SYSTEM_OPTIONS | This table has all the system level information like URLs, credential aliases, and so on. |
| 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_LOCK | This table stores group names and lock ids |
| BDI_GROUP | This table stores group names and its attributes. |
| BDI_GROUP_MEMBER | This table stores all group member details. |
| BDI_EMAIL_NOTIFICATION | This table persist records for email notifications sent |

## Process Restart

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

---

**Note:** The restart is for an instance that has already run and failed. 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 (or stopped) in the previous run. During the 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 the restart. It will overwrite the activity records upon restart.

## Statuses

Each activity instance and the process instance maintain the status of execution in the process schema. 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 has also 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.

## Implementing a JOS Flow

The following steps show how to implement a JOS Flow.

1. Download JosProcessFlow<version>ForAll19.x.xApps_eng_ga.zip and unzip the file.
2. Create a process flow DSL file that stitches the jobs. DSL is Groovy based and Groovy or Java code can be used with in action block. See the following sample of DSL.
3. Copy the DSL files to jos-process-home/setup-data/dsl/flows-in-scope folder.
4. Run the deployer script in the jos-process-home/bin folder to deploy the JOS Process flow.

## Activity Features

This section includes the following activity features:

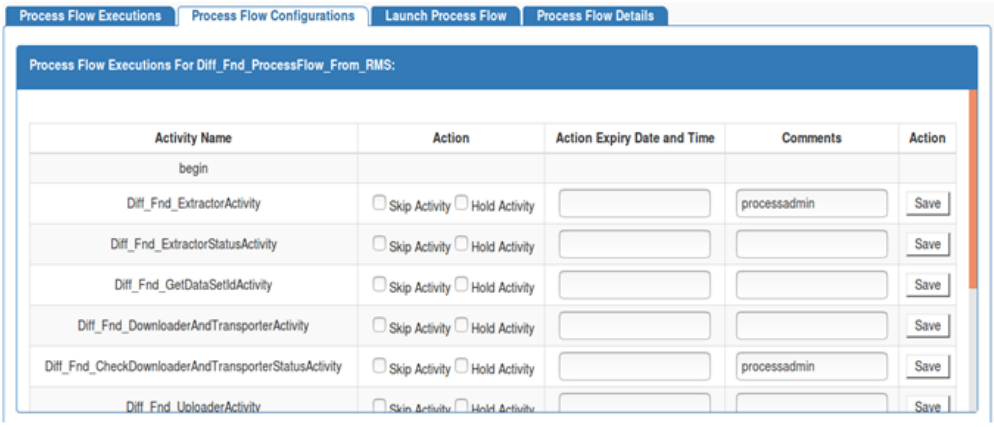
- Skip Activity
- REST Endpoint to Set the Skip Activity Flag
- Hold/Release Activity
- REST Endpoint to Set the Hold Activity Flag
- Bulk Skip/Hold

- 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 screen or REST endpoint. The skip flag can be set to expire, based on date and time. If the expiration date is not provided, then that activity will be skipped until the skip flag is removed. When an activity is set to skip, the process flow engine skips that activity and runs the next activity in the flow.

***Figure 4–2 Process Flow Configurations Screen***




## 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 screen or REST endpoint. The hold flag can be set to expire, based on date and time. If the expiration date is not provided, then that activity will be paused until the hold flag is removed. When an activity is set to hold, the process flow engine waits on that activity until the hold flag is removed or the time has expired.

## REST Endpoint to Set the Hold Activity Flag

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

## Bulk Skip/Hold

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

**Note:** Verify the updates in the BDI_ACTIVITY_DYNAMIC_CONFIG table.

```
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

The 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 JOS 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 JOS process flow, the user must a REST service call to the 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.

For example:

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

The following is the context information that must be passed to the JOS process flow from the calling system.

1. **callerId** - CallerId parameter is used to identify the invoker of process flow. In this case it is CAWA.
2. **`correlationId`** - Correlation ID is the main identifier used by the calling system (CAWA) 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 the 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 the Process Flow

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

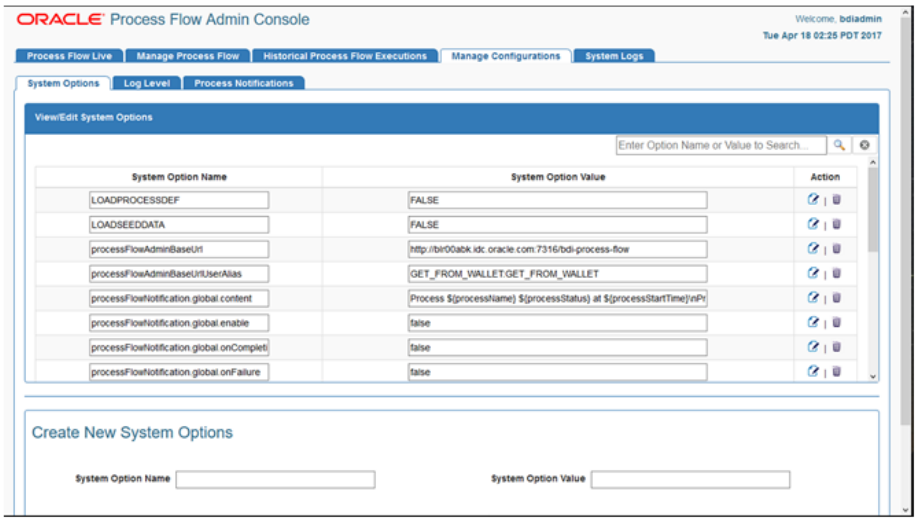
- The method internally invokes a REST call to the provided URL.
- The method uses basic authentication for the rest call. The credentials for the method call must be available in the process flow.
- The payload sent from process flow to the invoking application (CAWA) follows the contract as shown in 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 values from the DSL to invoking application (CAWA).
- The result of the callback REST service (in CAWA) must be a String value.
- If the callback service invocation fails for any reason (such as a network issue), the process flow activity fails and the process flow is marked as failed.

## How to Invoke the Callback Service Declaratively

Set up the callback URL in process flow system options. To configure a callback URL, you should add system options such as <serviceName>CallBackServiceUrl, for example, processCallBackServiceUrl.

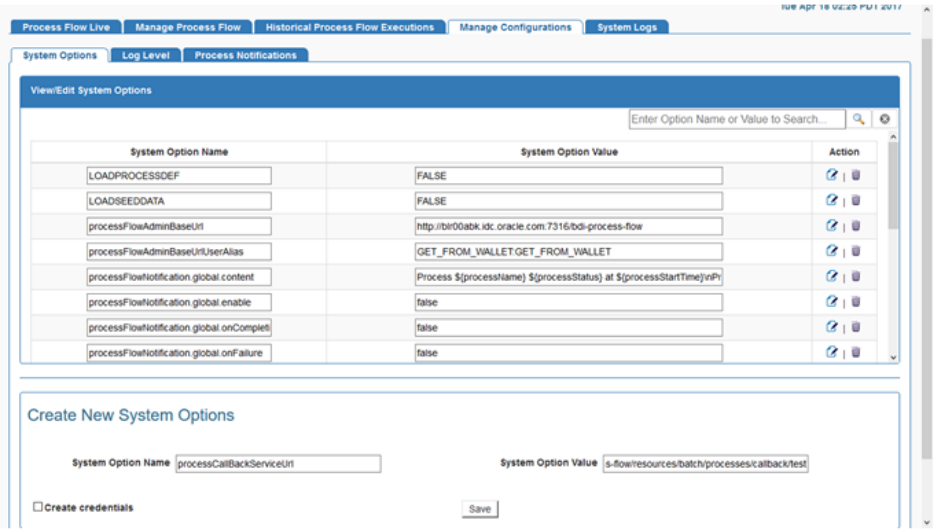
1. In the Process Flow Admin console, navigate to the Manage Configurations tab and the System Options sub-tab.

*Figure 4–3 Process Flow Admin Console Manage Configurations Tab*



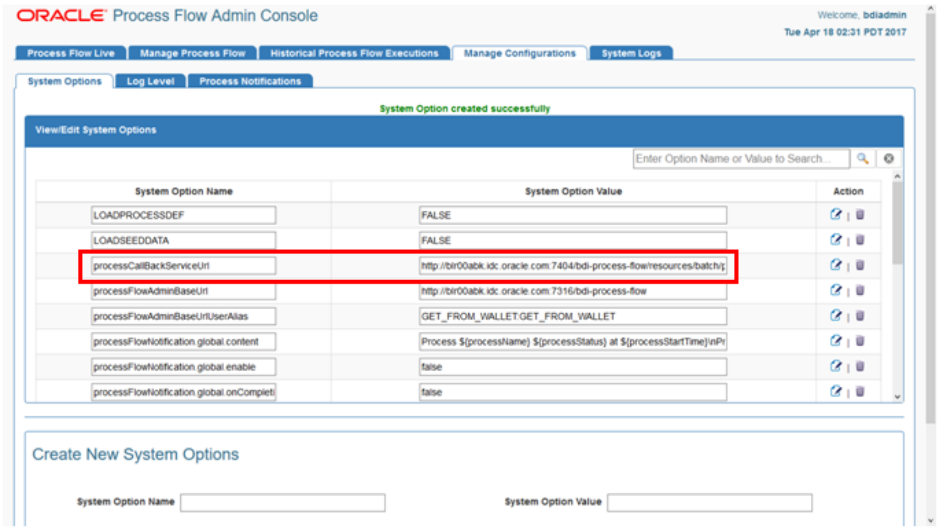

2. Scroll down to Create New System Options and enter **System Option Name** and **System Option Value**. The URL must be a valid ReST Service.

*Figure 4–4 Create New Systems Options Section*



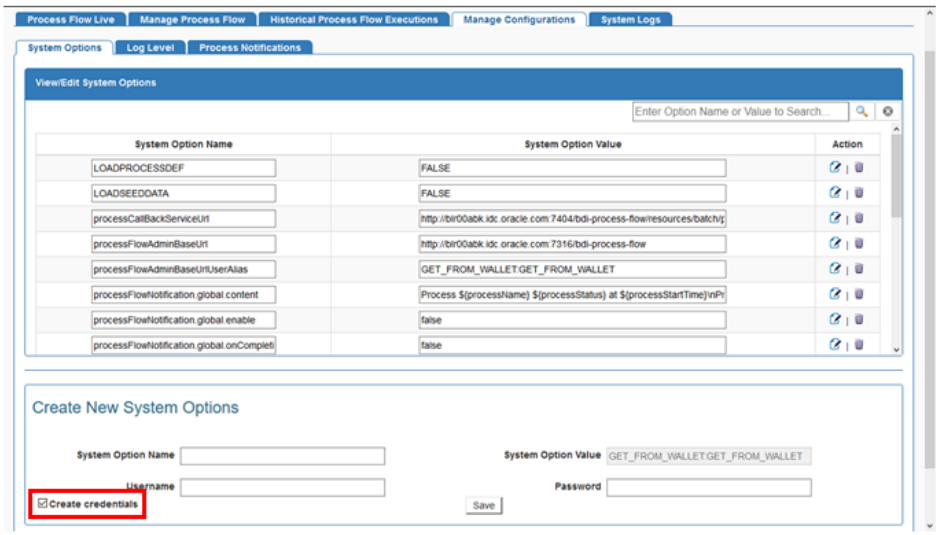

3. Click **Save**.

*Figure 4–5 Saved System Options*



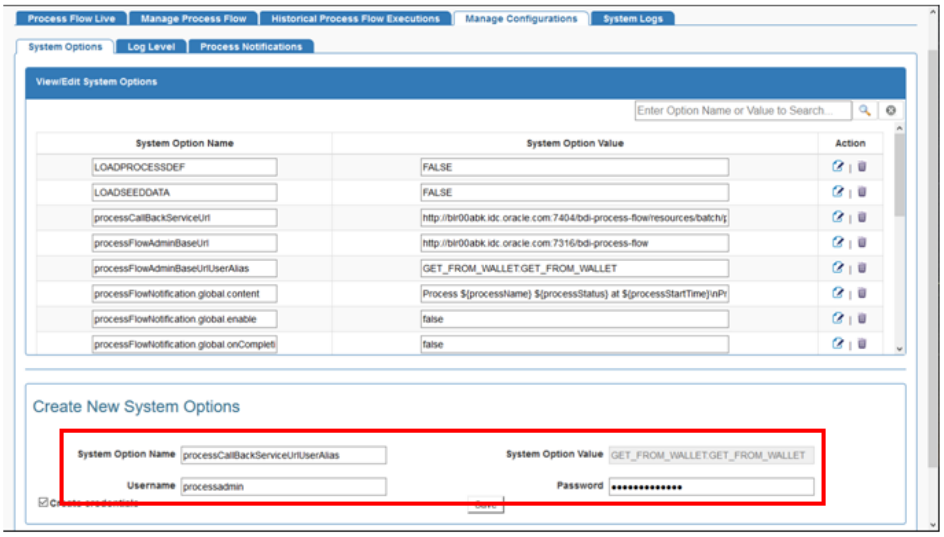

4. Set up the callback URL credential alias in the process flow. To add the callback URL credential alias, you must add credential alias such as <serviceName>CallBackServiceUrlUserAlias, for example, processCallBackServiceUrlUserAlias.
5. In the Create New System Options section, select **the Create Credentials** check box.

*Figure 4–6 Create Credentials Check Box*



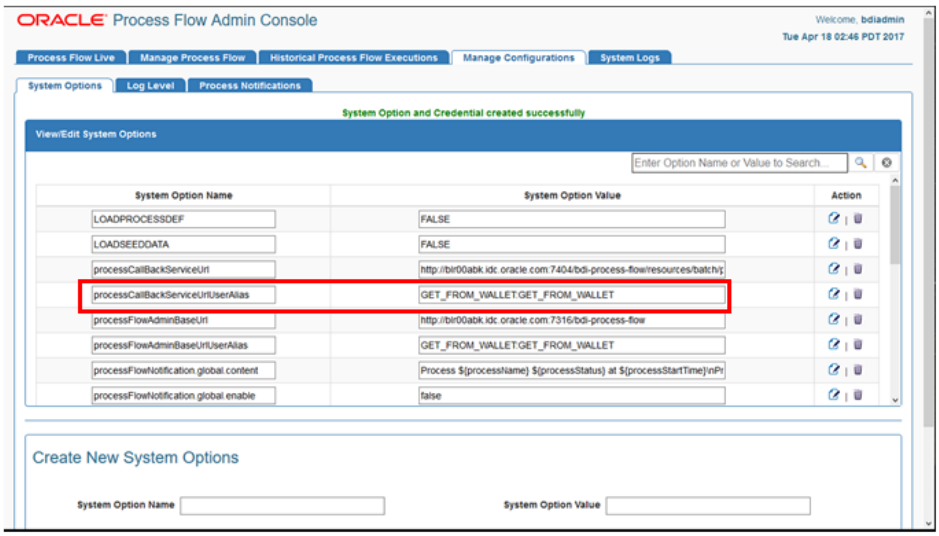

6. Enter **System Option Name**, **Username**, and **Password** for the URL provided in the previous step. If the system option name for the URL is **processCallBackServiceUrl**, then the system option name for the credential must be **processCallBackServiceUrlUserAlias**.

***Figure 4–7 Entering URL Credentials***




7. Click **Save**.

***Figure 4–8 System Option Name***



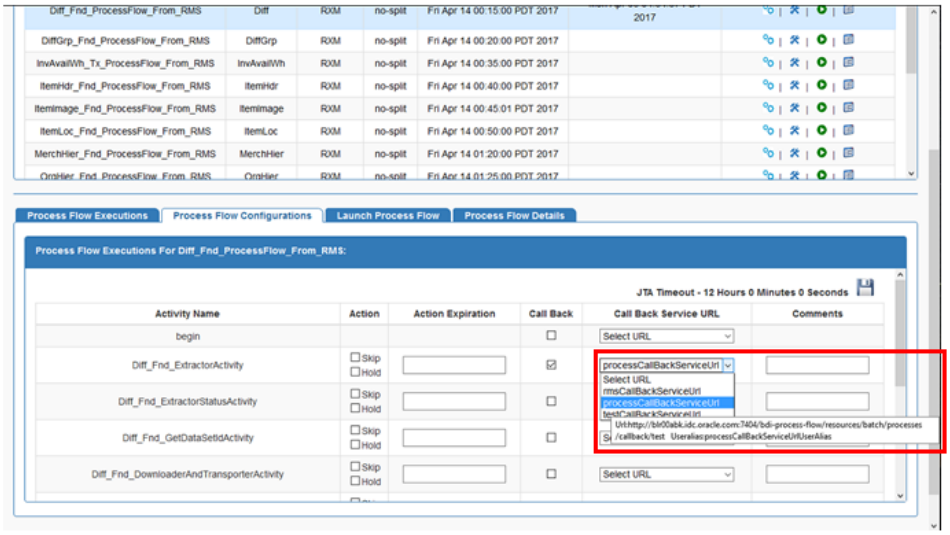

---

**Note:** Credentials created through the UI are available after a server restart; however, after the redeployment of the application, the credentials must be created again.

---

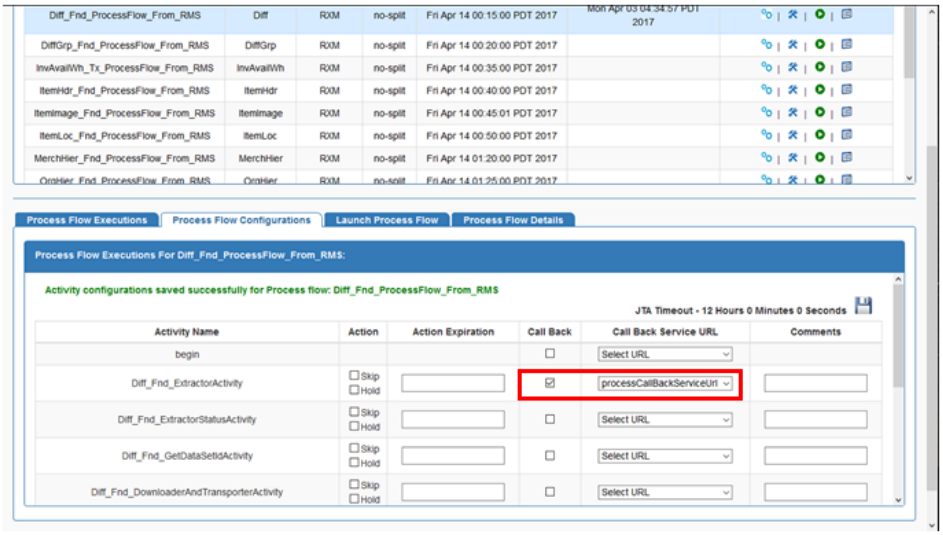
8. Navigate to the Manage Process Flow tab and select process flow, then go to Process Flow Configurations sub-tab.
9. Select **Callback** check box for the activities you want callback to be enabled for. Select **Callback URL** from the drop-down list.

*Figure 4–9 Configuring Callback Activities*




10. Click **Save**.

*Figure 4–10 Saved Callback Service URL*




## 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 properties must be set up in the System Options inside process flow.

### Example 1: Short Form

Add the following line inside the JOS 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 the user can update it with additional data inside an activity if necessary.

Add the following line inside the JOS 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 JOS process flow will make a POST REST call to the callBackService URL, passing in the following payload. JSON is the default content type.

***Figure 4–11 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"
   }
],
}
```

***Figure 4–12 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>
```

***Table 4–5 Call Back Service Scenarios***

| Activity Type | Activity Action | Callback Behavior | 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** |
| | 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 was started or restarted successfully. **ACTIVITY_FAILED** 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** |
| 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 was started or restarted successfully. **ACTIVITY_FAILED**if the job failed. | **ACTIVITY_FAILED** |

## Process Execution Trace

The Process Flow engine keeps track of process execution details in BDI_PROCESS_CALL_STACK_TRACE table. Also, in order for a sub-process to appear in the trace, the sub-process must be called with the new 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 BdiProcessAdminGroup, BdiProcessOperatorGroup, or BdiProcessMonitorGroup to use the process flow REST services and process flow admin application.

There are two authorization roles designed for the process flow application: the Operator role and the Admin 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, as identified in the table below. The Monitor role has the fewest access permissions.

***Table 4–6 Authorization Roles***

| Service/Action | Monitor Role | Operator Role | Admin Role |
|---|---|---|---|
| Update Process DSL | No | No | Yes |
| Start/Restart Process | No | Yes | Yes |
| Skip/Hold/Release | No | Yes | Yes |
| All other services | Yes | Yes | Yes |

# Process Customization

## Seed Data

During the deployment of Process Flow, seed data gets loaded. Seed data files are located in "jos-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 "jos-process-home/setup-data/dsl/flows-in-scope" folder. If you want to load DSLs again after DSLs are added or updated, 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 flows-in-scope i.e. /bdi-process-home/setup-data/dsl/flows-in-scope , 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.

Perform the following procedure:

1. Delete what was in scope before.

2. Copy the latest flows for what you are trying to integrate.
3. Deploy the application.

*Table 4–7 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 |

# Troubleshooting

Since the process flow can be written in Groovy and DSL, it is prone to programming mistakes. Any custom DSL must be properly tested before deploying. 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.

If any activity fails, the process is marked as failed. So in case of process failure, examine 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. The 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 a restart or a new run of the process flow can be used, depending on the requirement.

## 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 jos-process-home does not delete it from the process flow application because the process flow application refers to the database entries. In order to delete a process flow from the JOS Process Flow application, the script DELETE_PROCESS_FLOW.sql(jos-process-home/setup-data/dml/) must be run in the JOS Schema.

# Best Practices for Process Flow DSL

The following best practices for Process Flow DSL include:

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

- Use the built-in waitForJobCompletedOrFailed method to wait until job is complete or failed.
- Use the built-in triggerProcess to start a sub process.
- Access system options through externalVariables.
- Use processVariables to share variables between activities.
- Use the built-in waitForProcessInstancesToReachStatus to wait for other process instances.
- Use the built-in waitForProcessNamesToReachStatus to wait for other processes.
- It is recommended to use flo as the extension for the process flow DSL file.
- Use the built-in REST DSL to make rest calls.
- Organize process flows as hierarchical parent child flows, where the parent manages the child flows.
- Avoid using too many waitFor calls, as active threads can get blocked.

# 5

# Scheduler

The Scheduler application JOS product suite assists in the scheduling of batch processes to run at predefined configured time intervals. A schedule determines when a job, a process, or any program must be executed, as well as the frequency of execution.

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

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

The Scheduler supports the calling of REST services.

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

## JOS Scheduler Features

The Scheduler is a web application that provides a GUI for managing a schedule-based workload. It includes the following features:

- DSL based Schedule Action - Call process flows, running any local/remote programs.
- Run remote programs with REST calls.
- Externalized Schedule Definition and Schedule Actions. Easily import/export schedule and action definitions.
- GUI to create, edit, delete, enable, or disable schedules.
- Monitor schedule executions and logs.
- Monitor schedule's progress and history.
- Built-in e-mail notification.

## Scheduler Concepts

The following section describes the Scheduler concepts.

### Schedule Definition

A schedule definition contains details of a schedule such as Schedule Name and Schedule Group. These indicate the logical or functional grouping of schedules and schedule description.

### Schedule Execution

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

### Schedule Types

A schedule can be an interval-based schedule or a 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 five 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 a cron expression are defined as calendar-based schedules.

The following parameters define a calendar-based schedule. These are the 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 various scheduling mechanisms.

### Simple Scheduling

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

- Hourly
- Daily
- Weekly
- Monthly
- Weekday [Mon-Friday]
- Weekend [Sat-Sunday]
- Saturday
- Sunday
- First day of every month

- Last day of every month
- One time only (run once)
- User-specified frequency with intervals in the units minutes, hours, days, or weeks.

## Advanced Scheduling

The JOS Scheduler supports advanced scheduling, which is cron-like scheduling. Calendar-based schedules that can be expressed in cron-format can be set up 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 determine when the schedule must 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 a start datetime as 2016-08-15 10:00 a.m. and a repeat of 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 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 a start datetime as 2016-08-15 10:00 a.m. (which is a Monday) but repeats 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. 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, the first run of the Schedule equals Schedule Start Datetime. For calendar-based schedules, the first run of the Schedule may or may not be equal the Schedule Start Datetime, based on the schedule recurrence specified.

### Schedule End Datetime

It specifies an end date and time when the schedule must 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, if 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 Timzone

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 is refreshed when the UI is refreshed).

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

## Schedule Action

This section describes the various schedule actions.

### Schedule Action Definition

The Schedule Action defines what is executed when the schedule runs at the specified frequency. It is a DSL 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. JOS process flows are called by the Scheduler as REST services.

For example, to trigger a JOS 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. For example, the value of processFlowAdminBaseUrl might be https://<host>:<port>/bdi-process-flow.

For example, https://example.com:8001/bdi-process-flow

- /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 JOS Process Flow Admin's user credentials to access the process flow REST service.
  - The value for processFlowAdminBaseUrlUserAlias is specified during install time and is stored in the BDI System Options.

Basic authentication is used to access the JOS process flows. The Scheduler uses processFlowAdminBaseUrlUserAlias to look up 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. 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 JOS Process Flow Admin and not in the Schedule Action.

For example, if process-flow-2 must be run after process-flow-1 completes, use Process Flow Admin to define this dependency and not the Schedule Action.

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

---

**Note:** For security reasons, the usage of certain keywords is 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 a keyword is 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 must specify whether the schedule action is sync or async. Scheduler determines the action execution statuses according to the action type specified.

### Sync Action

It 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

Schedule action type determines the schedule action status during the execution life cycle.

### Sync Action Execution Statuses

It 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

- 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 schedule action as 'FAILED' when the action has been executed, the action should either throw an exception or return value as 'FAILED'.
- If the schedule action returns null or any other return value gracefully, the action execution status will be marked 'TRIGGERED' for async action and 'COMPLETED'

for sync action, and the returned response is stored as such in the Schedule Action Execution Log.

### Schedule Status

A schedule can be in one of the following statuses:

- **Active**: An active schedule is running at the specified frequency.
- **Inactive**: An inactive 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 various Scheduler Runtime options.

## 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 runtime timers for each schedule.

When the application is deployed for the first time, all schedules will be set up new. However, when the application is redeployed or the application server is restarted, any existing schedule timers 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 DLS contains any restricted keyword, the schedule will be Disabled at startup and will not run. The user must correct the schedule action definition from the Scheduler UI and enable the schedule to make it active.

## Schedule Runtime Execution

Scheduler uses 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 the callback method where the Scheduler will execute the schedule action.

Each schedule timer executes in 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.

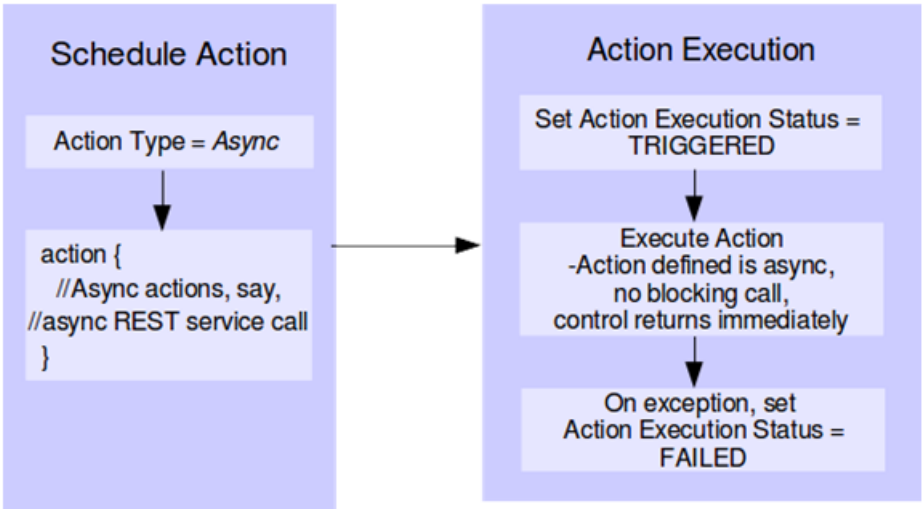
If any of the service call within the schedule action is a blocking synchronous call and the action is not defined to run in separate thread, then the action type must be 'sync'.

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 - Async Action

When the schedule action execution starts for an 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 separate thread returning the control immediately.

***Figure 5–1 Async Action***

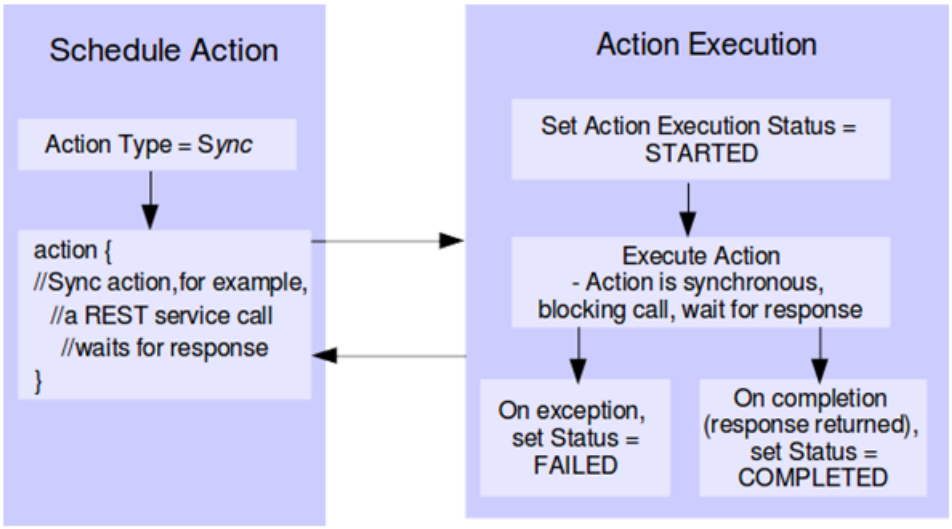



The execution of the action and the eventual status thereof will not be known to Scheduler. Once the control is returned, the schedule action execution ends but the status remains TRIGGERED. In case of an exception when the action is triggered, the status is set to FAILED and the execution ends.

## Schedule Execution - Sync Action

When the schedule action execution is started for a 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.

*Figure 5–2 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 returns, 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 was missed during the downtime. For example, if a schedule is scheduled to run every five minutes and the application server is down for fifteen minutes and restarted, the schedule will be run only one time and not three times. This is a feature supported by the Java EE container.

---

## Schedule Notification

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

- Notify when the schedule action execution begins.
  - This occurs when the schedule action execution is Started for sync action and before triggering of action execution for async action.
- Notify when the schedule action execution ends successfully.

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

## Scheduler Infrastructure Schema

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

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

***Table 5–1 Scheduler Infrastructure Schema***

| 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.<br>Seed data schedules are loaded in this table at deployment time during application startup. |
| BDI_SCHEDULE_EXECUTION | All schedule executions at runtime are persisted in this table. |
| BDI_SYSTEM_OPTIONS | This table contains system-level global parameters as key-value pairs used by the Scheduler at runtime, such as, Process Flow Admin Base URL, Process Flow Admin User Alias, which are configured at install time by the user. User can also add system parameters to be made available to the schedule actions. |
| BDI_EMAIL_NOTIFICATION | This table contains email notification details |

## Best Practices for Scheduler

Best practices include:

- Use POST DSL method to post to REST URL.
- Use externalVariables for accessing variables from BDI_SYSTEM_OPTIONS table.
- Use sch as extension for schedule action DSL file.
- Try not to use time-based dependency management between schedules; instead, use process flow to manage dependency.
- To schedule any existing jobs or programs, try to expose them as REST services and use the built-in DSL POST method to schedule action for executing the programs.
- Minimize use of synchronous schedule actions since they block until completion during each schedule execution.

# Scheduler Console

Scheduler Console (Admin UI) is a web user interface provided by 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 the Chrome web browser to access Scheduler Console since the calendar widget for datetime fields is supported by Chrome browser and not by Firefox or IE.

## Schedule Summary

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

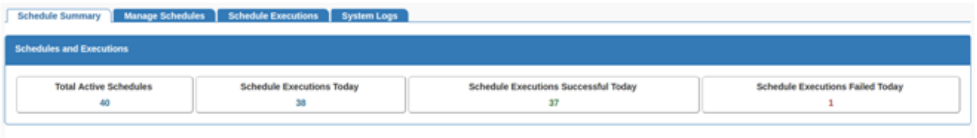
### Schedules and Executions

This displays the total count of:

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

> **Note:** The use of "Today" in the figure below indicates the duration from midnight to now.

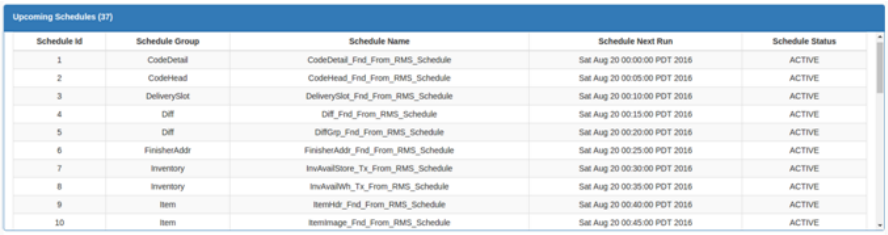
***Figure 5–3 Schedules and Executions Screen***




### Upcoming Schedules

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

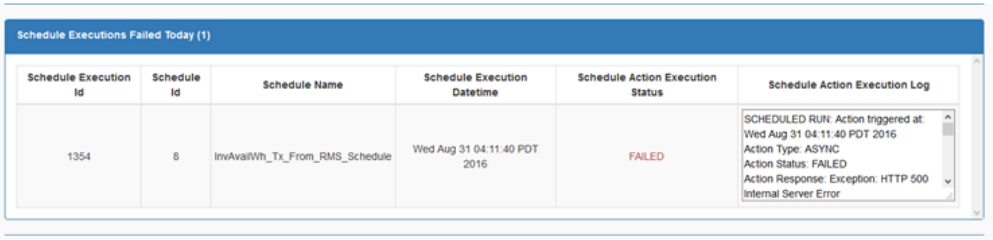
***Figure 5–4 Upcoming Schedules Screen***



### Schedule Executions Failed Today

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

***Figure 5–5 Schedule Executions Failed Today Screen***




### Schedule Executions Completed / Triggered Today

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

### Schedule Executions In Progress Today

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

### Schedules Past Due

This 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.

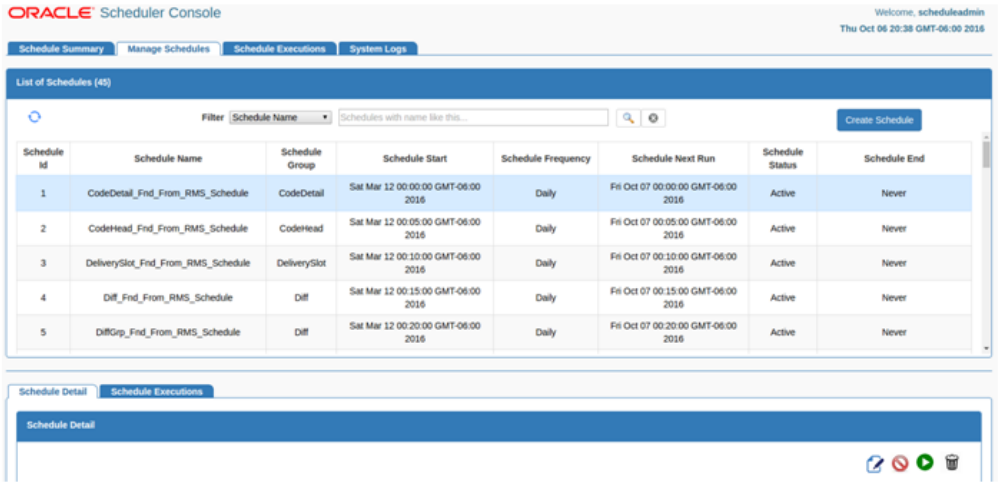
## Manage Schedules

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

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

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

*Figure 5–6 Scheduler Console*

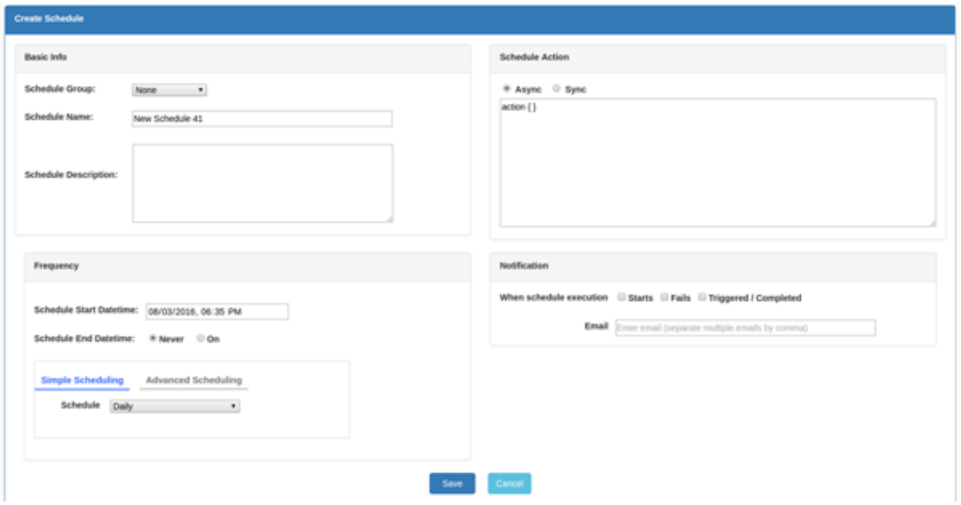



## Creating a Schedule

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.

- Basic Information
- Schedule Action
- Frequency
- Notification

*Figure 5–7 Create Schedule Screen*




### Basic Information

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 to the number of characters that these fields can accept.

### Schedule Action

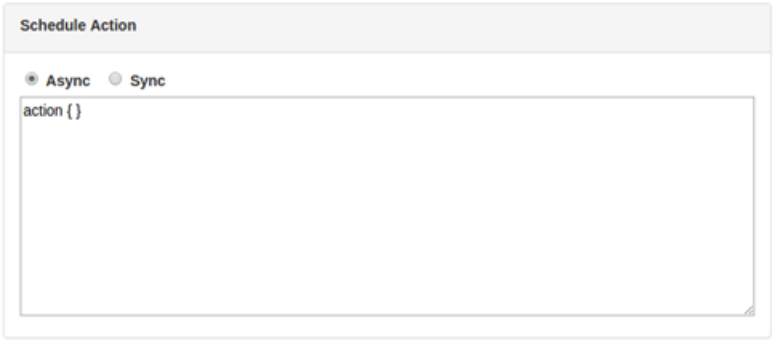
Specify a valid schedule action definition here that will get executed when the schedule runs.

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 a schedule, so any syntax or programming errors in the action definition will result in an exception at runtime and the schedule execution will fail.

***Figure 5–8 Schedule Action Screen***




### Schedule Frequency

It consists of Schedule Start Date time, End Date time, and Schedule Recurrence.

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.

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

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

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

#### Simple Scheduling

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

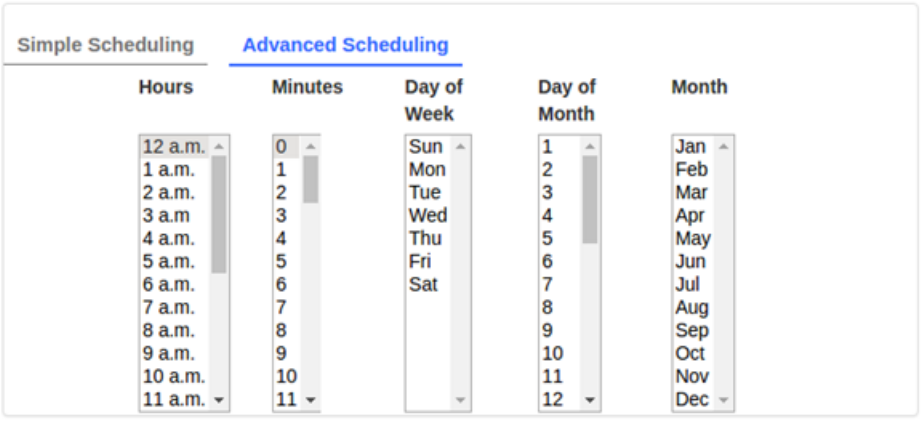
- Hourly
- Daily (selected by default)
- Weekly
- Every Weekday [Mon-Friday]
- Monthly
- On Weekends [Sat-Sunday]
- Every Saturday
- Every Sunday
- First day of every month
- Last day of every month
- One time only
- Specify a different frequency. User can use this option to specify a recurring interval in minutes, hours, days or weeks, for example, 30 minutes, 2 hours, 3 days, and so on.

### Advanced Scheduling

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 5–9 Advanced Scheduling***




## Schedule Notification

Use the schedule notification option to enable e-mail notification for the schedule when schedule execution starts or fails or completes.

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

### Starts:

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

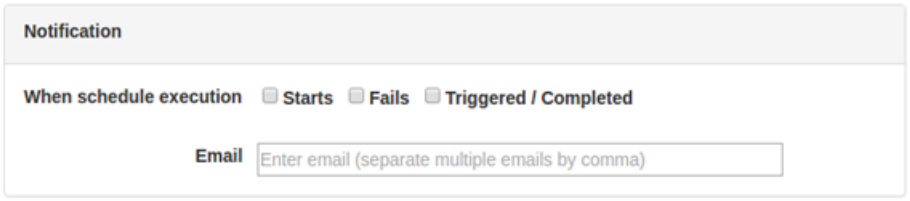
### Fails:

An e-mail is sent when there is an exception in 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 e-mail will be sent when the schedule action execution status is Triggered (for async actions) and Completed (for sync actions). This means the schedule execution is successful.

***Figure 5–10 Notification Screen***




---

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

---

## Updating a 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 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 schedule. Only an Admin user can edit a schedule.

---

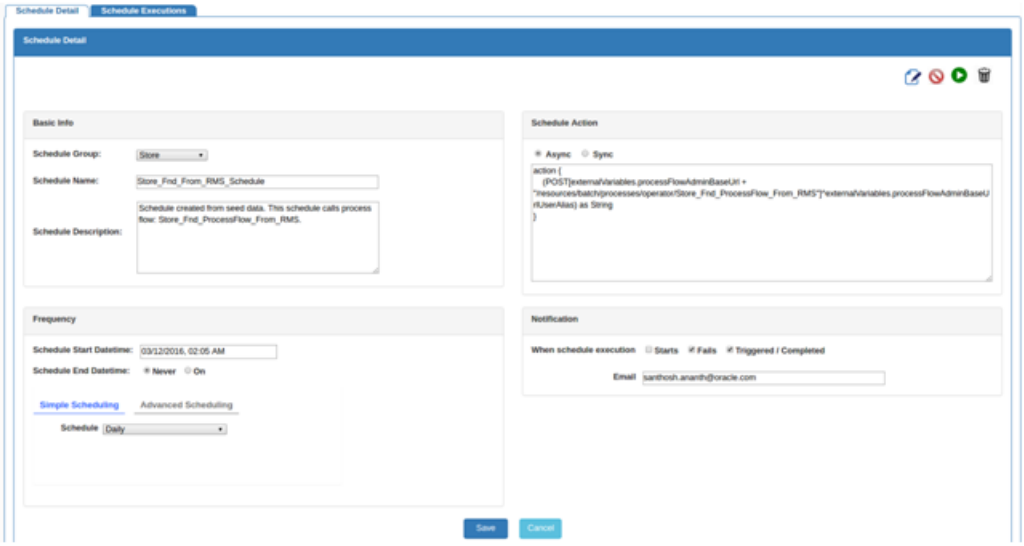
**Note:** Updating the schedule frequency will validate the schedule start datetime and end datetime (if specified), similar to when creating a schedule.

---

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 schedule definition will get updated as modified by the user.

When changing the schedule action definition, any restricted keywords will be validated.

*Figure 5–11 Schedule Detail Screen*




## Disabling a Schedule

A schedule can be disabled by selecting the schedule from Mange Schedule page and using the Disable schedule option in the Schedule Detail view. Only Admin and Operator users can disable a schedule.

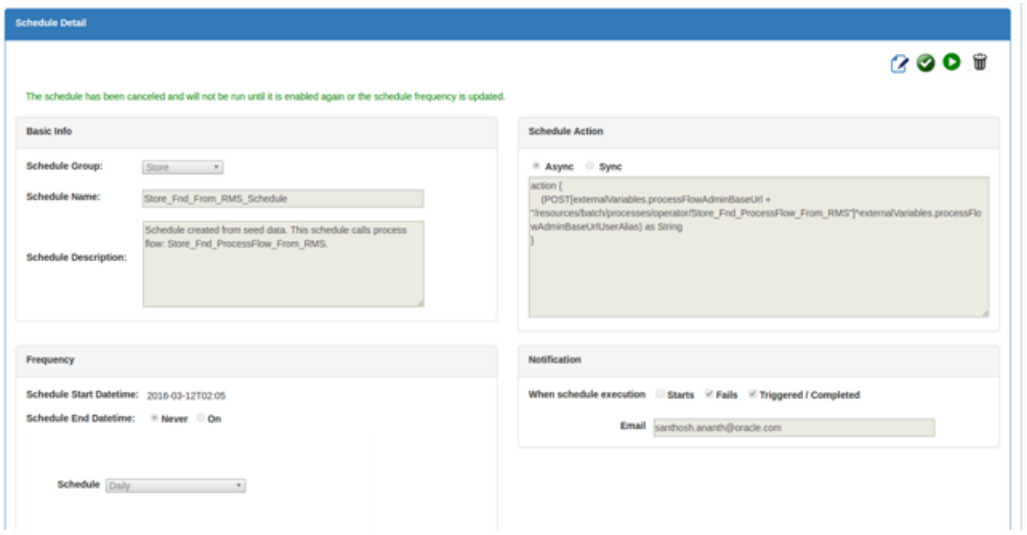
Disabling a schedule will change the schedule status to Disabled and the schedule will no longer run at the specified frequency. However, the schedule can be manually run using the Run Schedule Now option.

---

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

---

*Figure 5–12 Cancelled Schedule Message*

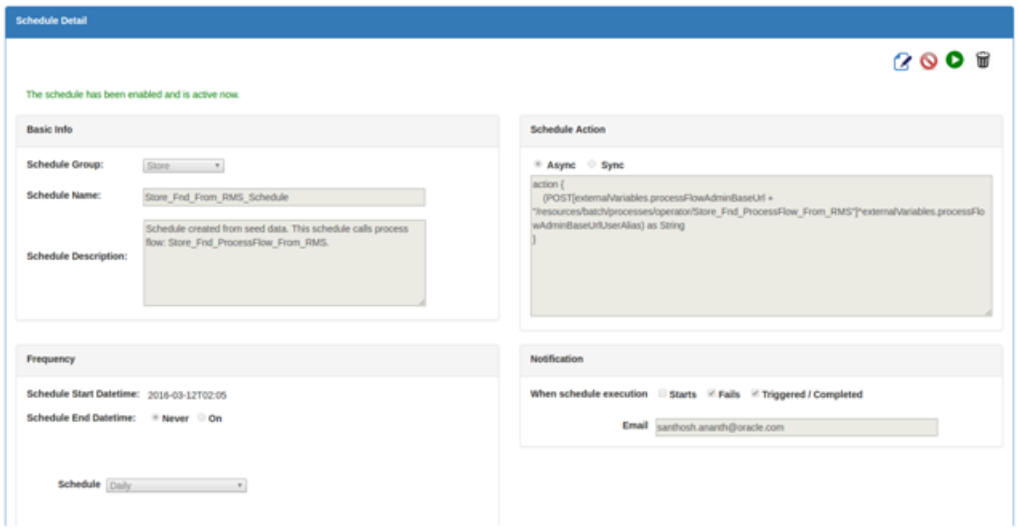


## Enabling a Schedule

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 5–13 Enabled Schedule Message***

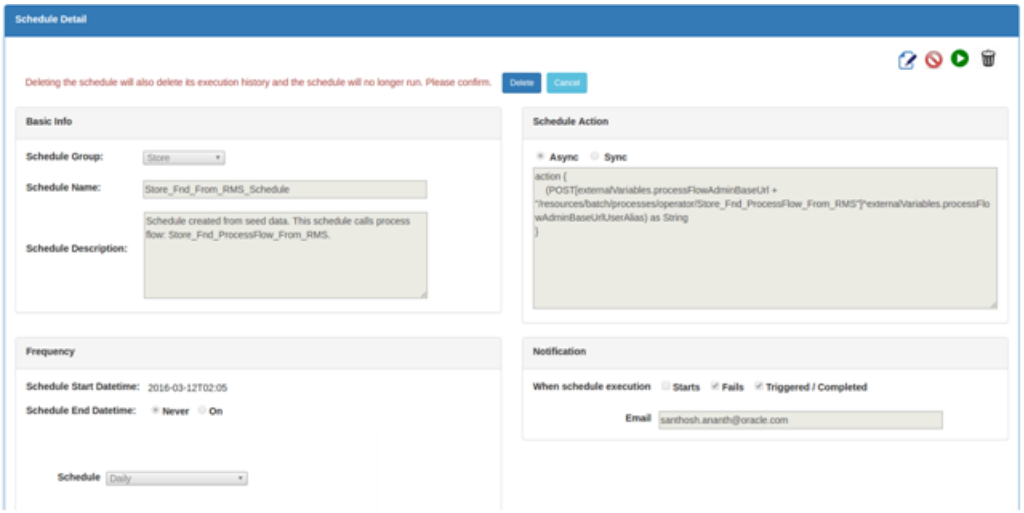



## Deleting a Schedule

A schedule can be deleted using the Delete schedule option in the Schedule Detail view. Only an Admin user 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 5–14 Deleted Schedule Message***



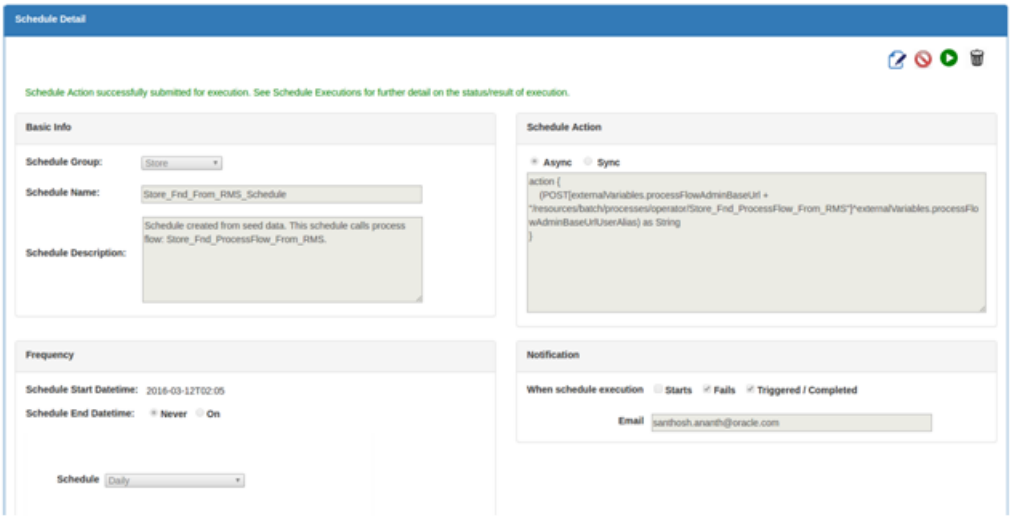
## Schedule a Manual Run

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 execution can be seen from the Schedule Executions view.

***Figure 5–15 On Demand Success Message***

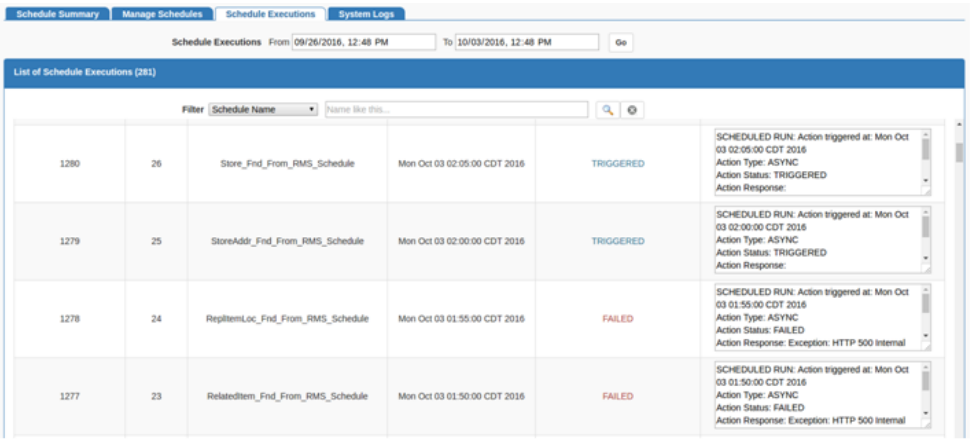



## Schedule Executions

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 obtain the corresponding schedule executions.

Within the list of schedule executions, the records can be filtered based on 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 5–16 List of Schedule Executions*



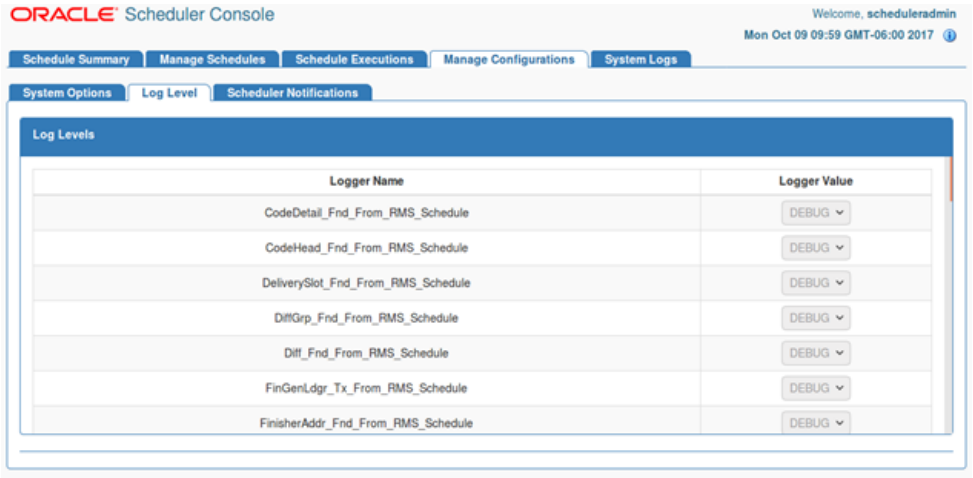

## Manage Configurations

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.

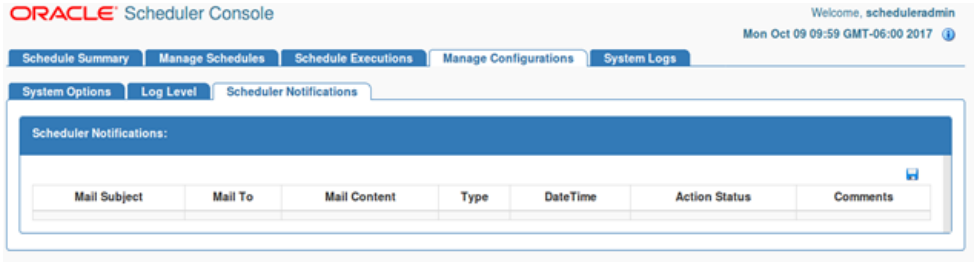
*Figure 5–17 Log Level Page*




### Notifications

Users can view/update notifications details from the Scheduler Notifications page.

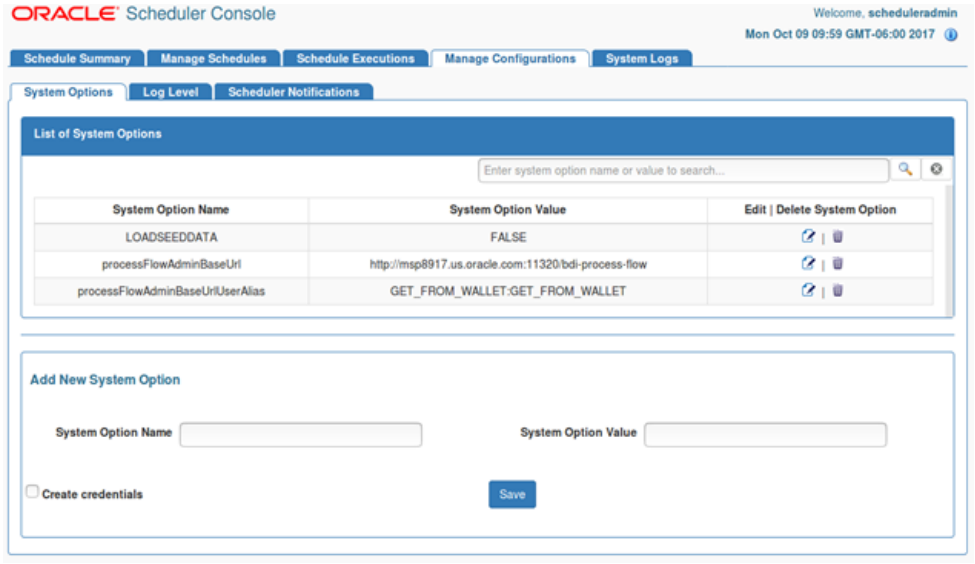
*Figure 5–18 Scheduler Notifications Page*




## System Options

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

*Figure 5–19 System Options Page*

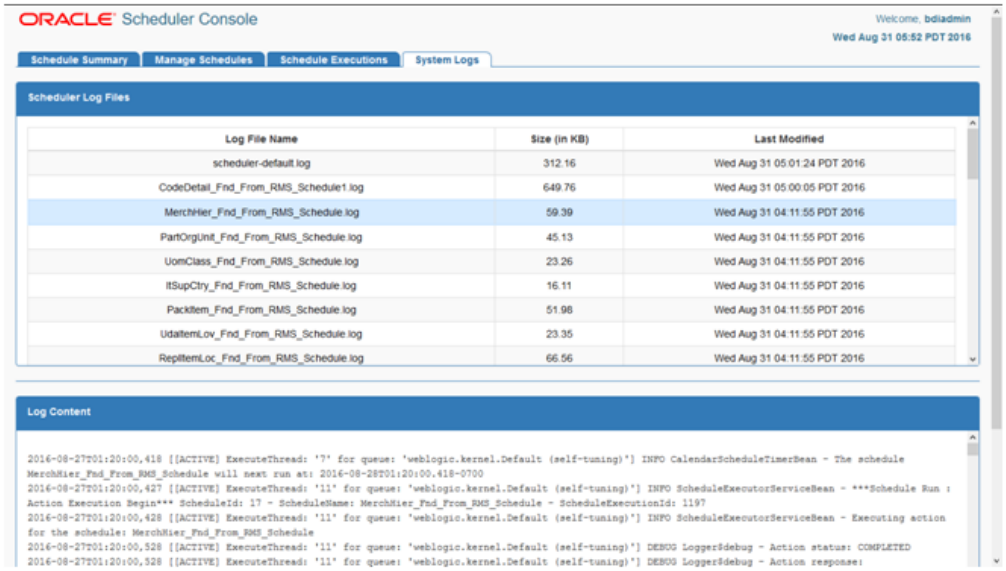



## System Logs

The System Logs page displays 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 last modified time of file, with most recently modified file first.

*Figure 5–20 System Logs*




# Scheduler Security Considerations

This section describes the various 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 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 with a specific role are allowed to access certain functionality in the Scheduler Console.

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

*Table 5–2 Scheduler Functions and Role Permissions*

| Function | Admin Role | Operator Role | Monitor Role |
|---|---|---|---|
| View and search | Yes | Yes | Yes |
| Create schedule | Yes | No | No |
| Edit schedule | Yes | No | No |
| Delete schedule | Yes | No | No |
| Manual run schedule | Yes | Yes | No |
| Disable schedule | Yes | Yes | No |

*Table 5–2 (Cont.) Scheduler Functions and Role Permissions*

| Function | Admin Role | Operator Role | Monitor Role |
|---|---|---|---|
| Enable schedule | Yes | Yes | No |

# Scheduler Operational Considerations

This section describes the various 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 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 the 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 (for example, a connection error when invoking a process flow), the action execution log will be:

```
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 previous action execution log examples indicate async actions. For sync actions, 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.

***Table 5–3 Schedule Log File Keyword Descriptions***

| Keyword | Description |
|---|---|
| ScheduleId | The primary key Id of 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:<br>Action Execution Begin*** and ***Schedule Run: Action Execution End***<br>For manual run, it will be ***Manual Run:<br>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. It is recommended to periodically purge historical schedule executions from the table 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 certain limits and prevent database growth.

# Scheduler Customization

This section describes the various Scheduler customizations.

## Seed Data Reload

The SQL script containing the seed data schedule definitions is located in the *jos-scheduler-home/setup-data/dml* folder.

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

If the Scheduler application must be redeployed and the seed data schedules must be reloaded during the redeployment (that is, to reset the schedules to the initial state as per seed data), set the LOADSEEDDATA column in BDI_SYSTEM_OPTIONS table to TRUE, and un-deploy 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: jos-scheduler-home/setup-data/dml/seed-data.sql**

**Schedule Action dsl files: jos-scheduler-home/setup-data/dsl**

An insert statement for a schedule seed data definition will look like below (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 of these fields (as shown in the previous SQL statement) 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 the 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:

  Must 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 e-mail id for notification (separate multiple e-mails by comma). E-mail is required if a 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 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 DLS 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, 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 Scheduler are the schedules that call the JOS process flows provided out-of-the-box. The Schedule Actions define the REST calls to the JOS 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 Action to invoke the services.

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 follows.

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

The following Schedule Action syntax specifies how a REST service can be called from Scheduler (assuming the REST resource does not require any authentication). The response from the REST service will be treated as a 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 the base URL configured in System Options 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 use the 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 to Appendix D for details on the System Setting REST resources.

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

## Scheduler Troubleshooting

Any failure in schedule execution can be analyzed 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 process flow, then the details of the corresponding process flow execution instance, the exception details, and any stack trace can be viewed in the corresponding process flow logs using Process Flow Admin console for further troubleshooting.

---

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

---

# Scheduler Known Issues

Scheduler Console provides a calendar widget for datetime fields that is supported only by Chrome browser. The latest version of the Chrome browser is recommended for access to 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 a text box. 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.

# 6

# Use Cases

This chapter provides details about the following use cases.

## Creating Job Admin Batch Jobs

The following steps outline the procedure for creating a batch job in Job Admin.

1. Download JosJobAdmin19.0.0ForAll19.x.xApps_eng_ga.zip and unzip the file.
2. Create job XML files using Java batch specification. See the following Job XMLsample.
3. Copy job XML files to jos-job-home/setup-data/META-INF/batch-jobs folder.
4. Copy jar file that contains code related to jobs in jos-job-home/lib folder.
5. Run the deployer script in jos-job-home/bin folder.

### Sample Job XML

Here is sample Job XML that runs the ls shell command.

```
<job id="ShellCommandRunnerBatchlet" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
version="1.0">
    <step id="shellCmd">
        <batchlet ref="ShellCommandRunnerBatchlet">
            <properties>
                <!-- externalCommand format - command param1 .. paramN
                    parameters can be static or dynamic
                    if a parameter is dynamic, then use #SysOpt.paramName.
                    paramName should be setup in BDI_SYSTEM_OPTIONS table
                -->
                <property name="externalCommand" value="ls"/>
                <!-- externalCommandWorkingDir is optional -->
                <property name="externalCommandWorkingDir" value="."/>
            </properties>
        </batchlet>
        <end on="COMPLETE"/>
    </step>
</job>
```

### Passing Job Parameters

Job parameters can be passed to a shell script from the job using the following syntax in the job.

```
#{jobParameters['param1']}

param1 - Name of the parameter

Sample job that uses job parameters

<job id="ShellCommandRunnerBatchlet" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
version="1.0">
    <step id="shellCmd">
        <batchlet ref="ShellCommandRunnerBatchlet">
            <properties>
                <property name="externalCommand" value="ls
#{jobParameters['param1']}  #{jobParameters['param2']}"/>
                <!-- externalCommandWorkingDir is optional -->
                <property name="externalCommandWorkingDir" value="."/>
            </properties>
        </batchlet>
        <end on="COMPLETE"/>
    </step>
</job>
```

If the following parameters are entered in the Job Admin UI during the launching of the above job, the following command will be run by the job.

**Job Parameters: param1=-a,param2=-l**

**Command executed: ls -a -l**

## Passing System Options

System options can be passed to a shell script from the job using the following syntax in the job.

```
#SysOpt.paramName
Sample job that uses system options


<job id="ShellCommandRunnerBatchlet" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
version="1.0">
    <step id="shellCmd">
        <batchlet ref="ShellCommandRunnerBatchlet">
            <properties>
                <property name="externalCommand" value="ls"/>
                <!-- externalCommandWorkingDir is optional -->
                <property name="externalCommandWorkingDir" value="#SysOpt.dir"/>
            </properties>
        </batchlet>
        <end on="COMPLETE"/>
    </step>
</job>
```

If the following system option is set in the Job Admin UI, the following command will be run by the job.

**System Option: dir=/home/batch**

Command executed in the working directory /home/batch.

## Passing System Properties

Java system properties can be passed to a shell script from the job using the following syntax in the job.

```
#{systemProperties['prop1']}
```

Sample job that uses system property

```
<job id="ShellCommandRunnerBatchlet" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
version="1.0">
    <step id="shellCmd">
        <batchlet ref="ShellCommandRunnerBatchlet">
            <properties>
                <property name="externalCommand" value="ls"/>
                <!-- externalCommandWorkingDir is optional -->
                <property name="externalCommandWorkingDir"
value="#{systemProperties['batchDir']}"/>
            </properties>
        </batchlet>
        <end on="COMPLETE"/>
    </step>
</job>
```

If the following system property is set in the JVM for Job Admin, the following command will be run by the job.

**System Property: -DbatchDir=/home/batch**

Command executed in the working directory /home/batch.

## Chaining Multiple Jobs

For running multiple jobs that must run in sequence (single flow), create a DSL to chain the jobs.

### Sample Process Flow

The following process flow runs two jobs, jobA and jobB in sequence.The activity AbcActivty starts jobA by calling a REST endpoint in Job Admin. The activity AbcStatusActivity calls a REST endpoint in Job Admin to check the status of the jobA. It waits until the job is complete or failed. This is a standard pattern for running a batch job. After jobA is complete, the process flow engine runs the jobB.

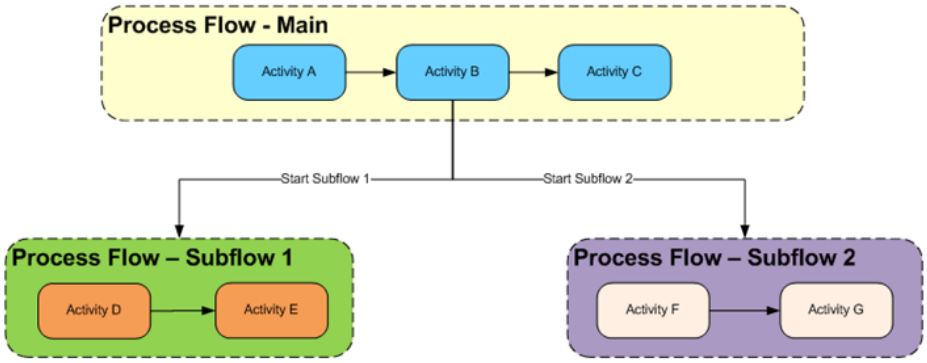
```
process {
name "AbcProcess"
var ([a:"b", c:"d", e: 5])

begin{
    action{
        println "$activityName Load variables"
        println "Access externalVariables=$externalVariables"
        return "okay"
    }
    on "okay" moveTo "AbcActivity"
}

activity{
    name "AbcActivity"
    action{
startOrRestartJob(externalVariables["jobAdminUrl"],"JobA",
externalVariables["jobAdminUrlUserAlias"])
        "okay"
    }
    on "okay" moveTo "AbcStatusActivity"
on "error" moveTo "ErrorActivity"
}
```

```
activity{
    name "AbcStatusActivity"
 action{
waitForJobCompletedOrFailed("AbcActivity",externalVariables["jobAdminUrl"] +
"/resources/batch/jobs/JobA/" + processVariables["jobExecutionId"],
externalVariables["jobAdminUrlUserAlias"])
        "okay"
    }
    on "okay" moveTo "DefActivity"
}

activity{
   name "DefActivity"
    Action{
startOrRestartJob(externalVariables["jobAdminBaseUrl"],"JobB",
externalVariables["jobAdminUrlUserAlias"])
"okay"
    }
    on "okay" moveTo "DefStatusActivity"
}

activity{
    name "DefStatusActivity"
    action{
waitForJobCompletedOrFailed("DefActivity",externalVariables["jobAdminUrl"] +
"/resources/batch/jobs/JobB/" + processVariables["jobExecutionId"],
externalVariables["jobAdminUrlUserAlias"])
"okay"
    }
    on "okay" moveTo "end"
}
 activity{
    name "ErrorActivity"
    action{
        println "$activityName This is error activity"
        return "okay"
    }
    on "okay" moveTo "end"
}

end{
    action{
        println "Got to end"
        return "COMPLETED"
    }
}
}
```

## Creating Split Flows

The main flow must fork other flows. Use the POST method to start a process flow from another process flow.

*Figure 6–1 Split Flows*




## Sample Split Flow

In this sample flow, the activity GhiProcessActivity posts a request to the process flow application to start a new process flow GhiProcess and the main flow continues with rest of the activities. The sub-flow runs independently of the main flow.

### Main Flow

```
process {
name "DefProcess"

begin{
    action{
    }
    on "okay" moveTo "GhiProcessActivity"
}

activity{
    name "GhiProcessActivity"
    action {
(POST[externalVariables.processFlowAdminBaseUrl +
"/resources/batch/processes/operator/ProcessGhi"]
]^externalVariables.processFlowAdminBaseUrlUserAlias)
           "okay"
    }
    on "okay" moveTo "DefActivity"
}

activity{
    name "DefActivity"
    action{
"okay"

    on "okay" moveTo "end"
}

end{
    action{
        return "COMPLETED"
    }
}
}
```

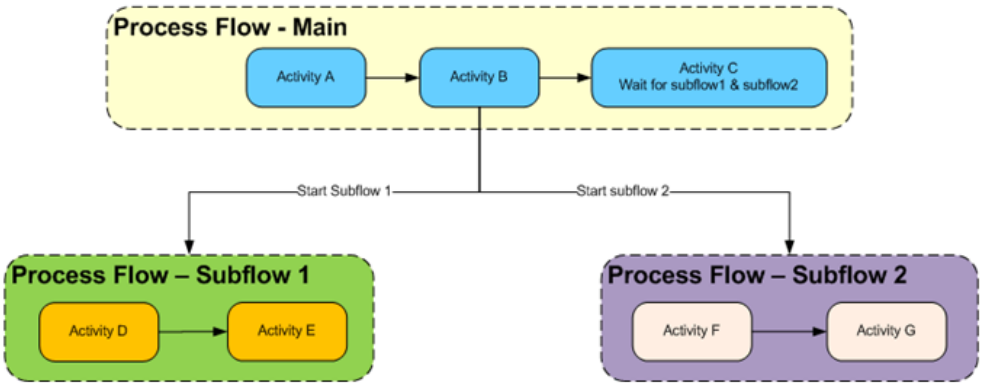
### Sub Flow

```
process {
name "GhiProcess"

begin{
    action{
    }
    on "okay" moveTo "GhiActivity"
}

activity{
    name "GhiActivity"

    action{
//do something here
    }
    on "okay" moveTo "end"
}

end{
    action{
        return "COMPLETED"
    }
}
}
```

## Creating Split and Join Flows

Process flow Abc starts process flow Def and Xyz. Process flow Abc must wait until Def and Xyz process flows are complete. The activity AbcActivity waits until DefProcess and XyzProcess are complete. Use waitForProcessInstancesToReachStatus method to wait for other flows to complete.

***Figure 6–2 Split and Join Flows***




## Sample Split and Join Flow

```
process {
name "AbcProcess"

begin{
    action{
"okay"
```

```
    }
    on "okay" moveTo "DefAndXyzActivity"
}

activity{
    name "DefAndXyzActivity"
    action {
def defExecution = ((POST[externalVariables.processFlowAdminBaseUrl +
"/resources/batch/processes/operator/ProcessDef"]
]^externalVariables.processFlowAdminBaseUrlUserAlias) as
ProcessExecutionIdsVo.ProcessExecutionIdVo)
processVariables[‘processDefExecution’] = defExecution.executionId
def xyzExecution = ((POST[externalVariables.processFlowAdminBaseUrl +
"/resources/batch/processes/operator/ProcessXyz"]
]^externalVariables.processFlowAdminBaseUrlUserAlias) as
ProcessExecutionIdsVo.ProcessExecutionIdVo)
processVariables[‘processXyzExecution’] = xyzExecution.executionId
           "okay"
    }
    on "okay" moveTo "AbcActivity"
}

activity{
    name "AbcActivity"
    Action{
waitForProcessInstancesToReachStatus([processVariables[‘processDefExecution’],
processVariables[‘processXyzExecution’]], PROCESS_COMPLETED, LOGICAL_AND)
"okay"
    }
    on "okay" moveTo "end"
}

end{
    action{
        println "Got to end"
        return "COMPLETED"
    }
}
}
```

## DefProcess Flow

```
process{
name "DefProcess"

begin{
    action{

    }
        on "okay" moveTo "defActivity"
}

activity{
    name "defActivity"
    action{
//do something here
}
                              on "okay" moveTo "end"
}
```

```
end{
    action{
        "COMPLETE"
    }
}
}
```

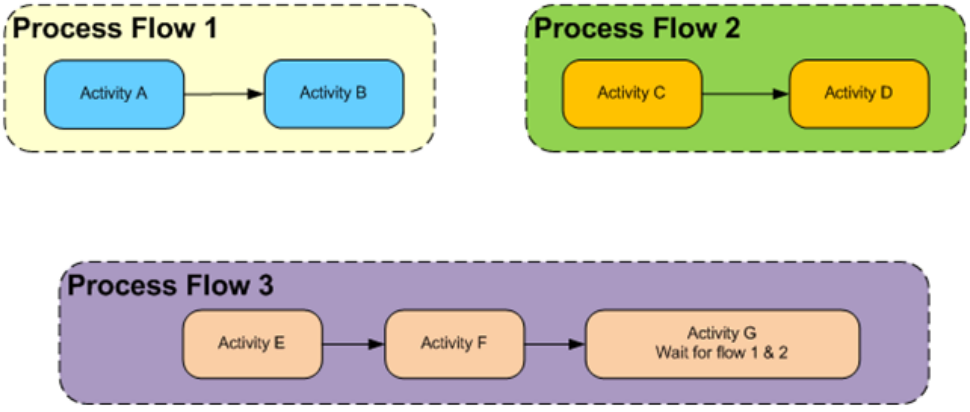
### XyzProcess Flow

```
process{
name "XyzProcess"

begin{
    action{

    }
        on "okay" moveTo "xyzActivity"
}

activity{
    name "xyzActivity"
    action{
    //do something here
}on "okay" moveTo "end"
}

end{
    action{
        "COMPLETE"
    }
}
}
```

## Creating a Join Flow with Other Flows

Process flow Def and Process flow Xyz run independently. Process flow Abc has to wait until process Def and Xyz are complete. Use waitForProcessNamesToReachStatus to wait for other processes to complete.

***Figure 6–3 Join Flows with Other Flows***



### Sample Join Flow

```
process {
name "AbcProcess"

begin{
    action{
    }
    on "okay" moveTo "AbcActivity"
}

activity{
    name "AbcActivity"
    action{
waitForProcessNamesToReachStatus(['DefProcess':2, 'XyzProcess':2],
now().minusDays(1), PROCESS_COMPLETED, LOGICAL_AND, LAST_EXECUTION_STATUS)
"okay"
    }
    on "okay" moveTo "end"
}

end{
    action{
        return "COMPLETED"
    }
}
}
```

## Sharing Data Between Process Flows

Process flow Abc must share data with process flow Def.

Use persistGlobalUserData and findGlobalUserData APIs to share information.

### Sample Flow that Shares Information with Other Flows

```
process {
name "AbcProcess"
begin{
    action{
    }
    on "okay" moveTo "AbcActivity"
}
activity{
    name "AbcActivity"
    action{
// Persist date as String
persistGlobalUserData("workDayStart", now().minusDays(1).toString())
"okay"
    }
    on "okay" moveTo "end"
}
end{
    action{
        return "COMPLETED"
    }
}
}

process {
```

```
name "DefProcess"
begin{
    action{
    }
    on "okay" moveTo "DefActivity"
}
activity{
    name "DefActivity"
    action{
/fetch the date from db
def workDayStartString = findGlobalUserData("workDayStart")
LocalDateTime workDayStartDateObject = LocalDateTime.parse(workDayStartString)
log.debug "WorkDayStart Global data asString(${workDayStartString}) and
asLocalDateTime(${workDayStartDateObject})"
"okay"
    }
    on "okay" moveTo "end"
}
end{
    action{
        return "COMPLETED"
    }

}
```

# Creating Schedules in Scheduler

Complete the following steps to create a schedule in Scheduler.

1. Download JosScheduler19.0.0ForAll19.x.xApps_eng_ga.zip and unzip the file.
2. Set up the schedule for the above created process through seed data or the UI. See the following seed data sample.
3. Create DSL file for action. DSL is Groovy based and Groovy or Java code can be used in Action block. See the following DSL sample.
4. Copy DSL file to jos-scheduler-home/setup-data/dsl folder.
5. Run the deployer script from jos-scheduler-home/bin folder.

## Using Sample Seed Data to Create a Schedule

Here are important fields in seed data that must be considered for the schedule being created.

- schedule_type - SIMPLE. if advanced scheduling is required, it must be created using Scheduler UI.
- schedule_start_datetime - Specify the date and time when to start the schedule, for example, '2016-11-22 10:20:00'
- schedule _frequency - Valid values are: DAILY, HOURLY, WEEKLY, MONTHLY, WEEKDAY, WEEKEND, SATURDAY, SUNDAY, FIRSTDAYOFMONTH, LASTDAYOFMONTH, ONCE
- schedule_action_type - ASYNC (asynchronous) or SYNC (synchronous)
- schedule_action_definition - Name of the schedule action DSL file

```
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 (1, 'Schedule1',
'Schedules', 'Schedule created from seed data. This schedule calls process flow:
AbcProcess.',
'ACTIVE', TIMESTAMP '2016-11-22 00:00:00', 'SIMPLE', 'DAILY', 'ON_SUCCESS,ON_
ERROR',
'admin@example.com', 'ASYNC', 'Abc.sch')
```

### Scheduling an Action DSL

Each schedule has a corresponding schedule action DSL. This will be the action that is executed when the schedule runs.

### Sample Action DSL

The following schedule action starts AbcProcess flow by sending a POST request to Process Flow.

```
action {
(POST[externalVariables.processFlowAdminBaseUrl +
"/resources/batch/processes/operator/AbcProcess"]^externalVariables.processFlowAdm
inBaseUrlUserAlias) as String
```

# 7

# Pre-Implementation Considerations

This chapter describes the pre-implementation considerations.

## Thread Pool Size in WebLogic

If many concurrent schedules/process flows/jobs are going to run, increase the thread pool size in WebLogic. This value can be changed for a managed server from the WebLogic Admin Console.

- Servers -> Server Name -> Tuning -> Advanced -> Self Tuning Thread Maximum Pool Size

## Database Connection Pool Size in WebLogic

If many concurrent jobs are going to run, increase the maximum capacity of the connection pool for the data sources that are associated with the jobs. The default value is 15. This value can be changed from the WebLogic Admin Console.

- Services -> Data Sources -> DataSource -> Connection Pool -> Maximum Capacity

# 8

# High Availability Considerations

This chapter provides information about high availability considerations.

## About High Availability

Modern business application requirements are classified by the abilities that the system must provide. This list of abilities, such as availability, scalability, reliability, scalability, 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 are addressed without having to do much development work within the business application. Clustering directly addresses availability, scalability, and recoverability requirements, which are very attractive to a business. In reality though it is a trade off, as clustered systems increase complexity and are normally more difficult to manage and secure, and so one should evaluate the pros and cons before deciding to use clustering.

Oracle provides many clustering solutions and options; those relevant to JOS 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.

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**

# Scaling JOS

JOS needs to be scaled horizontally to handle a large number of concurrent jobs. Single instances of Scheduler and Process Flow can be used since they are not resource intensive. JOS Admin can be very resource intensive. To handle large number of concurrent jobs, multiple instances of JOS Admin can be used to distribute jobs. WebLogic Server cluster that consists of multiple managed server instances provide horizontal scalability for JOS Admin.

# JOS on Cluster

As recommended above, for scaling JOS for large number of jobs, JOS components should be deployed to cluster. Following are some considerations to be taken into account when deploying JOS on a cluster.

## Logging

### Issue

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

### Solution

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

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

**$DOMAIN_HOME/logs/server1/jos-rms-job-admin._war**
**$DOMAIN_HOME/logs/server2/jos-rms-job-admin._war**

1. Create a common log directory (for example; /home/logs/jobadmin) for each JOS 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/jobadmin
       server1/jos-rms-job-admin._war
ln -s /home/logs/jobadmin
       server2/jos-rms-job-admin._war
```

3. If the directory $DOMAIN_HOME/logs/<server>/<app> already exists, it must be deleted before symbolic link is created.
4. The application must 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 the 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 must be updated through the URL of all the nodes in the cluster using UI or REST endpoint.

**Example**

```
http://server1:port1/jos-rms-batch-job-admin/system-setting/system-logs
http://server2:port2/jos-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 must be invoked on the other nodes in the cluster for that application in JOS.

**Example**

```
http://server1:port1/jos-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 must be invoked on all the nodes in the cluster for that application in JOS.

### Example

```
http://server1:port1/jos-rms-batch-job-admin/system-setting/system-credentials
http://server2:port2/jos-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>
```

# 9

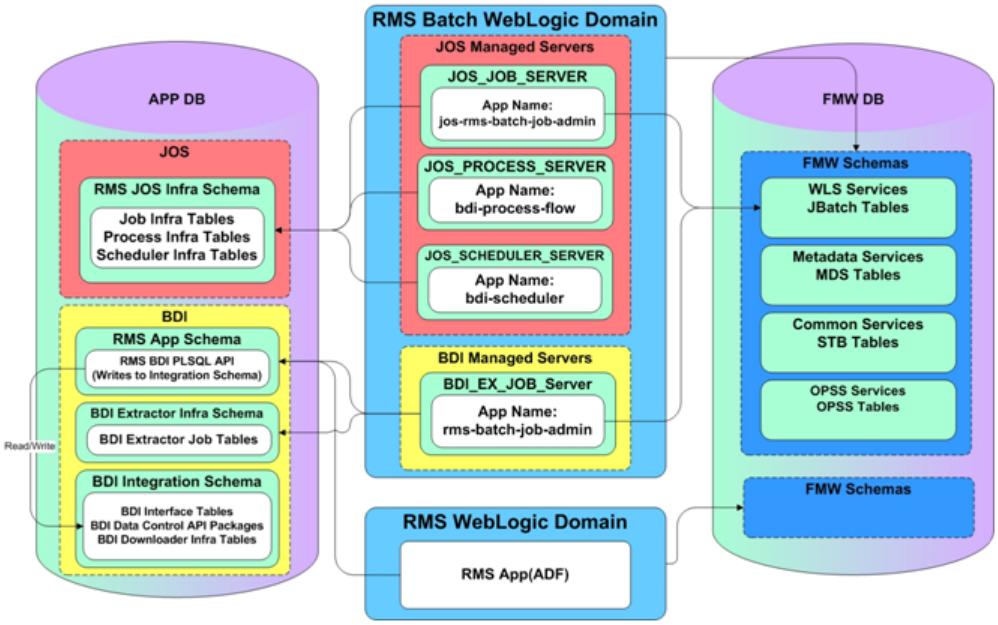
# Deployment Architecture

This chapter provides information about the following deployment architectures.

## JOS and BDI Deployment Architecture for RMS

This diagram shows the recommended deployment architecture for RMS that uses both JOS and BDI. Here JOS and BDI use the same batch schema as they are deployed in the same WebLogic domain. However, they use different infrastructure schemas.

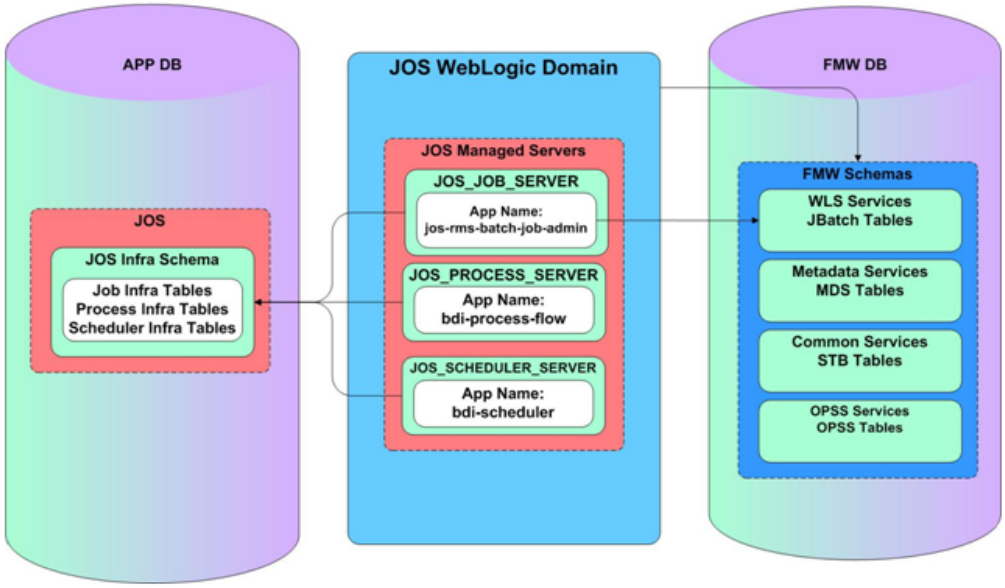
***Figure 9–1 RMS JOS and BDI Deployment Architecture***




## JOS Deployment Architecture

This diagram shows a simple deployment architecture for JOS. In this architecture, JOS Job Admin, JOS Process Flow, and JOS Scheduler are deployed in separate managed servers in a WebLogic domain. This is the recommended architecture if batch jobs are simple and not resource intensive.

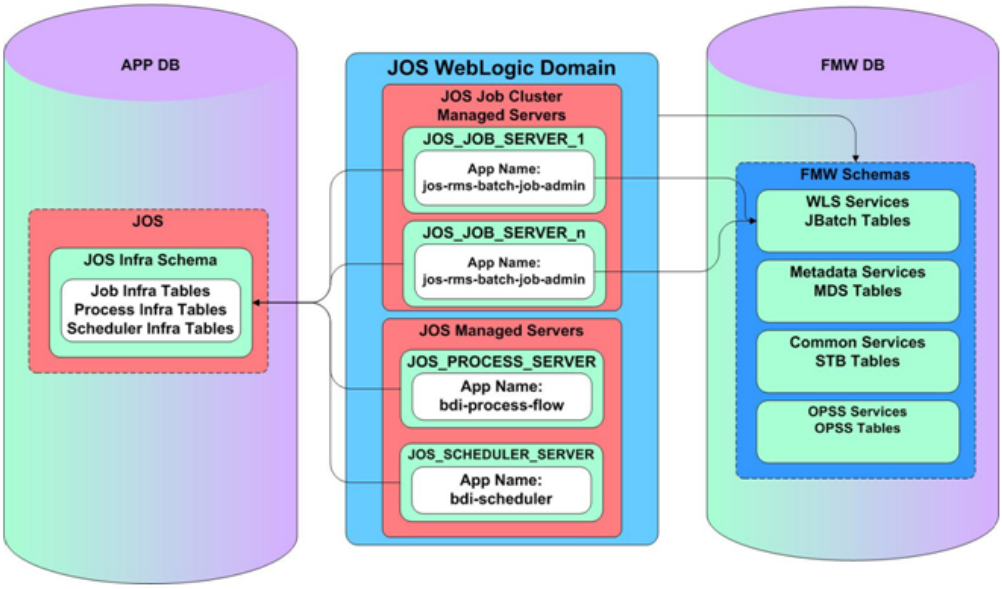
*Figure 9–2 JOS Deployment Architecture*




# JOS Scalable Deployment Architecture

This diagram shows scalable deployment architecture for JOS. In this architecture, Job Admin is deployed in multiple managed servers in a cluster. Process Flow and Scheduler are deployed in their own managed servers. This is the recommended architecture if batch jobs are resource intensive. This architecture allows Job Admin to be scaled horizontally and jobs can be distributed.

*Figure 9–3 JOS Scalable Deployment Architecture*



# 10

# Performance Considerations

This chapter describes performance considerations.

## CPU and Memory Considerations

The following is a list of CPU and memory considerations.

- As JOS application memory requirements are low, 1 GB should be sufficient. If you are running shell scripts from JOS, you must ensure that whatever memory is required by your scripts is available in the machine.
- CPU depends on the number of concurrent jobs you plan to run. If you plan to run many process flows concurrently, you must allocate at least that many threads to the WebLogic thread pool.
- JavaBatch automatically throttles concurrent jobs based on how many threads are available to the process.

# 11

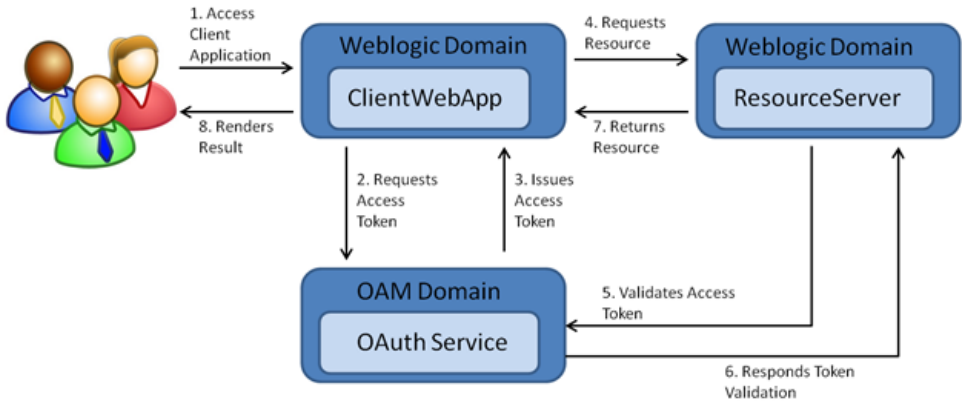
# OAuth 2.0

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 11–1 OAuth 2.0 Architecture Diagram***




## OAuth 2.0 Concepts

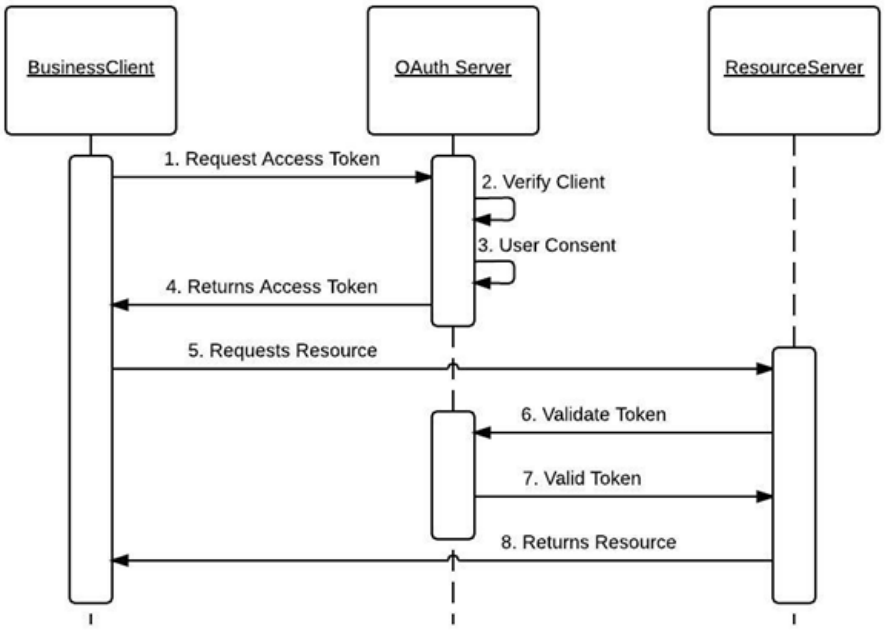
**Business to Business (2-legged flow)**

- 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 11–2 OAuth 2.0 Use Case Flow***



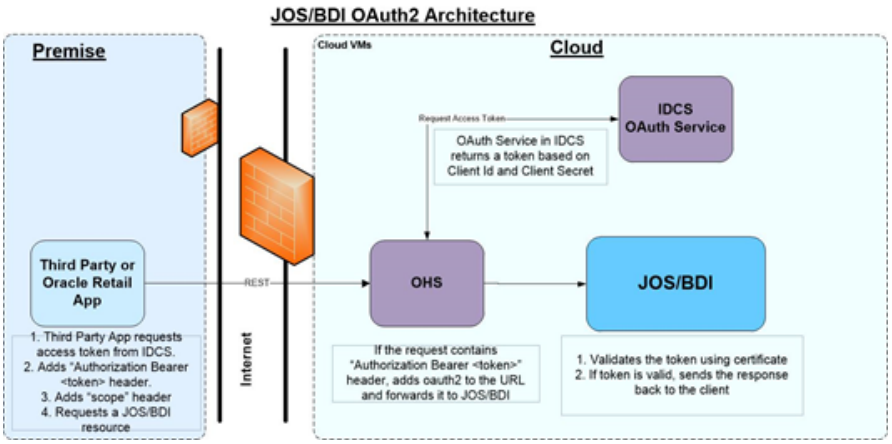

## OAuth 2.0 Terms

- 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.

## JOS OAuth 2.0 Architecture

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 11–3 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=bdi-process-flow,jos-rms-batch-job-admin

oauth-configuration-env-info.oauth-resource-server-interface.bdi-process-flow.scopeName=OperatorAccessScope

oauth-configuration-env-info.oauth-resource-server-interface.jos-rms-batch-job-admin.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.

Use the *OAuth 2.0 Installation Guide* to configure OHS.

## 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-19.0.0.jar
- oauth2-service-provider-api-19.0.0.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 OAuth2ServiceConsumer19.0.0ForAll19.0.0Apps_eng_ga.zip.
2. Unzip the downloaded archive. The "oauth2-consumer-home" directory will be created under the current directory.

   ```
   Unzip OAuth2ServiceConsumer19.0.0ForAll19.0.0Apps_eng_ga.zip
   ```

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

   - ./conf/oauth2-service-consumer-config.properties
   - ./lib/oauth2-common-19.0.0.jar
   - ./lib/oauth2-service-consumer-api-19.0.0.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 11–1 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<br>(scope - AdminAccessScope, OperatorAccessScope, MonitorAccessScope) |
| oauth2.default.scopeOfAccess.headers | headers.<scope> for default server<br>(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-19.0.0.jar
- oauth2-service-consumer-api-19.0.0.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
```

See "Consumer Configuration" on page 11-5 to find various scopes of accesses.

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":{
}
```

# A

# Appendix A: Scheduler REST Endpoints

The Scheduler provides RESTful services to retrieve information about schedules and to run the scheduler manually. The endpoint discover can be used to identify all endpoints provided by the Scheduler.

## REST Resource Descriptions

The following table describes the REST resources.

***Table A–1 REST Resource Descriptions***

| REST Resource | 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 one 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 |

***Table A–1 (Cont.) REST Resource Descriptions***

| REST Resource | Method | Description |
|---|---|---|
| /batch/schedules/executions/past/days/{days} | GET | Returns schedule executions for last n days |
| /batch/schedules/operator/run-schedule-now/{scheduleName} | GET | Runs the specified schedule, that is, executes the Schedule Action of the schedule and returns the Schedule Execution detail response.<br>This is a synchronous invocation, so client must wait for the response. |
| /batch/schedules/executions/time/{fromDateTime}/{toDateTime} | GET | Returns schedule executions between from and to time |

# B

# Appendix B: Process Flow REST Endpoints

The table in this appendix provides a description of the REST resources.

***Table B–1 REST Resource Descriptions***

| 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 Activity execution detail for the activity specified |

*Table B–1 (Cont.) REST Resource Descriptions*

| REST Resource | HTTP Method | Description |
|---|---|---|
| /batch/processes/executions/{processName}/{processExecutionId} | GET | Get all the activities for the process flow execution |
| /batch/processes/{processName}/activities | GET | Get all the activities for the process specified |
| /batch/processes/operator/{processName}/{processExecutionId} | POST | Restart a process execution instance |
| /batch/processes/operator/{processName}/resolve | POST | Sets the status of process to PROCESS_FAILED |
| /batch/processes/{processName}/{processExecutionId} | DELETE | Stops running process |
| /batch/processes/executions | DELETE | Stops all running processes |
| /telemetry/processes | GET | Returns process runtime metrics between fromTime and toTime |

# C

# Appendix C: 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.

***Table C–1 REST Resource Descriptions***

| 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 |
| /batch/jobs/executions/{jobExecutionId}/steps | GET | Gets execution steps |
| /batch/jobs/executions/{jobExecutionId}/steps/{stepExecutionId} | GET | Gets step details |

*Table C–1 (Cont.) REST Resource Descriptions*

| REST Resource | HTTP Method | Description |
|---|---|---|
| /batch/jobs/job-def-xml-files | GET | Gets all job XML files |
| /telemetry/jobs | GET | Returns runtime job metrics between fromTime and toTime |
| 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 |

# D

# Appendix D: System Setting Service

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

***Table D–1 REST Resource Descriptions***

| 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 | Get 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 | Gets system credentials. Only admin user is allowed to perform this operation. |
| /system-setting/system-credentials | POST | Gets system credentials. Only admin user is allowed to perform this operation. |

*Table D–1 (Cont.) REST Resource Descriptions*

| REST Resource | HTTP Method | Description |
|---|---|---|
| /system-setting/system-options/system-credentials | PUT | Creates system options and corresponding credentials |
| /system-setting/system-options/system-credentials | POST | Updates system options and corresponding credentials |
| /system-setting/system-options/system-credentials/{key} | DELETE | Deletes system options and corresponding credentials |
| /system-setting/system-logs | POST | Updates system log level |
| /system-setting/reset-cache | POST | Resets system option cache |
| /system-setting/system-credentials/{key} | DELETE | Deletes system credentials. Only admin user is allowed to perform this operation. |

# Managing System Options Using Curl

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

## Creating System Options

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

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

-d '{"key":"reimappJobAdminBaseUrlUserAlias" , "value":" GET_FROM_
WALLET:GET_FROM_WALLET "}'
```

## Updating System Options

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

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

-d '{"key":"reimappJobAdminBaseUrl" ,
"value":"http://server:port/reim-batch-job-admin"}'
```

## Deleting System Options

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

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

-d '{"key":"reimappJobAdminBaseUrl"}'
```

### Resetting System Options Cache

This command resets the cache for the system options, and it must be run on all the nodes to refresh the cache.

```
curl --user userId:password -i -X POST
http://server:port/bdi-process-flow/resources/system-setting/reset-cache
```

### Listing System Options

This command lists all the system options from the 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 the Process Flow. These commands can be run for Job Admin and Scheduler as well. Create/update/delete commands can only be run by an administrator.

### Creating Credentials

This command creates a credential in the Process Flow.

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

### Updating Credentials

This command updates a credential in the Process Flow.

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

### Deleting Credentials

This command deletes a credential from the Process Flow.

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

### Listing Credentials

This command lists credentials from the Process Flow.

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

# E

# Appendix E: Scheduler UI Screenshots

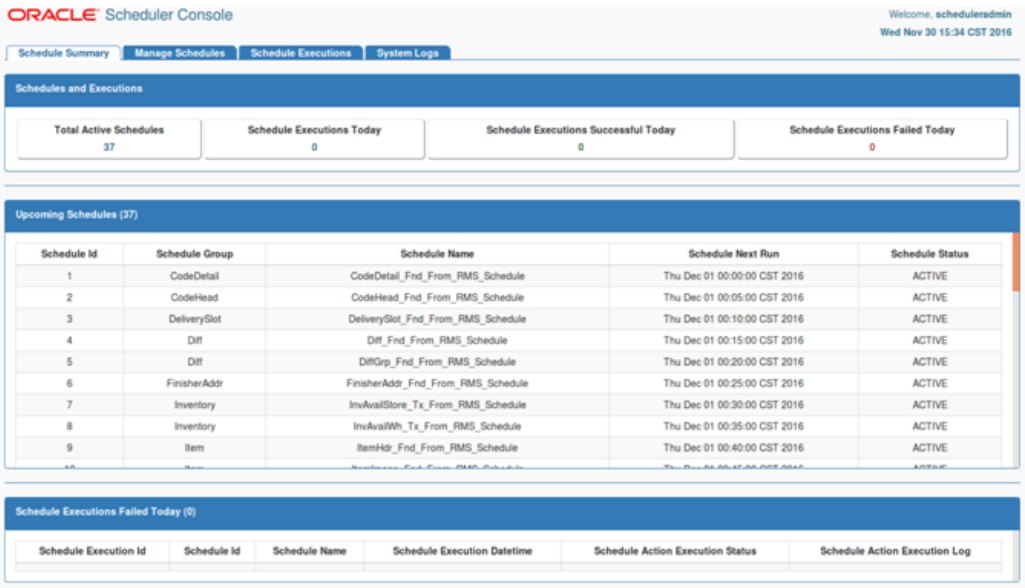
The screenshots in this appendix are part of the Scheduler User Interface.

## Scheduler Summary

This is the home page that provides the overall summary of the scheduler runtime.

> **Note:** The term *today* indicates the duration from midnight to now. It lists the future schedules that are expected to run in the next 24 hours from now. It also lists the schedule executions that have failed today (from midnight to now).

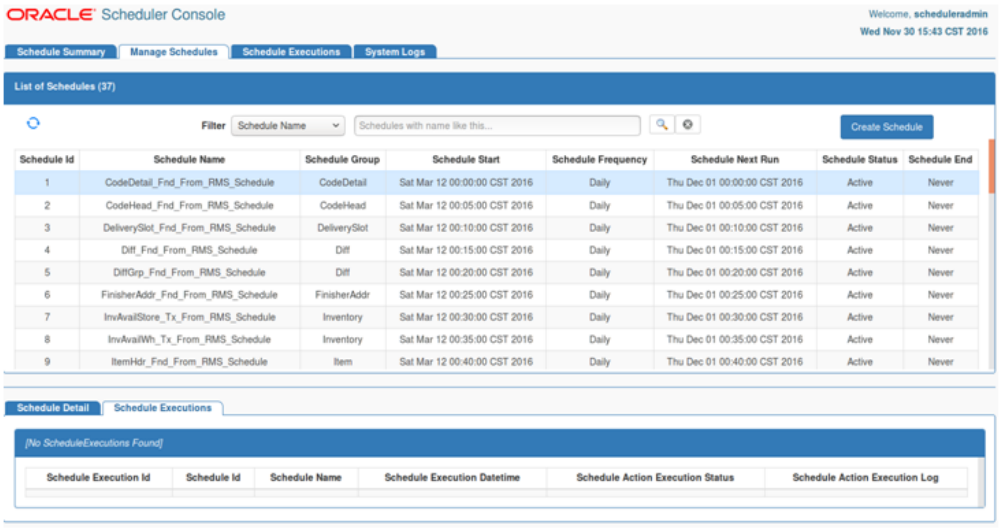
***Figure E–1 Scheduler Console Schedule Summary Tab***




## Manage Schedules - Schedule Executions

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

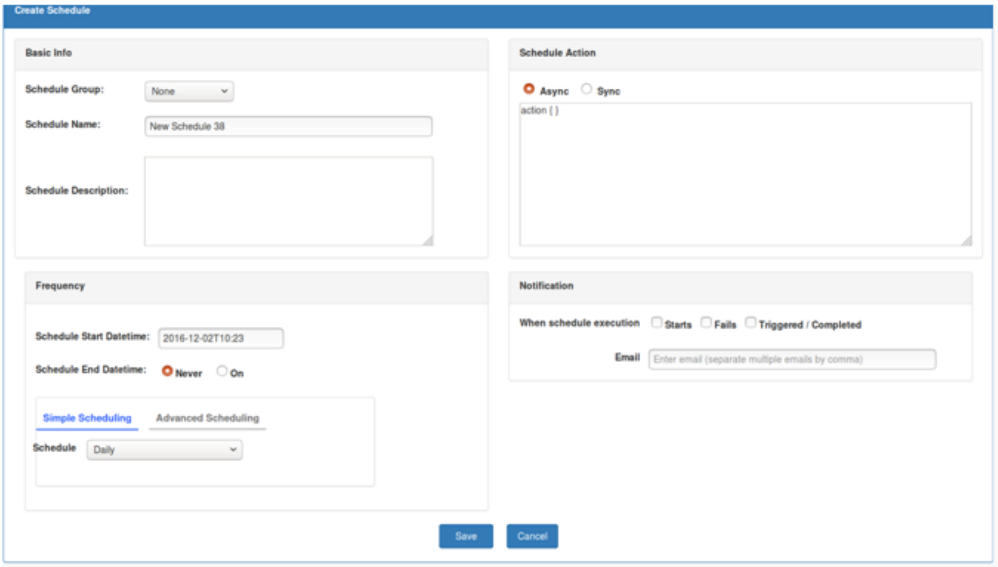
*Figure E–2 Scheduler Console Manage Schedules Tab*




## Manage Schedules - Create Schedule

The Create Schedule option displays one page where the user can enter and save all required information to create a schedule.

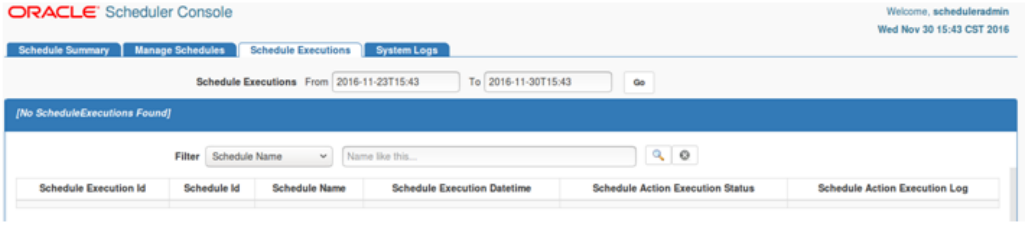
*Figure E–3 Create Schedule Screen*




# 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.

*Figure E–4 Schedule Executions Screen*



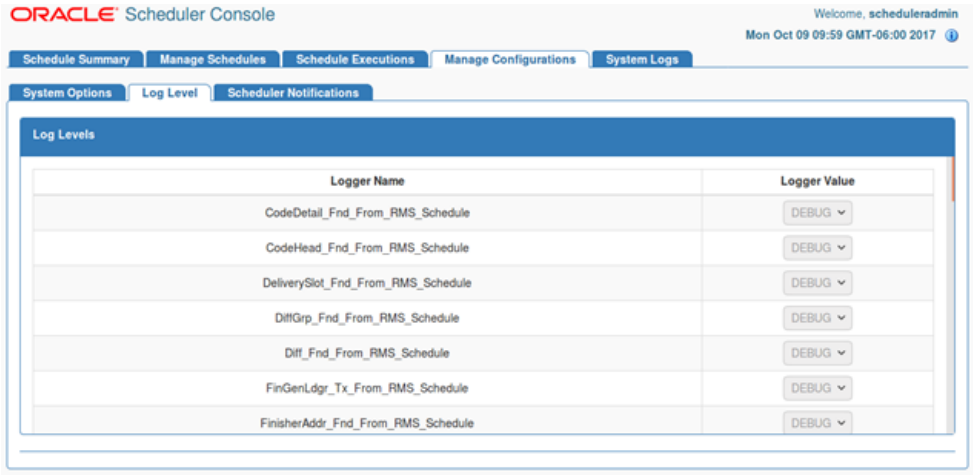

# Manage Configurations

Manage Configurations page allows the user to view/edit configuration for log levels, notifications, and system options.

## Log Level

Users can view log levels for all schedules. This page allows user to change log levels.

*Figure E–5 View Log Levels*

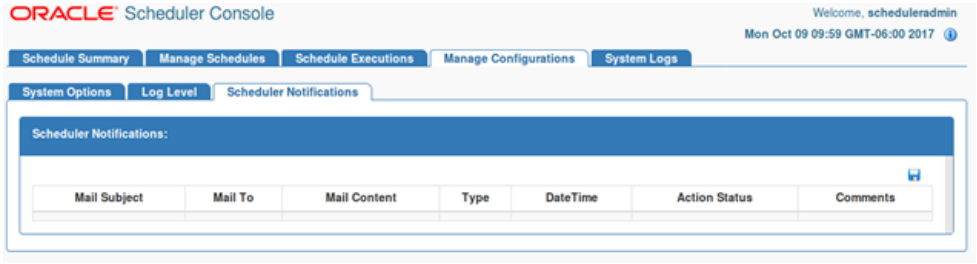



Build version and date is displayed on the info icon when the user selects the same. The icon is on the extreme right top corner of the page.

## Notifications

Users can view/edit notification details from the Notifications page.

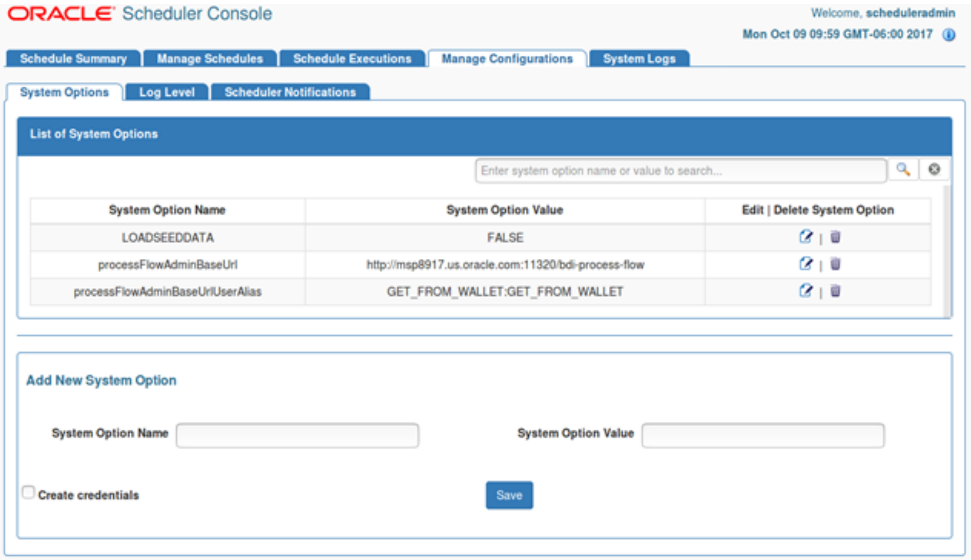
***Figure E–6 Notifications Page***




## System Options

Users can view/create/update/delete system options from this page. It also allows users to create credentials along with system options.

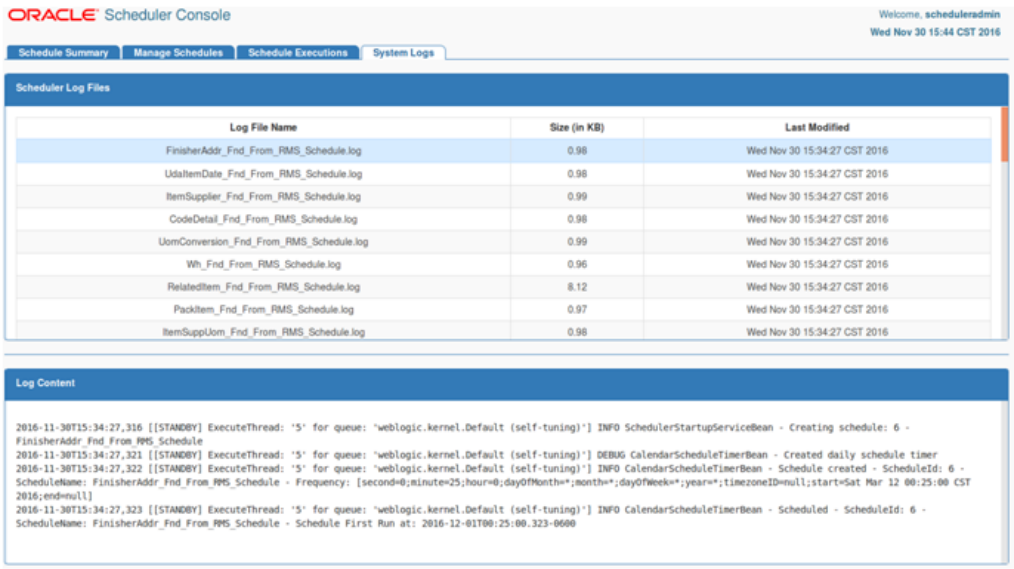
***Figure E–7 System Options Page***




# System Logs

The System Logs page displays a list of all the schedule log files and the 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.

***Figure E–8 Scheduler Console System Logs Tab***



# F

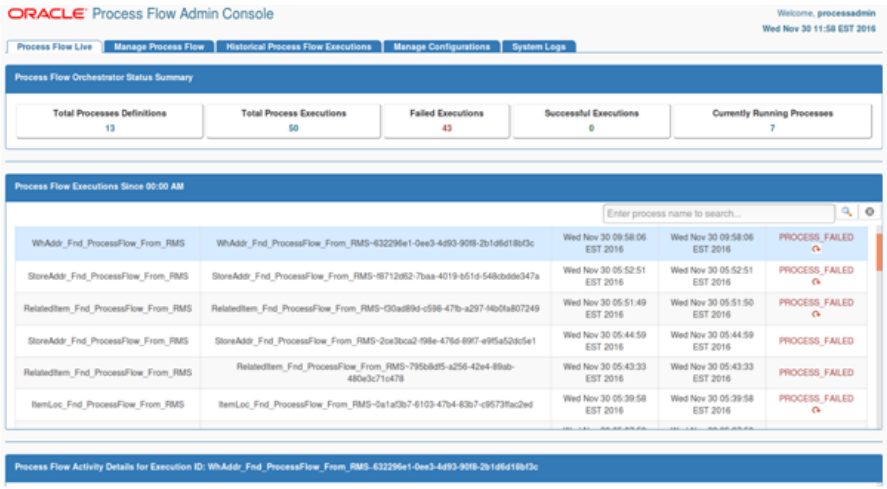
# Appendix F: Process Flow UI Screenshots

The screenshots in this appendix are part of the Process Flow User Interface.

## About Process Flow Live

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.

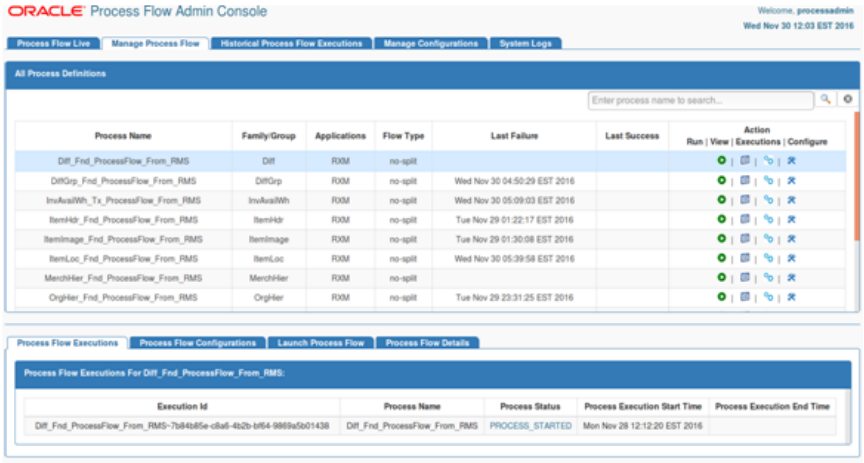
***Figure F–1 Process Flow Live Tab***




## About Manage Process Flow - Process Flow Executions

The Manage Process Flow tab is used to start a process flow, restart a failed process flow, view/edit a process flow, list the executions instances of a process flow, and view/edit the process flow configuration. A failed process flow instance can be restarted only if it is the latest failed instance and there are no successful executions after that.

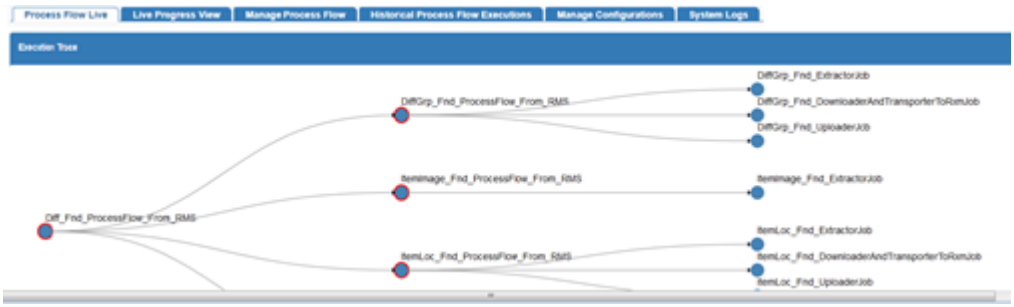
*Figure F–2 Process Flow Executions Screen*




# Execution Trace Graph

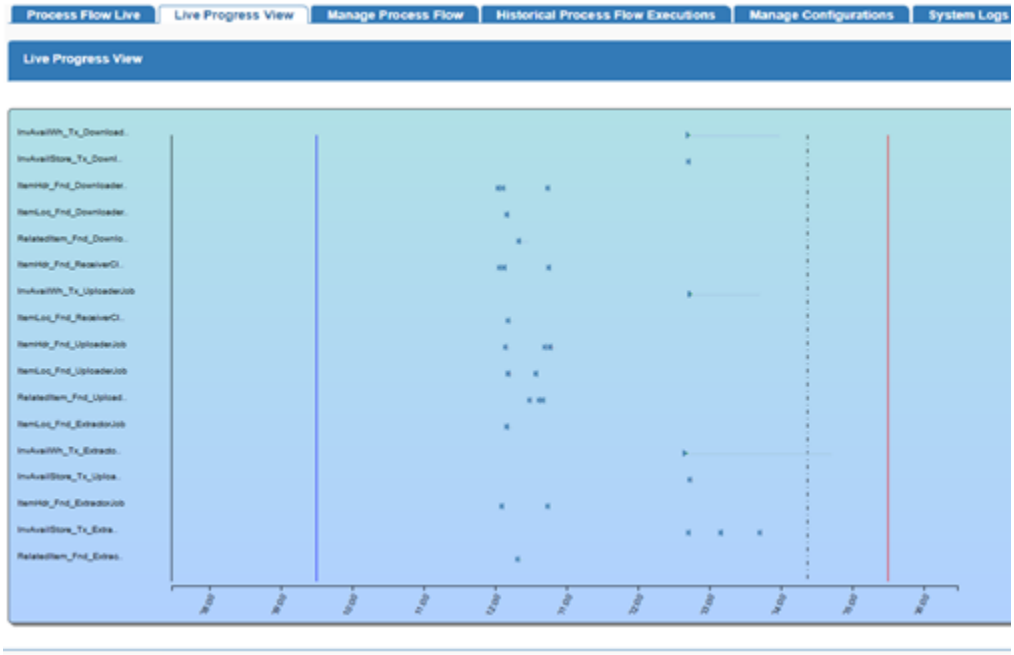
The Execution Trace Graph is also part of Process Flow Live tab. Execution trace graph shows the sub processes and jobs called from a process. The arrows show the relationship between the caller and the callee. The circular nodes of the graph represent the process or the job that was invoked. On hovering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

*Figure F–3 Execution Trace Graph*



# Live Progress View Tab

***Figure F–4 Live Progress View Tab***



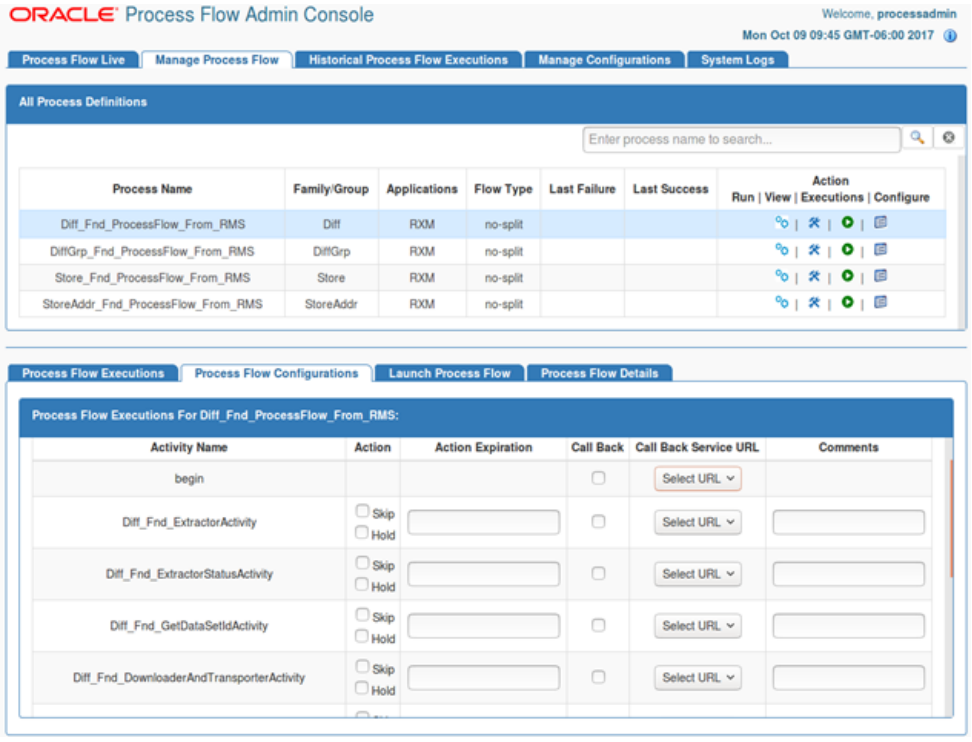

Live progress graph shows the batch status for a time window. Here are the details of the graph.

- 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 - Process Flow Configurations

The Process Flow Configurations tab is used to set skip/hold flags on activities in a flow. Call back flag and url can be selected from this page.

***Figure F–5 Process Flow Configuration***

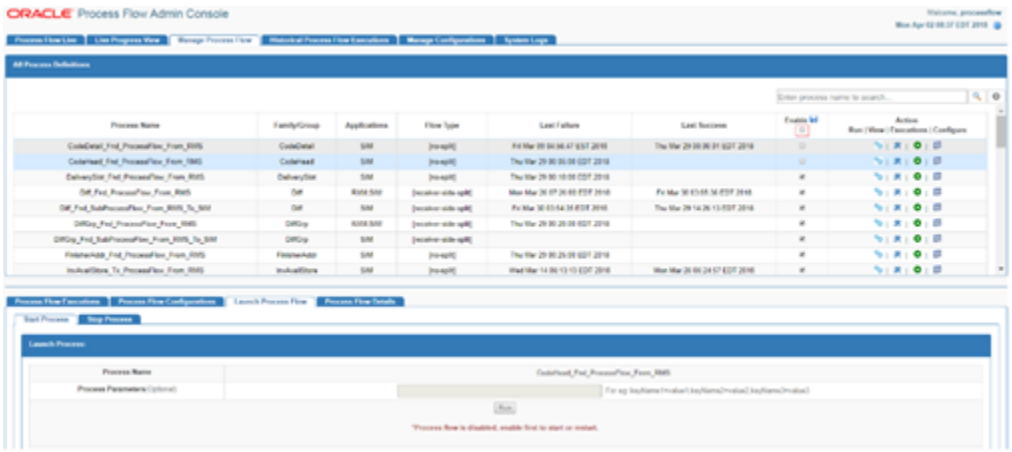



Build version and date is displayed on the info icon when the user selects the same. The icon is on the extreme right top corner of the page.

# Manage Process Flow - Launch Process Flow

The Manage Process Flow tab allows the user to Start a process flow, Restart a failed process flow, enable or disable the process flow etc. 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.

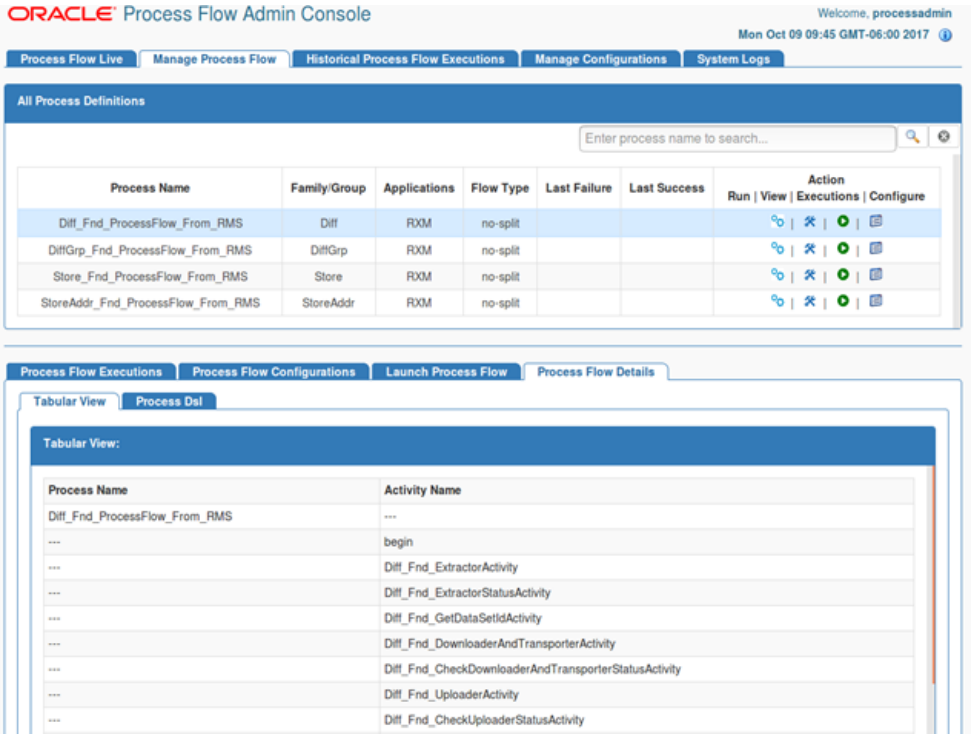
***Figure F–6 Launch Process Flow Screen***




# Manage Process Flow - Process Flow Details - Process Details

The Process Details tab displays process activities and configuration in a tabular form.

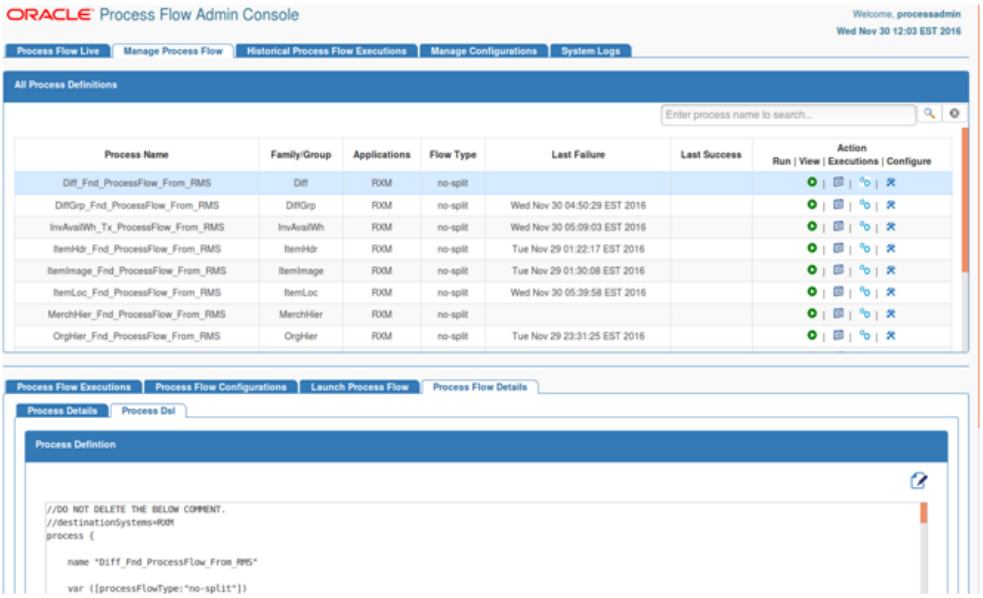
***Figure F–7 Process Details Tab***




# Manage Process Flow - Process Flow Details - Process DSL

The Process DSL tab displays DSL for the selected process flow.

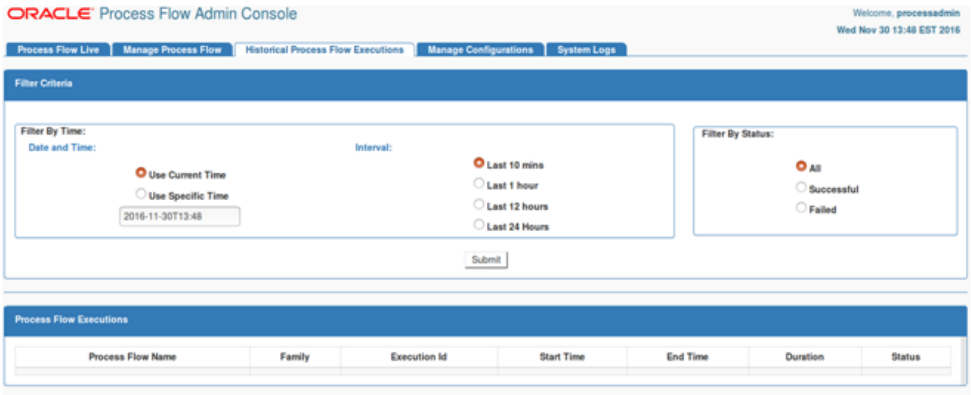
*Figure F–8 Process DSL Tab*




# Historical Process Flow Executions

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 a 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.

*Figure F–9 Historical Process Flow Execution Tab*




# Managing Configurations

The Manage Configurations tab allows to manage system options, log levels, and process notifications.

## 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. Admin Role, Operator Role and Monitor Role

***Figure F–10 PING with Success Message***

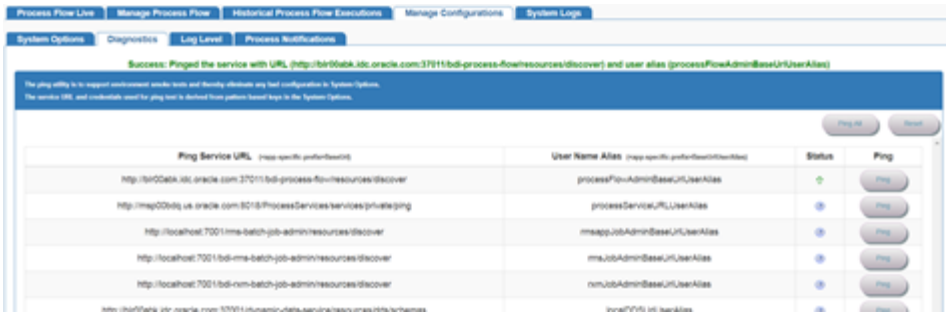



***Figure F–11 PING with Failure Message***

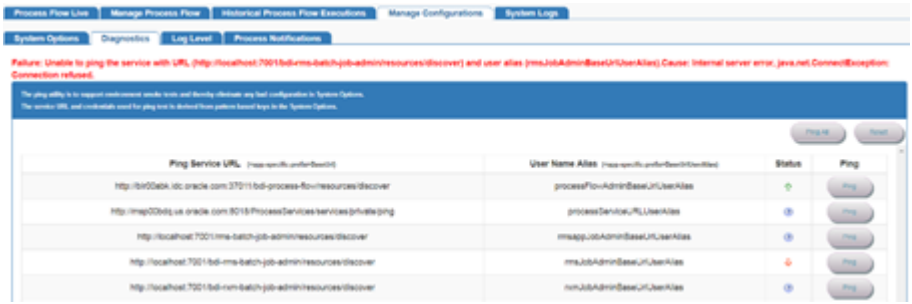
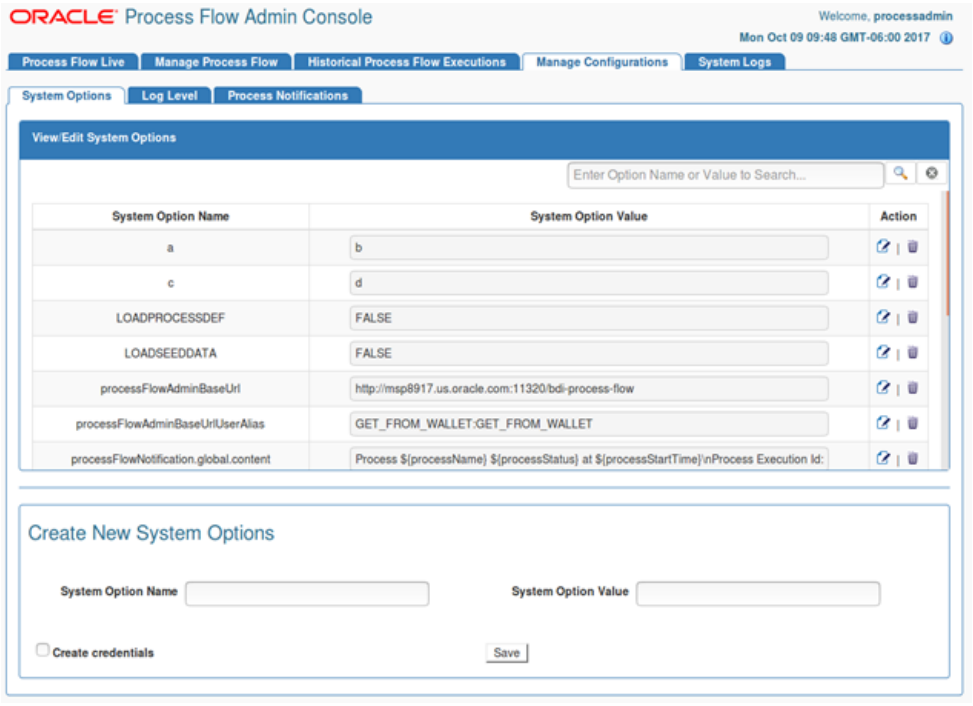


*Figure F–12 PING All with Message*

## System Options

The System Options tab allows users to view, edit, and create system options. This page displays the list of system options of the application. The user can modify the value of the existing system options, create new system options, and delete the existing system options. The user must have admin privileges for editing, creating, and deleting system options. A search option based on the system options name and value is also provided on this page.

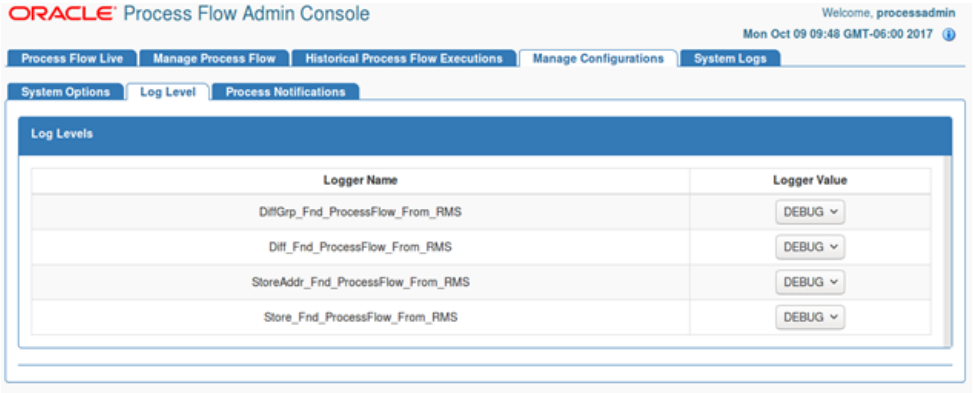
*Figure F–13 Manage Configurations Tab*




## Log Level

The Log Level tab displays log levels for all processes. Users can change log levels from this tab.

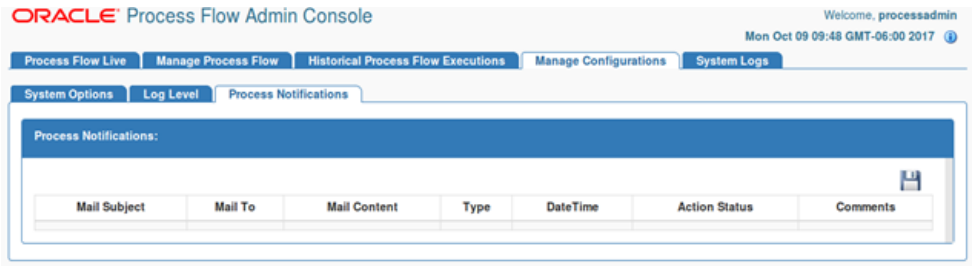
*Figure F–14 Log Level Tab*




## Process Notifications

The Process Notifications tab displays notification details. It allows user to change notification details.

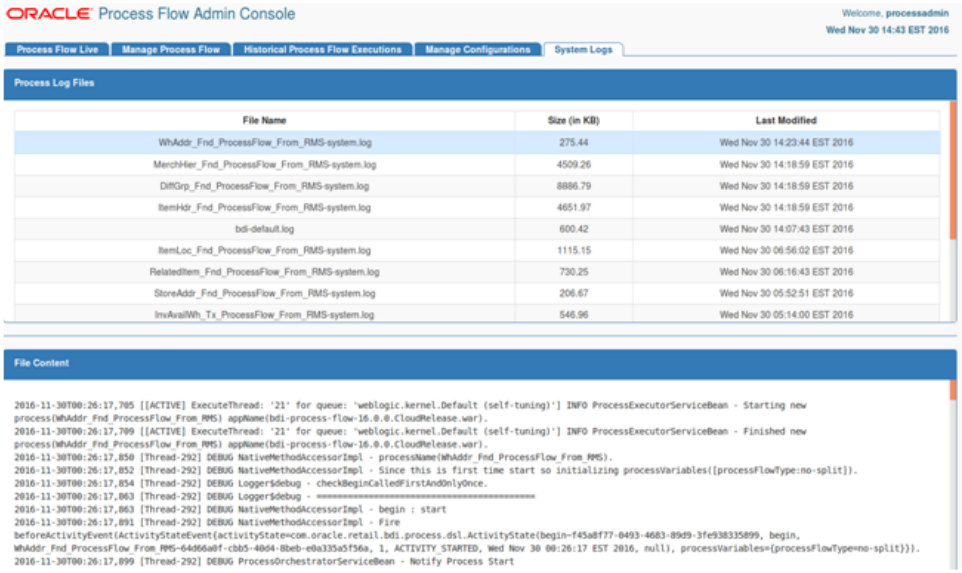
*Figure F–15 Process Notifications Tab*




# About System Logs

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.

***Figure F–16 System Logs Tab***



# G

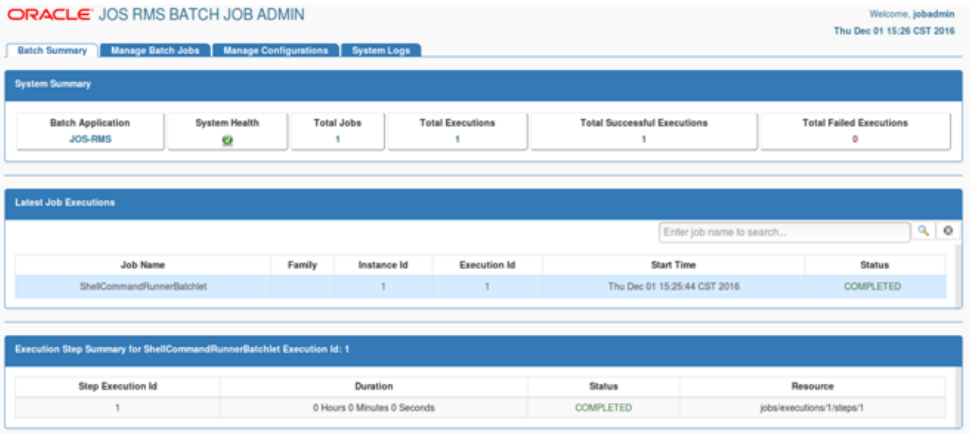
# Appendix G: Job Admin UI Screenshots

The screenshots in this appendix are part of the Job Admin User Interface.

## About the Batch Summary

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.

***Figure G–1 Batch Summary Tab***




## Manage Batch Jobs - Job Executions

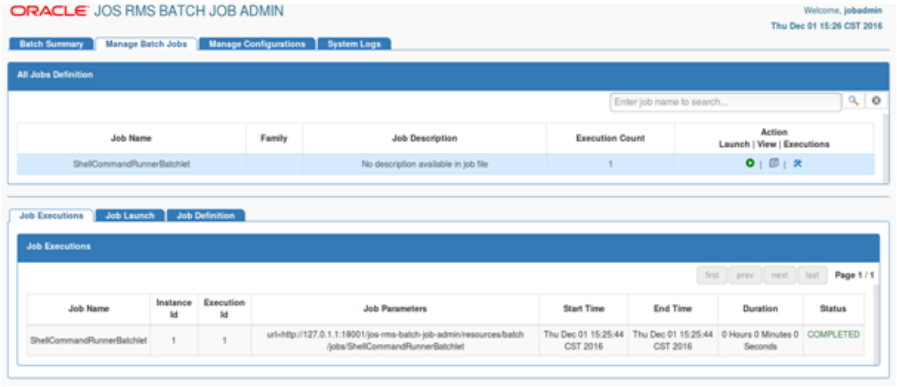
This tab shows the executions of the selected jobs. It can be used to restart the failed executions of a job. The restart button is available only for restartable executions in the status column. When the user clicks the restart button, it is redirected to the job launch tab with restart option and pre-populated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

---

**Note:** The URL is an infrastructure parameter. The user is not allowed to change its value.

---

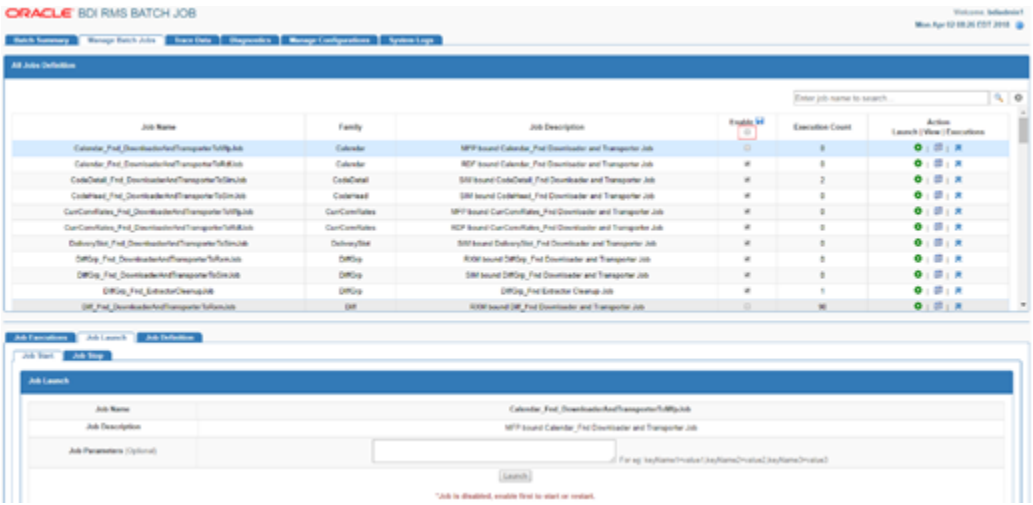
***Figure G–2 Job Executions Tab***




# Manage Batch Jobs - Job Launch

This tab can be used to launch jobs. Job Parameters is an optional input from the user to launch the jobs. Multiple job parameters can be entered in a comma separated value format.

***Figure G–3 Job Launch Tab***

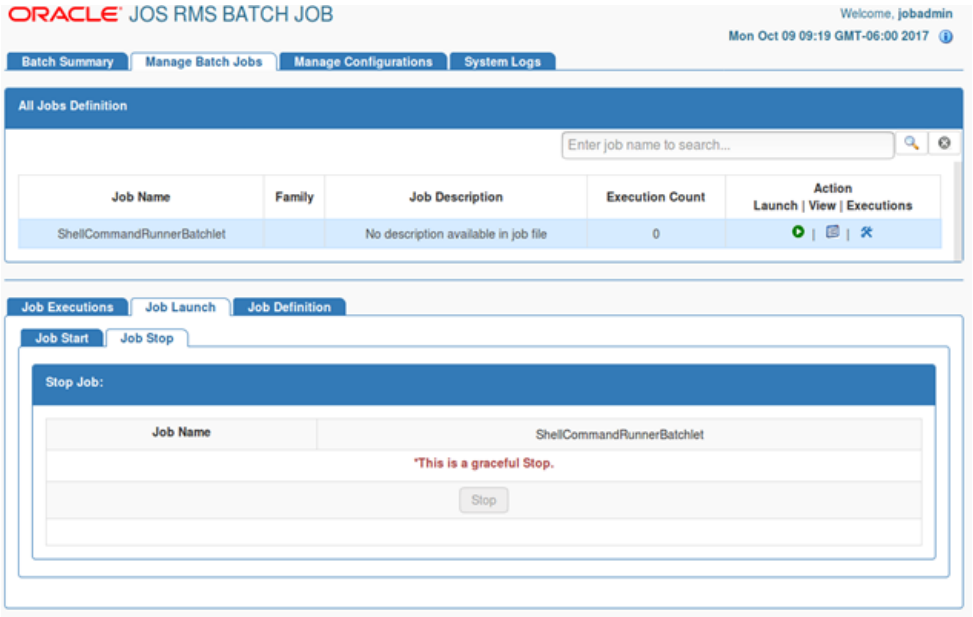



Build version and date is displayed on the info icon when the user selects the same. The icon is on the extreme right top corner of the page. By default all the jobs are enabled. Select the job row and check/uncheck the check box of each job and click on save image button in enable column. Only enabled 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.

## Job Stop

The Job Stop tab allows users to stop a job gracefully. There is no guarantee that the job will stop as it depends on whether the job has implemented stop functionality properly.

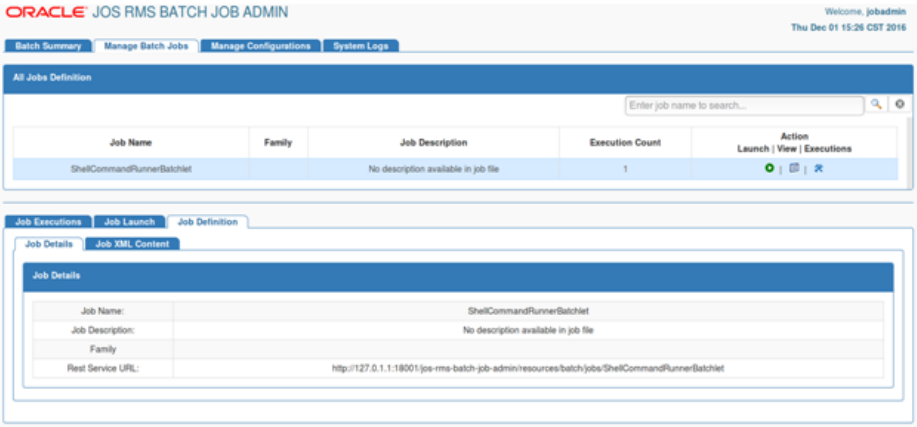
*Figure G–4 Job Stop Tab*




# Manage Batch Jobs - Job Definition - Job Details

This tab shows the details of the selected job such as Job Description, Family, and REST Service URL.

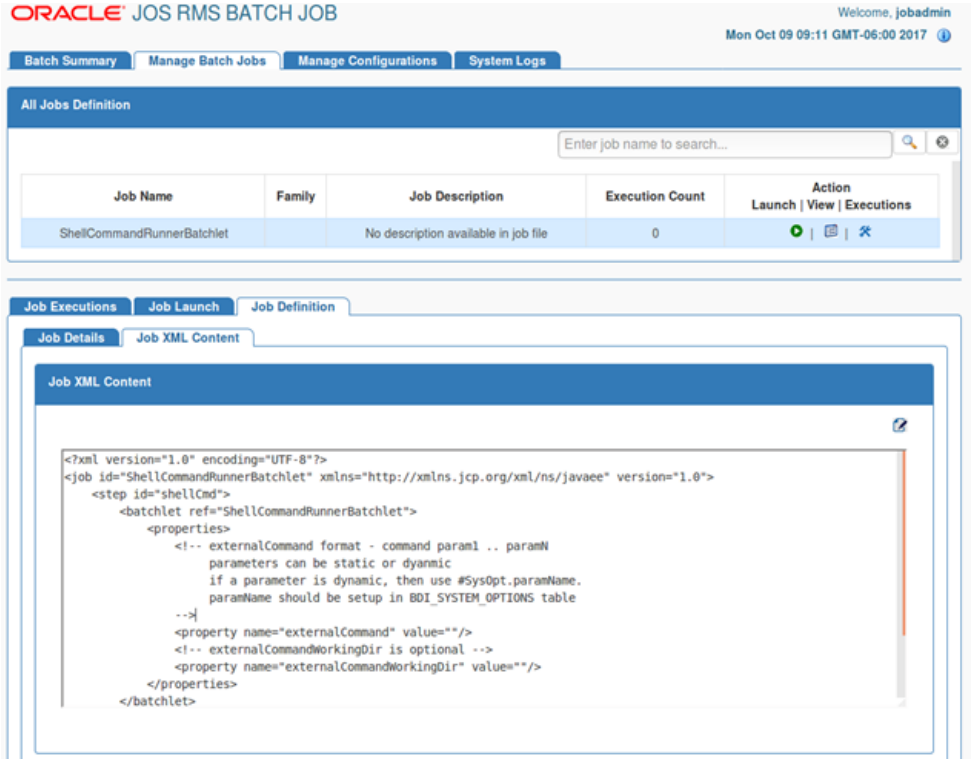
*Figure G–5 Job Details Tab*




# Manage Batch Jobs - Job Definition - Job XML Content

This tab shows the details of the selected job XML content. Users can edit job XML content from this tab.

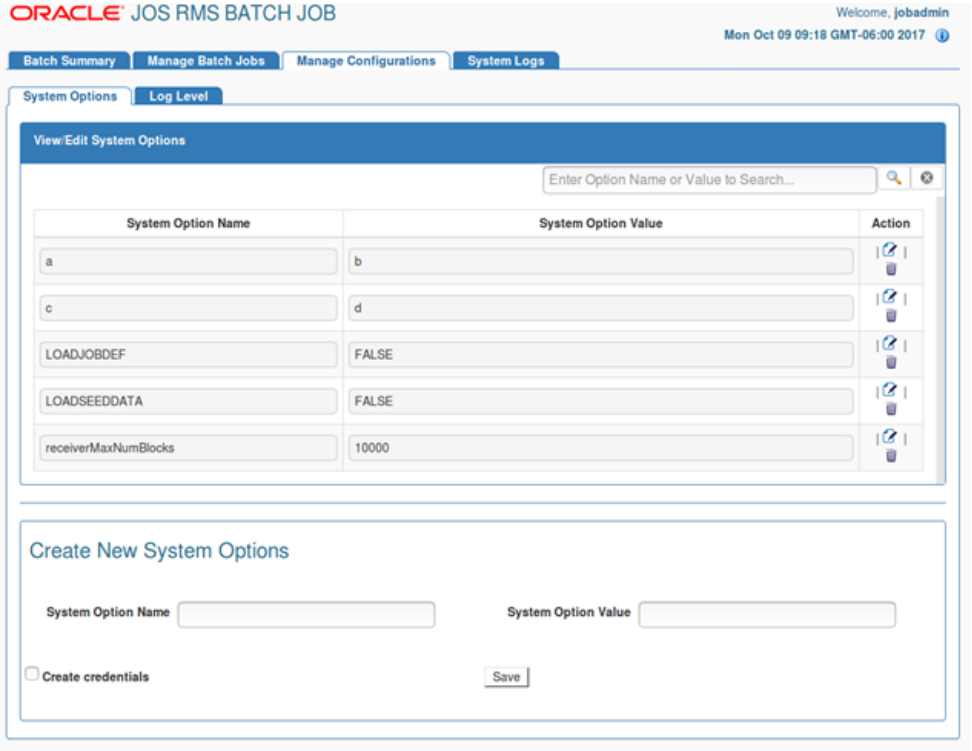
***Figure G–6 Job XML Content Tab***




# Manage Configurations

This tab shows the system options from the BDI_SYSTEM_OPTIONS table. It allows the user to add, edit, and delete new system options as well as credentials.

*Figure G–7 System Options Tab*




# System Logs

This tab shows logs at the job and system level.

*Figure G–8 System Logs Tab*