



TIBCO® Order Management Administration

Version 6.1.0 | October 2024

Contents

Contents	2
Deployment	6
Recommended Setup for a TIBCO Order Management Development Environment	6
Microservices	8
Connecting TIBCO Order Management to TIBCO® EMS Server with SSL Enabled	9
Configuring SSL for TIBCO® Order Management	10
Configuring on the Cloud	10
Configuring on-premises	15
HTTP Connection Pool Configuration	20
Configuring Authorization Server	21
Inter-service Communication	22
Configuration	23
Queue Management	23
Data Models	28
Model Loading Process	33
WebClient Configuration	41
Order Management System Configuration	41
User Interface Configuration	41
URL to Access Order Management System UI Component	42
Override Planfragment Destination	44
Managing Application Security	45
Authorization Service	51
Audit Trail	66
Enabling Internal Error Handler Support	66
Logging	67
How Logging Works	67

APIs for Changing log-level	69
Configuring Redis	71
Configuring Microsoft SQL Server	83
Configuring an External Identity Provider	85
Administration Tasks	94
Swagger API Reference	94
Docker	94
Building a Docker Image Without an Internet Connection	96
Copying Files to Docker Context	96
Building Docker Images	97
Setting Up the .env File	98
Configuring for Order Management Server Docker Containers	98
Running the Docker Containers	98
Extend Docker-Compose Files	100
Modifying a Container Time-Zone	101
Reading Container Logs	103
Troubleshooting Error from Building Docker Images	103
Order Sequencing	104
Enabling or Disabling Order Sequencing	105
Bulk Order Actions	105
Bulk Actions	106
WSDL Location	106
Error Codes	107
Invoke Bulk Order Operation	107
Tracking the Request Status	108
Logging	108
Schema	108
Sample Request	110
Sample Response	110
Performing Bulk Actions On error Plans Items	111
Multitenancy	113

Creating and Configuring a Tenant	113
Authorizing a Tenant	114
Managing Health Check Endpoint	115
Implementation of LDAP	115
User Mapping from Directory Service to Order Management service	120
Types of retries	121
API Monitoring	123
JMX MBeans	123
Prometheus	124
Elasticsearch	125
Dynatrace	127
Debugging tools for production	129
Read BLOB data from Database	129
GET REST APIs in Catalog Service and AOPD	130
Scaling of Order Management microservices	131
orderPriority	132
Order Schema Changes	133
Lower Priority Orders	133
Tuning Data Source	134
Catalog Caching	142
Integrate Inventory Information in AOPD Plan Generation	144
Integrate TIBCO OPE with Order Submission Process	145
Schema References	146
Plan Item	147
Product Model	151
Result Status	156
Message	157
Order Request	159
Order Request Header	161
Order Request Line	164

Process Component Model	170
TIBCO Documentation and Support Services	173
Legal and Third-Party Notices	175

Deployment

This section provides details about application deployment best practices and options.

Recommended Setup for a TIBCO Order Management Development Environment

The following details are the recommended setup for a TIBCO Order Management Development environment:

Component	Instances
Orchestrator	Multiple
Automated Order Plan Development	Multiple
Configurator	1
Authorization service	1
Data service	Multiple
Catalog service	Multiple
Tmf-om-adapter	1
Jeopardy service	1
PostgreSQL	1
Archival service	Multiple
Configurator UI	1

Component	Instances
Order Management System UI	1
Order Management Migration	1
Broker service	1

Hardware

8 GB of heap size is set for each instance. For orchestrator – 4 GB, for Automated Order Plan Development – 4 GB, for Data Service – 2 GB, and for Authentication–2GB memory is allocated.

Disk Space

Redis in-memory is used for TIBCO Order Management Services. For collecting 10 instances, 10-GB memory on disk is used. For 100,000 orders on 10 nodes, it takes up to 218 MB for each node.

Temporary Disk Space for UNIX Platform

The installer launcher first extracts a Java Virtual Machine (JVM) in a temporary directory and uses this JVM to open itself. The size of the extracted JVM differs from platform to platform.

On UNIX platforms, the following disk space is required in the temporary area:

256 MB of free disk space in /tmp location.

If your system does not have sufficient free disk space in the above temporary area, you can still run the installer with a different temporary area by using the following option when starting the installer:

`install_package_name.bin -is:tempdir /new_tmp` where /new_tmp has sufficient free disk space.

Microservices

Each TIBCO Order Management component, or *microservices* with the new architecture, has its embedded Tomcat container. The `roles` folder available in the `$OM_HOME` directory houses all the microservices.

The following table lists the microservices for TIBCO Order Management:

Microservice	Default Port
Configurator	9090
Orchestrator	9093
aopd	9094
Authorization service	9091
Data service	9095
Catalog service	9092
tmf-om-adapter	8181
Configurator UI	9104
Archival Service	9099
Order Management System UI	9097
Migration Service	9100
EncryptPWDUtility	9060
Jeopardy	9102
catalog-client	8082
Broker service	9105

Each microservice under the `$OM_HOME/roles/<service name>/standalone` directory has the following directory structure:

- **bin**

This directory contains shell and power shell scripts to start and stop the service. It also contains a `copyLib` script, which is a utility script that can be used to copy hibernate, JDBC, JMS and other essential dependencies.

- **config**

This directory contains the service's set of configuration files. Initially, each service has the following files:

- `application.properties`

When the service starts, it downloads its required logback files from the database.

- **lib**

This directory holds all external and internal dependency jar files.

- **logs**

This directory is created when the service starts and contains all the logs for that service.

- **services**

This directory holds the service jar file, which is launched by the start script.

Connecting TIBCO Order Management to TIBCO® EMS Server with SSL Enabled

Procedure

1. Change the following properties for each application:

- **Archival**

- `jndiConnectionFactory`
- `sslEnableVerifyHost`
- `securityProtocol`

- jmsSessionTransacted
- **Data Service**
 - tibjmsNamingSecurityProtocol
 - tibjmsNamingSslEnableVerifyHost
 - initialContextFactory
- **Jeopardy**
 - jndiConnectionFactory
 - sslEnableVerifyHost
 - securityProtocol
- **OMSUI**
 - jndiConnectionFactory
 - tibjmsNamingSecurityProtocol
 - tibjmsNamingSslEnableVerifyHost
- **Orchestrator**
 - jndiConnectionFactory
 - tibjmsNamingSecurityProtocol
 - tibjmsNamingSslEnableVerifyHost

Configuring SSL for TIBCO® Order Management

The Configuration of SSL for TIBCO® Order Management is available for both [on cloud](#) and [on-premise](#).

Configuring on the Cloud

The following section is added for testing purposes and is not recommended for the production environment. Currently, ingress is configured with SSL only for authorization service as a backend.

Procedure

1. To create a root certificate, run the following command:

```
openssl req -x509 -nodes -sha256 -days 365 -newkey rsa:2048 -subj
"/CN=test/O=TIBCO"
-keyout lab-caroot.key -out lab-caroot.crt
```

2. To create CSR for a service certificate, run the following command:

```
openssl req -out om-auth.csr -newkey rsa:2048 -nodes -keyout om-
auth.key -subj "/CN =
om-auth.test / O=auth-svc organization"
```

3. To sign the certificate with the root CA, run the following command:

```
openssl x509 -req -days 365 -CA lab-caroot.crt -CAkey lab-
caroot.key -set_serial 0 -
in om-auth.csr -out om-auth.crt
```

4. To create the Kubernetes secret, run the following command:

```
kubectl create secret tls tls-om-auth --key=om-auth.key --cert=om-
auth.crt
```

5. Add the secrets in the auth ingress YAML file:

```
```yaml
tls:
- hosts:
- om-auth.test # This should match a DNS name in the Certificate
secretName: tls-om-auth # This should match the Certificate
secretName
```

## Enabling SSL for TIBCO® Order Management

### Procedure

1. Go to the JAVA\_17\_HOME\bin directory and run the following commands:

```
keytool -genkeypair -alias om -keyalg RSA -keysize 2048 -sigalg
SHA256withRSA -validity 365 -keystore om.pkcs12 -storepass tibco123
-ext san=ip:10.xx.xx.xx,dns:10.x.x.x,ip:127.0.0.1
keytool -export -alias om -file om123.crt -keystore om.pkcs12
keytool -import -v -trustcacerts -alias om2 -file om123.crt -
keystore cacerts.pkcs12 -keypass changeit
```

When prompted, provide the password as 'changeit'.

2. Copy cacerts.pkcs12 and om.pkcs12 files from the JAVA\_HOME\bin directory to the base/1.0 directory and modify the base Dockerfile accordingly.

Example: copy om.pkcs12 and cacerts to location /home/tibuser/tibco/om/6.1

3. Copy the cacerts.pkcs12 file inside the \$OM\_HOME/roles/<Service\_name>/standalone/config/ directory of each service.
4. Run the copyLib.sh script from the roles directory.
5. Run the copy-required-files.sh script.
6. Modify the Order Management services (except authorization service) Dockerfile for entrypoint as follows:

```
ENTRYPOINT ["sh","-c",
"/home/tibuser/tibco/om/6.1/configurator/standalone/bin/
start.sh
-
Djavax.net.ssl.trustStore=/home/tibuser/tibco/om/6.1/roles/<service_name>/
standalone/config/cacerts.
pkcs12 -Djavax.net.ssl.trustStorePassword=changeit --run=FG"]
```

7. Create Docker images for all Order Management services.
8. Now, update the om\_services/values.yaml file from the \$OM\_HOME/helm directory as follows:
  - a. Add the following properties:

```
server_ssl_key_alias: om
server_ssl_key_store_password: tibco123
server_ssl_key_store: /home/tibuser/tibco/om/6.1/om.pkcs12
```

```

configuratorTrustStoreAbsolutePath:
/home/tibuser/tibco/om/6.1/cacerts.pkcs12
configuratorTrustStorePassword: changeit
configuratorTrustStoreType: pkcs12
trustStoreFileName: cacerts.pkcs12
trustStorePassword: changeit
trustStoreType: pkcs12
 aopdTrustStoreFileName: cacerts.pkcs12
 aopdTrustStorePassword: changeit
 aopdTrustStoreType: pkcs12
 migrationTrustStoreFileName: cacerts.pkcs12
 migrationTrustStorePassword: changeit
 migrationTrustStoreType: pkcs12
authServiceTrustStoreAbsolutePath=/home/tibuser/tibco/
om/6.1/cacerts.pkcs12
authServiceTrustStorePassword=changeit
authServiceTrustStoreType=pkcs12

allowedCorsOrigins: https://authorization-
svc.default.svc.cluster.local:9091,https://configurator-
svc.default.svc.cluster.local:9090,https://catalog-
svc.default.svc.cluster.local:9092,https://aopd-
svc.default.svc.cluster.local:9094,https://archival-
svc.default.svc.cluster.local:9099,https://dataservice-
svc.default.svc.cluster.local:9095,https://jeopardy-
svc.default.svc.cluster.local:9102,https://migration-
svc.default.svc.cluster.local:9100,https://orchestrator-
svc.default.svc.cluster.local:9093,https://pc-
svc.default.svc.cluster.local:9089,https://configuratorui-
svc.default.svc.cluster.local:9104,https://om-
catalog.test,https://om-archival.test,https://om-
orchestrator.test,https://om-jeopardy.test,https://om-
omsui.test,https://omsui-svc.default.svc.cluster.local:9097

com_tibco_af_omsui_httpChannelType: https
authorizationServiceTokenEndPoint: https://authorization-
svc.default.svc.cluster.local:9091
configuratorServiceUrl: https://configurator-
svc.default.svc.cluster.local:9090
pcResourceExecuteRequestURL: https://pc-
svc.default.svc.cluster.local:9089/planitem/
executionrequest

```

```

pcResourceSuspendRequestURL: https://pc-
svc.default.svc.cluster.local:9089/planitem/suspendrequest
pcResourceActivateRequestURL: https://pc-
svc.default.svc.cluster.local:9089/planitem/activaterequest
pcResourceExtErrorHandlerRequestURL: https://pc-
svc.default.svc.cluster.local:9089/planitem/
errorhandlerrequest
feasibleRequestPathRequestURL: https://pc-
svc.default.svc.cluster.local:9089/feasibility
pqfRequestPathRequestURL: https://pc-
svc.default.svc.cluster.local:9089/pqf
pcResourceMileReleaseRequestURL: https://pc-
svc.default.svc.cluster.local:9089/planitem/
milestonerelease
archivalGetOrderDetailsURL: https://archival-
svc.default.svc.cluster.local:9099/ordersByCriteria
omServerOrderUrl: https://orchestrator-
svc.default.svc.cluster.local:9093/order
omServerWithdrawOrderPath: https://orchestrator-
svc.default.svc.cluster.local:9093/order
omServerOrderDetailsPath: https://orchestrator-
svc.default.svc.cluster.local:9093/order
orchestratorBaseUrl: https://orchestrator-
svc.default.svc.cluster.local:9093
orchestratorServiceUrl: https://orchestrator-
svc.default.svc.cluster.local:9093
catalogServiceBaseUrl: https://catalog-
svc.default.svc.cluster.local:9092
catalogServiceUrl: https://catalog-
svc.default.svc.cluster.local:9092
archivalServiceUrl: https://archival-
svc.default.svc.cluster.local:9099
jeopardyServiceUrl: https://jeopardy-
svc.default.svc.cluster.local:9102
aopdBaseUrl: https://aopd-svc.default.svc.cluster.local:9094
migrationURL: https://migration-
svc.default.svc.cluster.local:9100/migration/order

```

- b. Update the scheme for each application to HTTPS.

Example: In the configurator application-

```

readinessProbe:
 failureThreshold: 3
 httpGet:
 path: /management/health/readiness
 port: 9090
 scheme: HTTPS
 periodSeconds: 300
 successThreshold: 1
 timeoutSeconds: 3
livenessProbe:
 failureThreshold: 3
 httpGet:
 path: /management/health/liveness
 port: 9090
 scheme: HTTPS
 periodSeconds: 300
 successThreshold: 1
 timeoutSeconds: 3

```

9. Specify the backend protocol as HTTPS for the Ingress in the `om_services/templates/om_ingress.yaml` file.

Example of using the Nginx Ingress:

```

annotations:
 nginx.ingress.kubernetes.io/backend-protocol: https

```

10. Create the required users from the authorization service and upload the required metadata, `app_properties`, and config files as per components from the configurator service.

The `values.yaml` file contains the required properties for starting authorization service, configurator service, and configurator UI services.

## Configuring on-premises

### Procedure

1. Go to the `JAVA17_HOME\bin` directory and run the following commands:

```
keytool -genkey -alias om -keyalg RSA -keysize 2048 -sigalg
```

```
SHA256withRSA -validity 365 -keystore om.pkcs12 -storepass tibco123
-ext san=ip:10.x.x.x,dns:10.x.x.x,ip:127.0.0.1
keytool -export -alias om -file om123.crt -keystore om.pkcs12
keytool -import -v -trustcacerts -alias om2 -file om123.crt -
keystore cacerts.pkcs12 -keypass changeit
```

When prompted, provide the password as 'changeit'.

2. Copy cacerts.pkcs12 and om.pkcs12 files from the <JAVA\_HOME>/bin directory at a location (such as /home/OM\_610/tibco/om/6.1/ssl), where your Order Management installation is present on the virtual machine.
3. For authorization service, modify the application.properties file present inside the config directory for the following properties:

```
server.ssl.key-alias=om
server.ssl.key-store-password=tibco123
server.ssl.key-store=/home/OM_610/tibco/om/6.1/ssl/om.pkcs12
```

```
#Allowed Cross Origin Resources
allowedCorsOrigins=https://10.x.x.x:9091,https://10.x.x.x:9090,
https://10.x.x.x:9092,
https://10.x.x.x:9094,https://10.x.x.x:9099,https://
10.x.x.x:9095,
https://10.x.x.x:9102,
https://10.x.x.x:9100,https://10.x.x.x:9093,https://
10.x.x.x:9089,
https://10.x.x.x:9104,
https://10.x.x.x:8090,https://10.x.x.x:8093,https://
10.x.x.x:8090
```

4. Run the following command to start the authorization service.

```
./start.sh -Djavax.net.ssl.trustStore= /home/OM_
610/tibco/om/6.1/ssl/cacerts.pkcs12
/cacerts.pkcs12 -Djavax.net.ssl.trustStorePassword=changeit
```

5. Create the required users. For more information, see [Create User](#)
6. Add the following properties for the configurator service:



```
server.ssl.key-alias=om
server.ssl.key-store-password=tibco123
server.ssl.key-store=/home/OM_610/tibco/om/6.1/ssl/om.pkcs12
```

7. Start the configurator service by running the following command:

```
./start.sh -Djavax.net.ssl.trustStore= /home/OM_610/tibco/om/6.1/ssl/cacerts.pkcs12
/cacerts.pkcs12 -Djavax.net.ssl.trustStorePassword=changeit
```

8. Modify the app\_properties file from the \$OM\_HOME/seed-data/app-properties directory for the following properties (also required minimum configurations by users):

- a. For AOPD service, under 'Orchestrator Configuration':

```
orchestratorBaseUrl = https://10.x.x.x:9093
trustStoreFileName = cacerts.pkcs12
trustStorePassword = changeit
trustStoreType = pkcs12
```

- b. For Archival service,

- Under 'Archival Engine Configurations':

```
allowedCorsOrigins = https://10.x.x.x:9097
```

- Under 'Orchestrator Configuration':

```
orchestratorBaseUrl = https://10.x.x.x:9093
trustStoreFileName = cacerts.pkcs12
trustStorePassword = changeit
trustStoreType = pkcs12
```

- c. For Catalog service,

- Under 'Catalog Engine Configuration':

```
allowedCorsOrigins=https://10.x.x.x:9097
```

- d. For Common Configuration, under 'Authorization Server Configuration

## Properties Used for Swagger UI':

```
authorizationServiceTokenEndPoint = https://10.x.x.x:9091
```

e. For the Data service, there are no changes.

f. For Jeopardy service,

- Under 'Catalog Service Configuration':

```
catalogServiceBaseUrl = https://10.x.x.x:9092
catalogServiceTrustStoreFileName = cacerts.pkcs12
catalogServiceTrustStorePassword = changeit
catalogServiceTrustStoreType = pkcs12
```

- Under 'Jeopardy General Configuration':

```
allowedCorsOrigins = https://10.x.x.x:9097
```

- Under 'Orchestrator Service Configuration':

```
orchestratorBaseUrl = https://10.x.x.x:9093
orchestratorTrustStoreFileName = cacerts.pkcs12
orchestratorTrustStorePassword = changeit
orchestratorTrustStoreType = pkcs12
```

g. For the Migration service, there are no changes.

h. For Order Management System UI Service, under 'OMS UI Engine Configuration':

```
archivalServiceBaseUrl = https://10.x.x.x:9099
catalogServiceBaseUrl = https://10.x.x.x:9092
jeopardyBaseUrl = https://10.x.x.x:9102
orchestratorBaseUrl= https://10.x.x.x:9093
```

i. For Orchestrator service,

- Under 'Orchestrator Functional Configuration':

```
allowedCorsOrigins=https://10.x.x.x:9097
```

- j. For tmfAdapter Service, under 'Orchestrator Service Configuration':

```
omServerOrderDetailsPath = https://10.x.x.x:9093/order
omServerOrderUrl = https://10.x.x.x:9093/order
omServerWithdrawOrderPath = https://10.x.x.x:9093/order
```

- k. For Broker Service, under 'Catalog Client Configuration':

```
catalogTrustStoreFileName = cacerts.pkcs12
catalogTrustStorePassword = changeit
catalogTrustStoreType = pkcs12
catalogServiceBaseUrl = https://10.x.x.x:9092
```

9. Modify the `application_metadata.json` property file. For each "applicationId", add "cacerts.pkcs12" and "om.pkcs12" (file names as per the ones created in step 1), under the "configurationFiles".
10. Upload the metadata through the Configurator Swagger.
11. Upload the "om.pkcs12" and "cacerts.pkcs12" files for each service through the Configurator API along with the other Configuration files. See 'Upload Configuration File for Application ID' section in the *TIBCO® Order Management Web Services Guide*.
12. Copy "cacerts.pkcs12" to `$OM_HOME/roles/<configurator/configurator-ui/authorization-service>/standalone/config` directory.
13. For configurator-ui and the rest of the Order Management services, update the `application.properties` file as follows:

```
server.ssl.key-alias=om
server.ssl.key-store-password=tibco123
server.ssl.key-store=/home/OM_610/tibco/om/6.1/ssl/om.pkcs12
configuratorTrustStoreAbsolutePath=cacerts.pkcs12
configuratorTrustStorePassword=changeit
configuratorTrustStoreType=pkcs12
authServiceTrustStoreAbsolutePath=cacerts.pkcs12
authServiceTrustStorePassword=changeit
authServiceTrustStoreType=pkcs12
```

14. Start all services by the following command from the <service-name>/bin directory:

```
./start.sh -Djavax.net.ssl.trustStore= /home/OM_610/tibco/om/6.1/ssl/cacerts.pkcs12 /cacerts.pkcs12 -Djavax.net.ssl.trustStorePassword=changeit
```

## HTTP Connection Pool Configuration

Configuring an HTTP connection pool is essential for optimizing the performance of applications that make frequent HTTP requests. You can configure the following key parameters:

Parameter	Description
http.client.cpool.maxTotal	Maximum number of open connections.
http.client.cpool.defaultMaxPerRoute	Maximum number of concurrent connections per route.
http.client.cpool.connectionRequestTimeout	Maximum time, in milliseconds, to wait to get a connection from the connection manager or pool. Zero is interpreted as an infinite timeout.
http.client.cpool.connectTimeout	Timeout, in milliseconds, to establish a connection with a remote host or server. Zero is interpreted as an

Parameter	Description
	infinite timeout.
http.client.cpool.socketTimeout	Maximum time gap, in milliseconds, between two consecutive data packets when transferring data from the server to the client.

## Configuring Authorization Server

You can configure the authorization server by setting the following key parameters in the ConfigValues\_OMSUI file and ConfigValues\_Common.JSON file:

Parameter	Description
authServiceApiKey	Auth service header ID (auth) used to create a token.
authServiceApiId	Auth service header Key (auth) used to create a token.
authSuperUserAppId	Used as the super user app ID to create a token.
authSuperUserAppKey	Used as the super user app key to create a token.
authServiceRetryDuration	Auth service retry duration in seconds.

Parameter	Description
authServiceRetryCount	Auth service retry count.
authServiceTrustStorePassword	Auth service SSL Trust Store password.
authServiceTrustStoreType	Auth service SSL TrustStore type.
authServiceTrustStoreAbsoluteFileName	Auth service SSL TrustStore absolute filename.
enableSecureAPI	Enable security for APIs based on this flag.

## Inter-service Communication

By using the inter-service communication, the services communicate with each other within a system using REST services. You can configure the `apiKey` in the `ConfigValues_Common.JSON` file. The `apiKey` is responsible for generating a token to facilitate interaction between applications.

# Configuration

---

This section covers all the configuration details for TIBCO Order Management.

## Queue Management

REST services are used by TIBCO Order Management for publishing models. If you select TIBCO Enterprise Message Service, then it communicates with the external systems through the JMS messaging capability provided by TIBCO Enterprise Message Service. It has two inbound queues to receive the messages from external systems and also for inter-component communication in some cases. The number of listeners on these queues can be configured using the changing `concurrent.ems.consumer` flag. By default, the listener count on each queue is set to a minimal value. The queue configurations are available under different categories distributed component wise.

You can update the `ConfigValues_OrchService.json` file to set the following properties for the queue management:

Parameter	Description
<code>com.tibco.fom.orch.planItem.execute.response.queue</code>	Plan item execution response queue
<code>com.tibco.fom.orch.planItem.execute.request.queue</code>	Plan item execution request queue
<code>com.tibco.fom.orch.planItem.execute.response.dead.queue</code>	Plan item execution response dead queue

Parameter	Description
com.tibco.fom.orch.planitem.execute.response.receiver.count	Plan item execution response receiver count
com.tibco.fom.orch.planItem.suspend.request.queue	Plan item suspend request queue
com.tibco.fom.orch.planItem.suspend.response.queue	Plan item suspend response queue
com.tibco.fom.orch.planitem.suspend.response.receiver.count	Plan item suspend response receiver count
com.tibco.fom.orch.planItem.suspend.response.dead.queue	Plan item suspend response dead queue
com.tibco.fom.orch.planItem.activate.request.queue	Plan item activate request queue
com.tibco.fom.orch.planItem.milestone.releaseRequest.queue	Milestone release request from Orchestrator to process components queue
com.tibco.fom.orch.planItem.milestone.notifyRequest.queue	Milestone notify request



Parameter	Description
	queue
com.tibco.fom.orch.planItem.milestone.notifyRequest.dead.queue	Milestone notify request dead queue
com.tibco.fom.orch.planItem.milestone.notifyRequest.receiver.count	Milestone notify request receiver count
com.tibco.af.oms.ordersService.queue	Queue for receiving SOAP Over JMS Order Service requests
com.tibco.af.oms.webservice.soap.jms.concurrentConsumers	Number of concurrent consumers for SOAP Over JMS Order Service requests (default 1)
com.tibco.fom.orch.prequalificationfailed.request.queue	External pre-qualification failed request queue
com.tibco.fom.orch.prequalificationfailed.reply.queue	External pre-qualification failed reply queue
com.tibco.fom.orch.prequalificationfailed.reply.queue.receiver.count	External pre-qualification failed reply

Parameter	Description
	queue receiver count
com.tibco.fom.orch.prequalificationfailed.reply.dead.queue	External pre-qualification failed reply dead queue
com.tibco.fom.orch.feasibility.request.queue	External feasibility request queue
com.tibco.fom.orch.feasibility.reply.queue	External feasibility reply queue
com.tibco.fom.orch.feasibility.reply.queue.receiver.count	External feasibility reply queue receiver count
com.tibco.fom.orch.feasibility.reply.dead.queue	External feasibility reply dead queue
com.tibco.fom.orch.planItem.errhandler.response.queue	PlanItem error handler response queue
com.tibco.fom.orch.planItem.errhandler.response.count	PlanItem error handler response receiver count
com.tibco.fom.orch.planItem.errhandler.response.dead.queue	PlanItem error handler response dead

Parameter	Description
	queue
com.tibco.fom.orch.order.sequencing.queue	Order sequencing queue
com.tibco.fom.orch.order.sequencing.dead.queue	Order sequencing dead queue
com.tibco.fom.orch.order.sequencing.retry.count	Order sequencing retry count
com.tibco.fom.orch.order.sequencing.redelivery.delay	Order sequencing notification redelivery delay, in milliseconds, between each retry
com.tibco.fom.orch.order.sequencing.receiver.count	Order sequencing receiver count
com.tibco.fom.orch.planItem.errhandler.request.queue	PlanItem error handler request queue
southboundReplyMessageRetryCount	Southbound reply message retry count
southboundReplyMessageRetryDuration	Retry interval, in milliseconds,

Parameter	Description
	between each retry
com.tibco.fom.orch.plan.failed.request.queue	Plan generation failed request queue
com.tibco.fom.orch.plan.failed.response.queue	Plan generation failed response queue
com.tibco.fom.orch.plan.failed.response.dead.queue	Plan generation failed response dead queue
com.tibco.fom.orch.plan.failed.response.receiver.count	Plan generation failed receiver count

## Data Models

TIBCO Order Management requires a variety of data models (catalogs) for its different functionalities.

TIBCO Order Management uses the following data models:

Data Models	Description
Product Model	It is used by the Automated Order Plan Development component for generating the execution plan for the newly submitted orders.
Action Model	It is optionally used by the Automated Order Plan Development component when generating the execution plans specifically for the ProductDependsOn feature.

<b>Data Models</b>	<b>Description</b>
Plan Fragment Model	It is used by the Orchestrator component for running the plan for a particular order.

The following table summarizes the models required by the components in TIBCO Order Management:

<b>Components</b>	<b>Product model</b>	<b>Plan Fragment model</b>	<b>Action model</b>
Automated Order Plan Development	Required	Not Required	Optional
Orchestrator	Not Required	Required	Not Required

You can configure the following catalog properties:

<b>Property</b>	<b>Description</b>
isOfferSearchIndexEnabled	Enables outbound notifications to EMS upon successful product catalog loading.
modelPurgeWorkerThreadCount	Specifies the number of parallel processing threads for purging the catalog.
productCatalogLoadingQueue	Queue name for bulk product catalog loading.
bulkProductConcurrentEmsConsumers	Number of concurrent EMS consumers for bulk product loading.

Property	Description
planfragmentCatalogLoadingQueue	Queue name for bulk plan-fragment catalog loading.
bulkPlanFragmentConcurrentEmsConsumers	Number of concurrent EMS consumers for bulk plan-fragment loading.
actionCatalogLoadingQueue	Queue name for bulk action catalog loading.
bulkActionConcurrentEmsConsumers	Number of concurrent EMS consumers for bulk action loading.
priceCatalogLoadingQueue	Queue name for bulk price catalog loading.
bulkPriceConcurrentEmsConsumers	Number of concurrent EMS consumers for bulk price loading.
discountCatalogLoadingQueue	Queue name for discount catalog loading.
bulkDiscountConcurrentEmsConsumers	Number of concurrent EMS consumers for bulk discount loading.
ruleCatalogLoadingQueue	Queue name for rule catalog loading.
Queue ruleConcurrentEmsConsumers	Number of concurrent EMS consumers for rule loading.
categoryCatalogLoadingQueue	Queue name for category catalog loading.

Property	Description
categoryConcurrentEmsConsumers	Number of concurrent EMS consumers for category loading.
offerSearchProductIndexQueue	Queue name for offer search product index.
singleProductCatalogLoadingQueue	Queue name for single product catalog loading.
singleProductConcurrentEmsConsumers	Number of concurrent EMS consumers for single product loading.
singlePlanfragmentCatalogLoadingQueue	Queue name for single plan fragment catalog loading.
singlePlanFragmentConcurrentEmsConsumers	Number of concurrent EMS consumers for single plan fragment loading.
singleActionCatalogLoadingQueue	Queue name for single action catalog loading.
singleActionConcurrentEmsConsumers	Number of concurrent EMS consumers for single action loading.
singlePriceCatalogLoadingQueue	Queue name for single price catalog loading.
singlePriceConcurrentEmsConsumers	Number of concurrent EMS consumers for single price loading.
singleDiscountCatalogLoadingQueue	Queue name for single discount catalog loading.

Property	Description
singleDiscountConcurrentEmsConsumers	Number of concurrent EMS consumers for single discount loading.
productCatalogLoadingDeadQueue	Queue name for product catalog loading dead queue.
singleProductCatalogLoadingDeadQueue	Queue name for single product catalog loading dead queue.
planfragmentCatalogLoadingDeadQueue	Queue name for plan fragment catalog loading dead queue.
singlePlanfragmentCatalogLoadingDeadQueue	Queue name for single plan fragment catalog loading dead queue.
actionCatalogLoadingDeadQueue	Queue name for action catalog loading dead queue.
singleActionCatalogLoadingDeadQueue	Queue name for single action catalog loading dead queue.
priceCatalogLoadingDeadQueue	Queue name for price catalog loading dead queue.
discountCatalogLoadingDeadQueue	Queue name for discount catalog loading dead queue.
categoryCatalogLoadingDeadQueue	Queue name for category



Property	Description
	catalog loading dead queue.
ruleCatalogLoadingDeadQueue	Queue name for rule catalog loading dead queue.
singlePriceCatalogLoadingDeadQueue	Queue name for single price catalog loading dead queue.
singleDiscountCatalogLoadingDeadQueue	Queue name for single discount catalog loading dead queue.
singlePlanFragmentLoadingQueue	Queue name for single plan fragment loading queue.
singleActionModelLoadingQueue	Queue name for single action model loading queue.

## Model Loading Process

The models mentioned in [Data Models](#) must be loaded up TIBCO Order Management so that they can be used by different components. These data models are modeled as catalogs using repositories and relationships readily available in TIBCO Product and Service Catalog. After modeling the catalogs in TIBCO Product and Service Catalog, they can be made available to TIBCO Order Management and these models loaded through a catalog service.

Use the following ways to load the models into TIBCO Order Management:

- [Catalog Web Service Model Loading](#)
- [Online Model Loading](#)
- [Offline Model Loading](#)

## Online Model Loading

Online model loading requires the invoking of the catalog publish workflow in TIBCO Product and Service Catalog using the exposed SOAP service.

You can invoke the catalog publish workflow in TIBCO Product and Service Catalog directly by using the sample SOAP web service requests. The request can be sent using any standard SOAP client tools such as SOAPUI. Specify the correct enterprise name, user name, and password in the request. Also, specify the correct MASTERCATALOGNAME key and a PRODUCTID to publish the specific catalog.

Invoke the request against the running instance of TIBCO Product and Service Catalog on the URL, which typically looks like

`http://<HOST>:<PORT>/eml/services/router/MasterCatalogRecordAction` where HOST and PORT are the machine name and port number where TIBCO Product and Service Catalog is deployed and running.

Refer to the TIBCO Product and Service Catalog documentation for more details.

TIBCO Product and Service Catalog publishes the models on respective topics as mentioned in the following table:

Model (Catalog)	TIBCO Product and Service Catalog JMS Topic
Product	tibco.ac.productmodel.topic
Action	tibco.ac.actionmodel.topic
Plan Fragment	tibco.ac.planfragmentmodel.topic

To make these models available to TIBCO Order Management, the following JMS bridges must be created between the TIBCO Product and Service Catalog topics and the corresponding TIBCO Order Management queues as mentioned in the following table:

TIBCO Product and Service Catalog Source Topic	TIBCO Order Management Target Queue
tibco.ac.productmodel.topic	tibco.aff.catalog.product.request

<b>TIBCO Product and Service Catalog Source Topic</b>	<b>TIBCO Order Management Target Queue</b>
tibco.ac.actionmodel.topic	tibco.aff.catalog.action.request
tibco.ac.planfragmentmodel.topic	tibco.aff.catalog.planfragment.request

## Catalog Web Service Model Loading

Catalog Web Service is used to load the models into TIBCO Order Management.

The URL for catalog service is `PROTOCOL://<HOST>:<PORT>/swagger-ui.html#`

Following are the types of Catalog web services:

- Post request for `/v1/planfragmentmodel`  
Operation to load single planfragment model
- Post request `/v1/planfragmentmodel/bulk`  
Operation to load multiple planfragment models
- Post request for `/v1/productmodel`  
Operation to load single product model
- Post request for `/v1/productmodel/bulk`  
Operation to load multiple product models
- Post request for `/v1/actionmodel`  
Operation to load single action model
- Post request for `/v1/actionmodel/bulk`  
Operation to load multiple action models
- Delete request for `/v1/actionmodel/bulk`  
Operation to purge action model
- Delete request for `/v1/planfragmentmodel/bulk`  
Operation to purge planfragment model
- Delete request for `/v1/productmodel/bulk`

Operation to purge product model

- Get request for `/v1/planfragmentmodel/bulk`

Operation to get Bulk Plan Fragment Model

- Get request for `/v1/planfragmentmodel/all`

Operation to get All Plan Fragment Models

- Get request for `/v1/productmodel/bulk`

Operation to get Bulk product Model

- Get request for `/v1/productmodel/all`

Operation to get All product Models

- Get request for `/v1/actionmodel/bulk`

Operation to get Bulk action Model

- Get request for `/v1/actionmodel/all`

Operation to get All action Models

## Offline Model Loading

A client can upload offline catalogs to the Catalog Service. Based on `catalogPublishMode`, it uploads the model via either EMS or REST API. It can publish catalogs in parallel. As per your environment, provide an appropriate value of `workerThreadCount`. If a file has multiple catalogs, for example `<ProductModels>` has multiple `<ProductModel>`, each catalog would be published to the Catalog Service separately.

## Setting up Catalog Client

For offline model loading, the user can use the `catalog-client`, do the following steps:

In the `$OM_HOME/samples/catalog-client/config/application.properties` file, set the properties mentioned in the following table.



**Note:** Make sure that the catalog service is up.

## General Configuration Properties

Property	Description	Default value	Notes
server.port	The default port on which this service is running	8082	
workerThreadCount	Number of worker threads available to publish catalogs in parallel	2	
default.tenant.id	Default Tenant ID	TIBCO	
enableSecureAPI	Whether enableSecureAPI is true for Catalog Service	True	
catalogPublishMode	Channel on which offline models would be published	JMS	Allowed values are JMS and REST
catalogServiceEndpoint	Base URL of Catalog Service	http://<host_name>:9092	Used to make REST call when catalogPublishMode is selected as REST
catalogServiceTrustStoreFileName	File name of the catalog service trust store		
catalogServiceTrustStorePassword	Password of the catalog service trust store		

Property	Description	Default value	Notes
catalogServiceTrustStoreType	Type of the catalog service trust store		

## Authorization Properties

Property	Description	Default value	Notes
authorization.service.username	Username to generate OAuth Token	admin	
authorization.service.password	User password to generate OAuth Token	ENC (T9aNk07NM\$U=)	Encrypted value of admin. Use EncryptorDecryptorUtil to encrypt the key
authorizationServiceTokenEndPoint	Authorization Server OAuth URL	http://<host_name>:9091	
authServiceTrustStoreFileName	File name of the authorization service trust store		
authServiceTrustStorePassword	Password of the authorization service trust store		
authServiceTrustStoreType	Type of the		

Property	Description	Default value	Notes
	authorization service trust store		
authorization.access.token.validity		43200	

## JMS Configurations

Property	Description	Default Value	Notes
emsServerURL	EMS Server URL	tcp://localhost:7222	
emsServerUsername	EMS Server username	Admin	
emsServerPassword	EMS Server Password	ENC(T9aNk07NM\$U=)	Encrypted value of admin. Use EncryptorDecrypt orUtil to encrypt the key
timeoutMillis	EMS message acknowled ge timeout	10000	
securityProtocol	Security protocol to use in Tibjms JNDI		

Property	Description	Default Value	Notes
	lookups		
sslEnableVerifyHost	Enable TrustStore verification using SSL		
productModelPublishQueue	Product Model publishes Queue	tibco.aff.catalog.product.request.single	
planFragmentPublishQueue	Plan fragment Model publishes Queue	tibco.aff.catalog.planfragment.request.single	
actionModelPublishQueue	Action Model publishes Queue	tibco.aff.catalog.action.request.single	
priceModelPublishQueue	Price Model Publish Queue	tibco.aff.catalog.price.request.single	
discountModelPublishQueue	Discount Model Publish Queue	tibco.aff.catalog.discount.request.single	
categoryModelsPublishQueue	Category Models	tibco.aff.catalog.category.request	



Property	Description	Default Value	Notes
	Publish Queue		
ruleModelsPublishQueue	Rule Models Publish Queue	tibco.aff.catalog.operulemodel.request	

## WebClient Configuration

When you start a container such as Kubernetes or Helm chart, all the services start together. As the configurator service too starts along with the other services, the catalog service fails as it is dependent on the configurator.

As a workaround, a retry mechanism is added.

In the `$OM_HOME/roles//catalog-service/standalone/config/application.properties` file, `configuratorServiceRetryCount` and `configuratorServiceRetryDuration` flags are added. Before failing, the catalog service tries to reload the number of times that you specified in this flag. Also, you can set the retry duration in seconds.

## Order Management System Configuration

### User Interface Configuration

Order Management System provides a web user interface to browse and perform actions on the orders and execution plans. Order Management System UI is deployed as a separate application, and it requires parameters to connect to the Archival, Orchestrator, Jeopardy, and Catalog service.

**Note:** Order Management System UI also provides configurable parameters to control the access to the application.

## User Interface Configuration

The screenshot shows the TIBCO Configurator interface. On the left is a sidebar with a list of available applications: AOPD, Archival Service, Catalog Service, Common Configuration, Data Service, Jeopardy Service, Migration Service, Order Management System UI (highlighted), Order Management System, and TMFAdapter Service. At the bottom of the sidebar is a 'SEEDING' button. The main area is titled 'Configurator' and has two tabs: 'App properties' (selected) and 'Configuration files'. A search bar at the top right says 'Search in Order Management System UI'. In the center, there's a 'Choose category' dropdown menu with a refresh icon. Below it, a list of categories is shown: Application Security, Messaging Configuration, OMS UI Engine Configuration (highlighted), Orchestrator Configuration, and Persistence. To the right of this menu is a table titled 'Order Management System UI > OMS UI Engine Configuration' with an 'EDIT' button. The table has three columns: Property name, Value, and Description.

Property name	Value	Description
archivalServiceUri	http://localhost:9099	Archival service URL
catalogServiceUri	http://localhost:9092	Catalog service URL
com.tibco.af.concurrencyControl.maxSessionPerUser	1	
com.tibco.af.oms.default.tenantId	TIBCO	Default tenant information
com.tibco.af.oms.security.bcryptPasswordEncoder	org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder	bcryptPasswordEncoder
com.tibco.af.omsui.enableRecordCountFetch	true	Enable the Record count fetch for pagination. This will make the data fetch slower and enable Last Page option in pagination.
com.tibco.af.omsui.http.port	9097	
com.tibco.af.omsui.httpChannelType	http	
com.tibco.af.omsui.https.port	8443	

## URL to Access Order Management System UI Component

Users can directly access the Order Management System UI by providing `http://localhost:9097/` in the browser tab.

Initially, the user navigated to the login page and once the authentication is completed. The user is redirected to the dashboard's Home component where upcoming orders can be viewed.

Observe the following two scenarios:

### Find specific order using the 'Order Details' component

The URL is redirected to a specific component, based on the target orderID in the search parameters of `http://localhost:9097/#/dashboard/order-details?id=orderID`

---

**Accessing worktray orders directly to take action**

Users can view orders and take an action that were previously added to the worktray by navigating to

<http://localhost:9097/#/dashboard/orders?id=bulk>

To add an order to the worktray the user needs to navigate to Dashboard Home and select the orders from the 'Find Orders' table.

---

## Side Navigation for Order Management System UI

The side drawer navigation of Order Management System UI are as follows:

- Dashboard
- Bulk Action Job
- Saved Searches
- Jeopardy Rules

## Filtration of orders in Order Management System UI

User can filter orders by giving certain search criteria in **Dashboard > Search Orders >** 

The search categories of the filter widget are as follows:

---

Order	<ul style="list-style-type: none"> <li>• Search By IDs (Order ID, Order Ref, Customer ID, Subscriber ID)</li> <li>• Search by Dates</li> <li>• Filter By Order Status</li> <li>• Search by Custom Headers</li> </ul>
Plan	<ul style="list-style-type: none"> <li>• Search By IDs (Plan ID, Order ID, Order Ref, Process Component ID, Process Component Name)</li> <li>• Search by Dates</li> <li>• Filter By Status</li> </ul>

---

## Override Planfragment Destination

You can use the following property under the category 'Orchestrator Functional Configuration' to determine messages sent to the process component is sent to a new JMS destination or not:

```
{
 "propName": "tenantSpecificDestination",
 "propDescription": "Flag to enable or disable using the tenant
specific destination for process component",
 "propValue": "false",
 "valueType": "string",
 "isTenantProperty": "true"
}
```

If this property is set to `true`, the messages are sent to a JMS destination prefixed with the tenant ID to the existing destination as follows:

`<TENANTID>.tibco.aff.orchestrator.planItem.execute.request`

Along with the existing property `overridePlanfragmentDestination`, the new flag `tenantSpecificDestination` works in the following manner:

- If `overridePlanfragmentDestination` is set to `true` and `tenantSpecificDestination` is set to `false`, the messages are sent to the configured destination for the respective process component.
- If `overridePlanfragmentDestination` is set to `false` and if `tenantSpecificDestination` is `true`, then
  - if the owner is defined for this process component, the JMS destination is `tibco.aff.orchestrator.planItem.<planFragment-owner>.execute.request`.
  - if the owner is not defined for this process component, the JMS destination is `<TENANTID>.tibco.aff.orchestrator.planItem.execute.request`.
- If `overridePlanfragmentDestination` is set to `false` and if `tenantSpecificDestination` is `false`, then
  - if the owner is defined for this process component, the JMS destination is `tibco.aff.orchestrator.planItem.<planFragment-owner>.execute.request`.
  - if the owner is not defined for this process component, the JMS destination is

```
tibco.aff.orchestrator.planItem.execute.request.
```

**i Note:** By default, both properties are set to false.

## Managing Application Security

Order Management Server provides the below security option.

- [Authorization Service](#)

## Managing Users and Roles

Order Management Server supports role-based authorization. The user must belong to either ROLE\_USER or ROLE\_ADMIN.

The following table shows business functions and a list of roles that are authorized to perform the business functions.

**i Note:** You can change the roles to perform business functions. This can be achieved by changing 'Application Security Configurations' category for all services application from the configurator UI.

Order Management Server Interface	Function	Roles
Orchestrator	Submit Order	ROLE_ADMIN

Order Management Server Interface	Function	Roles
	Get Order Detail	ROLE_USER, ROLE_ADMIN
	Order withdraw	ROLE_ADMIN
	PlanItem Execute Reply	ROLE_ADMIN
	AmendOrder	ROLE_ADMIN
	FeasibilityReply	ROLE_ADMIN
	ActivateOrderRequest	ROLE_ADMIN
	CancelOrder	ROLE_ADMIN
	GetOrderExecutionPlan	ROLE_USER, ROLE_ADMIN
	SuspendOrderRequest	ROLE_ADMIN
	BulkAction	ROLE_ADMIN
	PlanItemBulkErrorHandler	ROLE_ADMIN
	PlanItemErrorHandler	ROLE_ADMIN
	MilestoneNotifyRequest	ROLE_ADMIN
	PlanItemSuspendResponse	ROLE_ADMIN
	PreQualificationFailedReply	ROLE_ADMIN
	submitOrderExecutionPlan	ROLE_ADMIN
	planItemSuspendReply	ROLE_ADMIN

Order Management Server Interface	Function	Roles
	Purge Order	ROLE_ADMIN
	orderScXml	ROLE_USER, ROLE_ADMIN
	getplanfragment	ROLE_USER, ROLE_ADMIN
	GetOrderMessages	ROLE_USER,ROLE_ADMIN
	GetOrderStatus	ROLE_USER,ROLE_ADMIN
	submitPlanErrorNotification	ROLE_ADMIN



Order Management Server Interface	Function	Roles
Catalog Service	submitPlanFragmentModel	ROLE_ADMIN
	purgePlanFragmentModel	ROLE_ADMIN
	submitProductModel	ROLE_ADMIN
	purgeProductModel	ROLE_ADMIN
	submitActionModel	ROLE_ADMIN
	purgeActionModel	ROLE_ADMIN
	getProductModelRoles	ROLE_ADMIN
	getPlanFragmentModelRoles	ROLE_ADMIN
	getActionModelRoles	ROLE_ADMIN
	getAllActionModelRoles	ROLE_ADMIN
	getAllProductModelRoles	ROLE_ADMIN
	getAllPlanFragmentModelRoles	ROLE_ADMIN
Data Service	setPlanRequest	ROLE_ADMIN
	setPlanItemRequest	ROLE_ADMIN
	getPlanItemsRequest	ROLE_USER, ROLE_ADMIN
	getPlanRequest	ROLE_USER, ROLE_ADMIN
Archival Service	getOrderSummary	ROLE_USER, ROLE_ADMIN

Order Management Server Interface	Function	Roles
	getOrdersByCriteria	ROLE_USER, ROLE_ADMIN
	getPlansByCriteria	ROLE_USER,ROLE_ADMIN
	getAuditTrailsData	ROLE_USER,ROLE_ADMIN
	purgeOrder	ROLE_ADMIN
	SubmitAuditTrail	ROLE_ADMIN
	GetSavedSearches	ROLE_USER,ROLE_ADMIN
	UpdateSavedSearches	ROLE_ADMIN
	SavedSearches	ROLE_ADMIN
	DeleteSavedSearches	ROLE_ADMIN

## Changing the Default Roles of a User

### Procedure

1. Open the `$OM_HOME/roles/authorization-service/standalone/config/application.properties` file in a text editor and update `allowedUserRoles` property with the required role values.



**Note:** In the case of OIDC, add UserRoles specific to your organization in this property.

2. Register the required tenant. See [Registering a Tenant](#). For more information on the Tenant Registration APIs, see the "Authorization Service API Samples" topic in the

*TIBCO® Order Management Web Services Guide.*

3. Create a user with the roles that are set in the previous step. See [Create User](#).
4. Open the \$OM\_HOME/roles/configurator/standalone/config/application.properties file in a text editor and update the configuratorAccessRoles property with the required roles.
5. Update the operation role values under 'Application Security Configurations' category for all the services from the configurator UI.

## Authorization Service

To ensure secure access to TIBCO Order Management system REST APIs and support multitenancy, token-based authentication is implemented in TIBCO Order Management.

The authentication service in TIBCO Order Management uses the JSON Web Token (JWT) to validate user credentials (user name, password, and tenantID).

The following functions are covered under the Authorization Service:

- [Registering a Tenant](#)
- [Update tenant information](#)
- [Get tenant information](#)
- [Delete tenant](#)
- [Create User](#)
- [Update User](#)
- [Get User](#)
- [Delete User](#)

After a user is created, authenticate it by following the procedures in [Generating an authorization token](#) topic.

## Registering a Tenant

You can register a tenant by setting the identity provider type to Oracle, PostgreSQL, LDAP, or EXTERNAL. Separate databases are created for each registered tenant's user.

Tenant registration API is shown as follows:

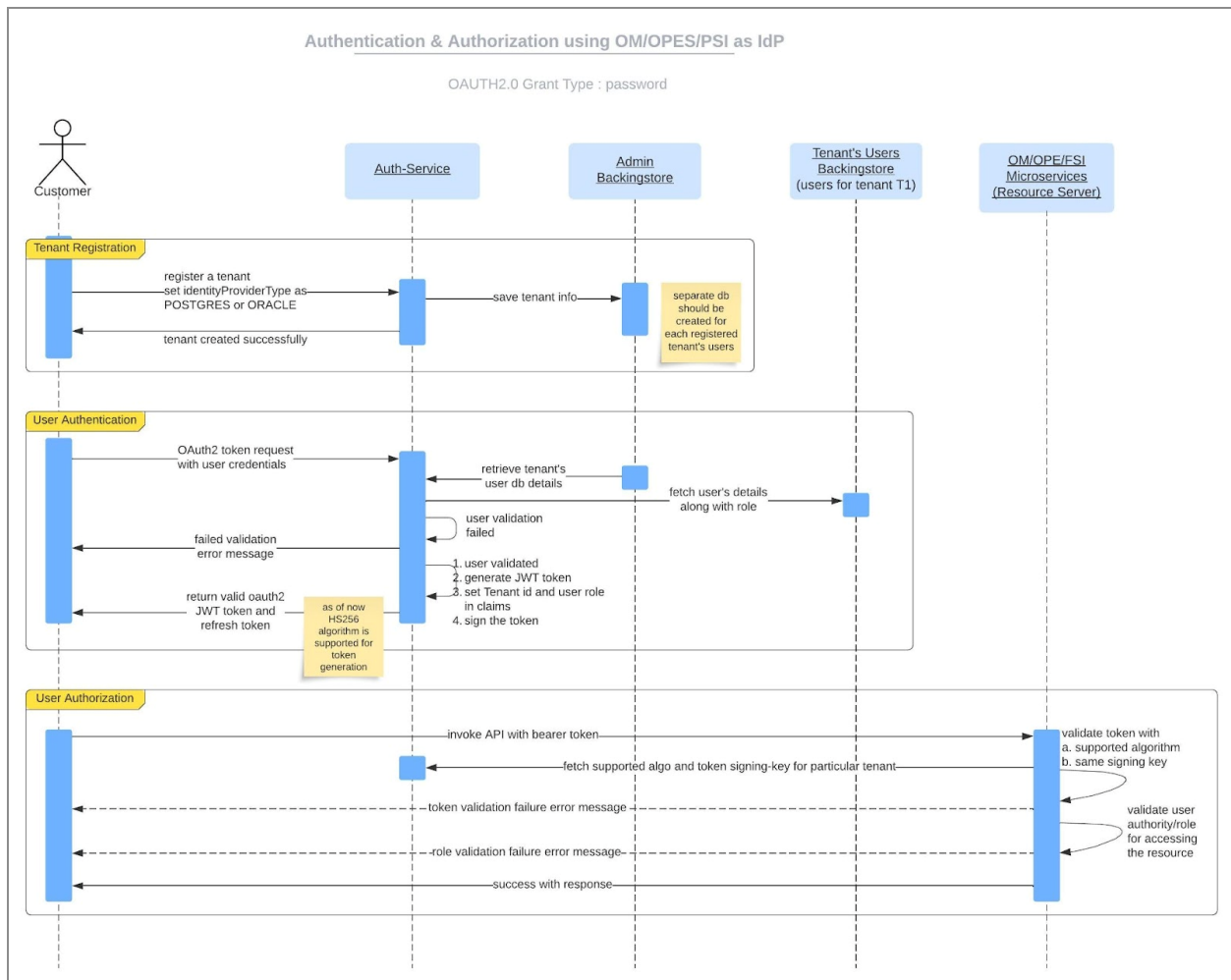
This operation registers tenant information. This API can handle only single tenant registration at a time.

**Method:** HTTP POST

**Endpoint:** `http://<host_address>:<port_address>/v1/tenant`

Parameter	Cardinality	Description
X-API-AppId	Mandatory	The application ID is used for getting the user details.
X-API-Key	Mandatory	This key is used for getting the user details.

If you set the identity provider as Oracle or PostgreSQL, then you have to create separate databases for each tenant.



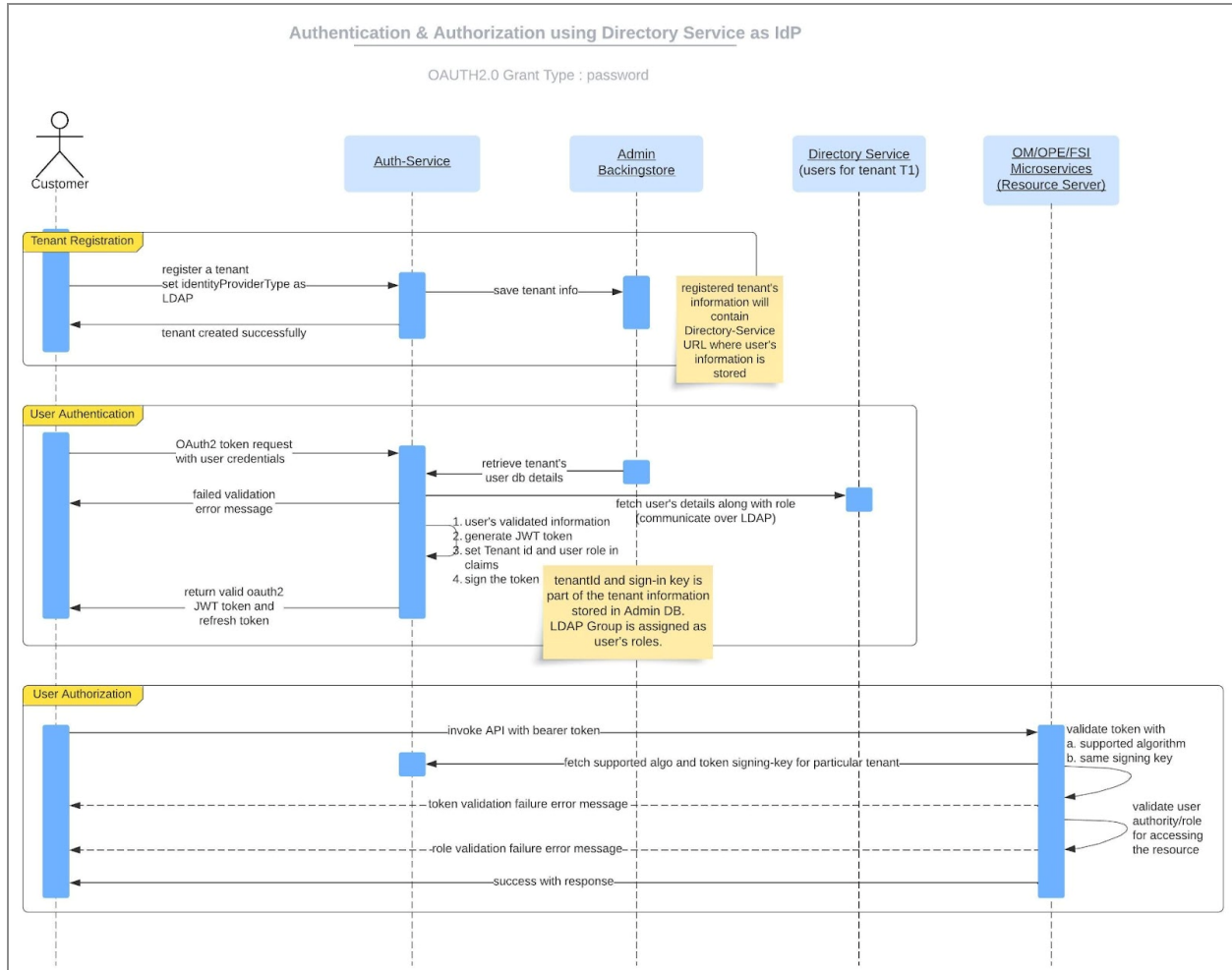
The following sample is shown for RelationalSchema (Postgres/Oracle) identityProviderType:

```

{
 "tenantId": "TIBCO",
 "clientId": "order",
 "clientSecret": "order",
 "identityProviderType": "POSTGRES",
 "supportAlgorithm": "HS256",
 "signingKey": "100f4c1f-f333-4c25-bd8c-e4809722b6a7",
 "relationalSchema": {
 "dataSourceURL":
 "jdbc:postgresql://localhost:5432/userdb11?currentSchema=userschemall",
 "dataSourceUserName": "user11",
 "dataSourcePassword": "user11"
 }
}

```

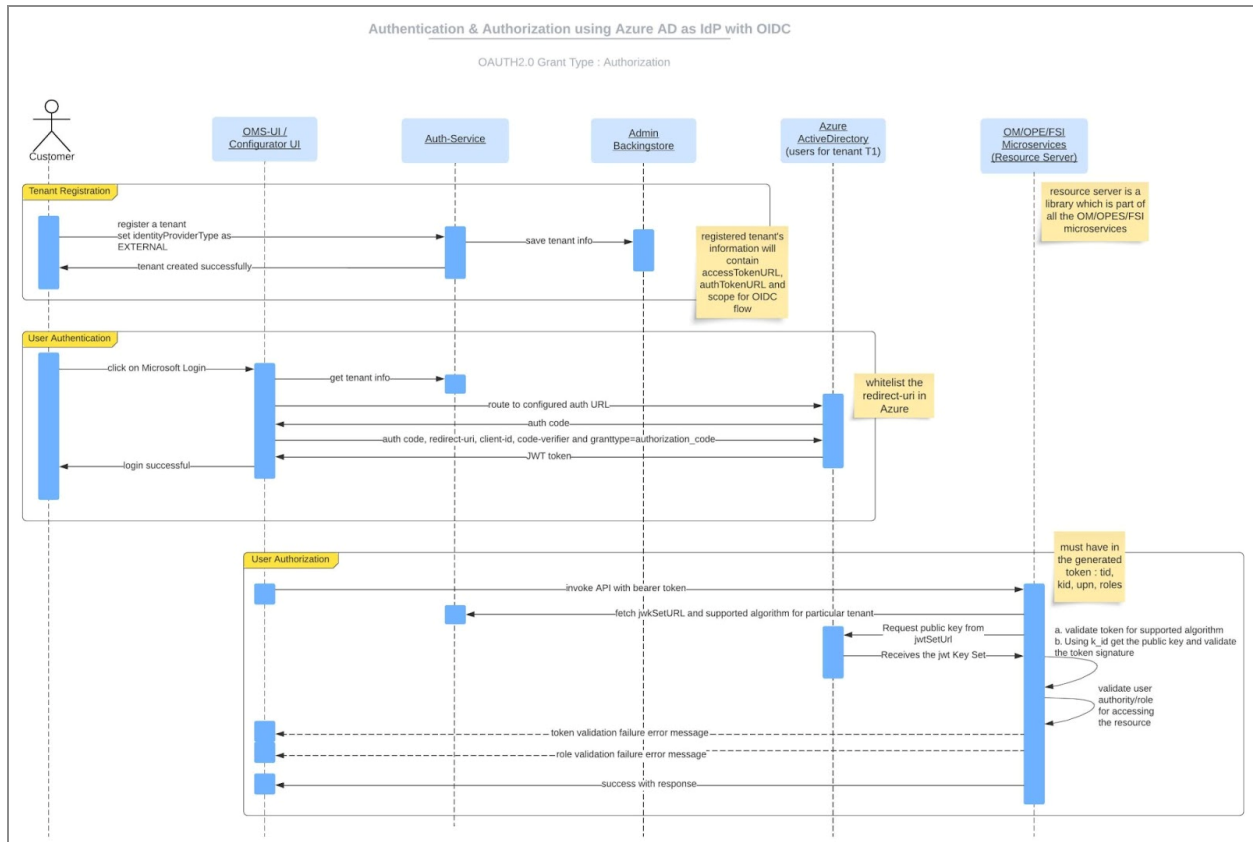
When you have set the identity provider as LDAP, all the users and their roles are maintained in some Directory service.



The following sample is shown for LDAP identityProviderType:

```
{
 "tenantId": "TIBCOLDAP",
 "clientId": "tibco-ldap-client",
 "clientSecret": "tibco-ldap-secret",
 "identityProviderType": "LDAP",
 "supportAlgorithm": "HS256",
 "signingKey": "100f4c1f-f333-4c25-bd8c-e4809722b6a7",
 "ldapSchema": {
 "ldapURLForDirectoryService": "string",
 "directoryServiceDomainName": "string",
 "directoryServiceRootDistinguishedName": "string"
 }
}
```

When you have set identity provider as EXTERNAL, you do not have to use the Order Management's Authentication service for user authentication and token generation. As of now, we support Microsoft Azure Active-Directory as the external authentication service. Even when you have set the identity provider as EXTERNAL, the tenant information is still stored in the Order Management's Authentication service's relational database.



The following sample is shown for EXTERNAL identityProviderType:

```

{
 "tenantId": "string",
 "clientId": "string",
 "clientSecret": "string",
 "identityProviderType": "EXTERNAL",
 "signingKey": "100f4c1f-f333-4c25-bd8c-e4809722b6a7",
 "supportAlgorithm": "RS256",
 "jwkSetUrl": "string",
 "issuer": "string",
 "oidcSchema": {
 "authUrl": "string",
 "accessTokenUrl": "string",
 "scope": "string"
 }
}

```

```
}
}
```

Authorization service can generate a token for all tenants. Each tenant can have a different token algorithm. The following algorithms are supported:

- HMAC (HS256, HS384, HS512)
- RSA (RS256, RS384, RS512)

Order Management Authorization service generates token with HS256. All services can decode or handle any of the above algorithms.



**Note:**

- Supported algorithms must match with one, which is used at the time of registration, This is used for validating tokens (Only in the case of RSA).
- Issuer is validated during registration while validating the token.

## Update tenant information

This operation updates tenant information if the tenant details are already present in database.

**Method:** HTTP PUT

**Endpoint:** `http://<host_address>:<port_address>/v1/tenant`

### Get User Parameters

Parameter	Cardinality	Description
X-API-AppId	Mandatory	The application ID is used for getting the user details.
X-API-Key	Mandatory	This key is used for getting the user details.

For more information on various identityProviderType scenarios, see the sample from the 'Register tenant' topic in *TIBCO® Order Management Web Services Guide*.



## Get tenant information

This operation is used to get the tenant information if it is already present in database.

**Method:** HTTP GET

**Endpoint:** `http://<host_address>:<port_address>/v1/tenant`

*Get User Parameters*

Parameter	Cardinality	Description
tenantId	Mandatory	This is the TENANT value as stored in the users table in the database.
X-API-AppId	Mandatory	The application ID is used for getting the user details.
X-API-Key	Mandatory	This key is used for getting the user details.

## Delete tenant

This operation deletes tenant information if the tenant is already present in database.

**Method:** HTTP DELETE

**Endpoint:** `http://<host_address>:<port_address>/v1/tenant`

*Get User Parameters*

Parameter	Cardinality	Description
tenantId	Mandatory	This is the TENANT value as stored in the users table in the database.
X-API-AppId	Mandatory	The application ID is used for getting the user details.
X-API-Key	Mandatory	This key is used for getting the user details.

## Create User

This request is used to create users.

**Method:** HTTP POST method

**Endpoint:** http://<host\_address>:<port\_address>/v1/user

#### Create User Parameters

Parameter		Cardinality	Description
tenantId		Mandatory	This is the TENANT value as stored in the users table in the database. If the tenantId is not present in the database, then a new TENANT is created.
X-API-AppId		Mandatory	The application ID is used for getting user details. The default ID is auth.
X-API-Key		Mandatory	This key is used for getting user details. The default ID is auth.
userInfo (Body)	enabled	Mandatory	The value can be true or false. true makes the user accessible through the configurator and Order Management System UI and false makes the user disable.
	password	Mandatory	The password to be used for the user.
	Username	Mandatory	It specifies the user name to be created or modified.
	userRoles	Mandatory	It assigns the role to the user.  The default valid role values are ROLE_ADMIN and ROLE_USER. You can override the default roles if required.



**Note:** If the userName and tenantId provided in the request exist, then the user is modified with the provided values.

Example for the Create User request:

```
{
 "user": [
 {
 "password": "string",
 "userName": "string",
 "enabled": true,
 "userRoles": [
 "string"
]
 }
]
}
```

## Update User

This request is used to update an existing user.

**Method:** HTTP PUT method

**Endpoint:** `http://<host_address>:<port_address>/v1/user`

### *Update User Parameters*

Parameter		Cardinality	Description
tenantId		Mandatory	This is the TENANT value as stored in the users table in the database. If the tenantId is not present in the database, then a new TENANT is created.
X-API-AppId		Mandatory	The application ID is used for getting user details. The default ID is auth.
X-API-Key		Mandatory	This key is used for getting user details. The default ID is auth.
userInfo (Body)	enabled	Mandatory	The value can be true or false. true makes the user accessible through the configurator and Order Management System UI and false makes the user disable.

Parameter	Cardinality	Description
password	Mandatory	The password to be used for the user.
userName	Mandatory	It specifies the user name to be created or modified.
userRoles	Mandatory	It assigns the role to the user. The valid role values are ROLE_ADMIN and ROLE_USER.



**Note:** If the userName and tenantId provided in the request exist, then the user is modified with the provided values.

Example for the Update User request:

```
{
 "user": [
 {
 "password": "string",
 "userName": "string",
 "enabled": true,
 "userRoles": [
 "string"
]
 }
]
}
```

## Get User

This request is used to get the details of the existing user.

**Method:** HTTP GET method

**Endpoint:** [http://<host\\_address>:<port\\_address>/v1/user](http://<host_address>:<port_address>/v1/user)

*Get User Parameters*

Parameter	Cardinality	Description
X-API-AppId	Mandatory	The application ID is used for getting the user details. The default ID is auth.
X-API-Key	Mandatory	This key is used for getting the user details. The default ID is auth.
tenantId	Mandatory	This is the TENANT value as stored in the users table in the database.
userId	Mandatory	This is the username value as stored in the users table in the database.

## Delete User

This request is used to delete the existing user.

**Method:** HTTP DELETE method

**Endpoint:** `http://<host_address>:<port_address>/v1/user`

*Delete User Parameters*

Parameter	Cardinality	Description
tenantId	Mandatory	This is the tenant value as stored in the users table in the database.
X-API-AppId	Mandatory	The application ID is used for getting user details. The default ID is auth.
X-API-Key	Mandatory	This key is used for getting user details. The default ID is auth.
userInfo (Body)	userName Mandatory	It specifies the user name to be deleted.

Example for Delete User request:

```
[
 {
 "userName": "testuser",
 }
]
```

## Generating an authorization token

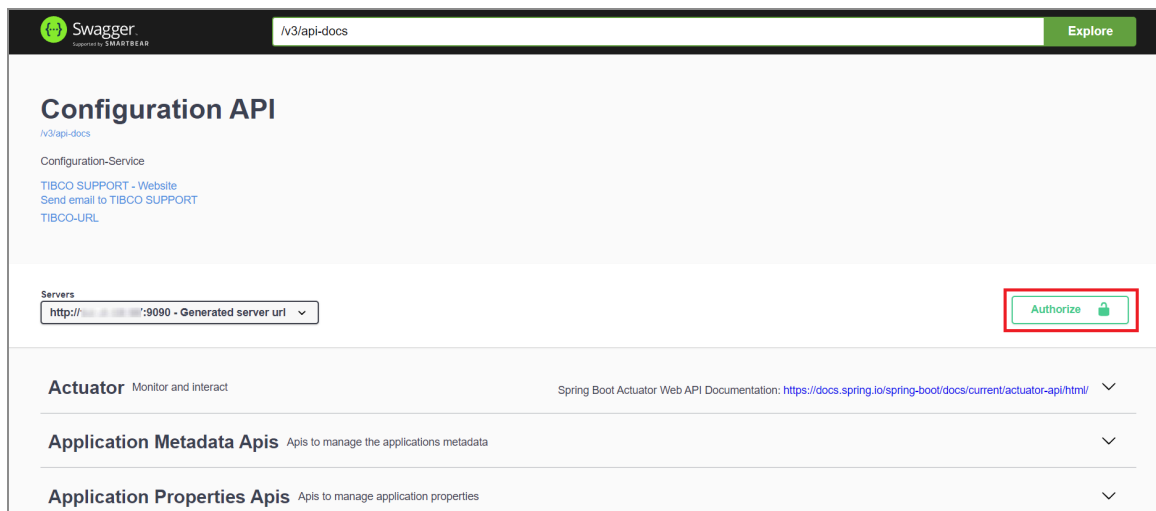
This token can be used to access the operations of all the services like data service, catalog service, orchestrator, and archival service.

### Procedure

1. To authorize a particular service, open the REST API home page of that service in a browser.

**Note:** If the `enableSecureAPI` value is set as `false`, the authentication is bypassed, and you do not have to authorize the service. For the REST services, the authorization token is not required. However, you must provide the `tenantID`.

2. Click the **Authorize** button.



The **Available authorizations** window opens.

### 3. Pass the following mandatory parameters:

#### *Authorization parameters and description*

Element Name	Element Type	Description
user name	String	username@tenantId
password	String	Existing password
Client credentials location		Select Authorization header or Request body from the dropdown options.
client_id	String	as provided in Tenant Registration
client_secret	String	as provided in Tenant Registration

### Available authorizations x

Scopes are used to grant an application different levels of access to data on behalf of the end user.  
Each API may declare one or more scopes.  
API requires the following scopes. Select which ones you want to grant to Swagger UI.

#### OAuth Password (OAuth2, password)

Token URL: <http://10.10.10.10:9091/oauth/token>  
Flow: password

**username:**

**password:**

**Client credentials location:**

Authorization header ▾

**client\_id:**

**client\_secret:**

**Scopes:** [select all](#) [select none](#)

☐ *read*  
*read scope*

☐ *write*  
*write scope*

Authorize

Close

4. Select the **read** and **write** checkboxes as per the requirements and then click the **Authorize** button.

## Result

An authorization token is generated for the particular service. This token is unique and valid only for the dedicated user with tenant ID. The access token comes with an expiry.



## Authorization Token APIs

- [Generate OAuth token](#)

**Note:**

- If you use an External Auth service, then User Management and Token Generation do not work. For this, use POSTMAN as Swagger authentication does not work.
- If you use Azure, the token is generated by Azure and not by Order Management Authorization service.
- OIDC works only with SSL.
- When you have chosen OIDC, Swagger cannot handle the OIDC flow.
- In the case of OIDC, if the token is expired, it generates an error.
- Client credentials cannot be handled via the Swagger.

### Generate OAuth Token

This request is used to generate authorization OAuth token.

**Method:** HTTP POST method

**Endpoint:** `http://<host_address>:<port_address>/oauth/token`

*Generate Authorization Header Parameters*

Parameter	Cardinality	Description
grant_type	Mandatory	You can select password or refresh token.
scope	Mandatory	You can select read, write, or 'read write'.
refresh_token		Refresh token from previously generated token. Required only when grant_type=refresh_token
user name		Required only when grant_type=password
password		Required only when grant_type=password

Parameter	Cardinality	Description
tenantId		Required only when grant_type=password
Authorization	Mandatory	
Content-Type	Mandatory	

## Audit Trail

Audit trail can be enabled or disabled by using the configuration parameter in ConfigValues\_OrchService.JSON and ConfigValues\_ArchivalService.JSON files:

```
{
 "propName": "enableAuditTrail",
 "propDescription": "Enable Audit Trail Persistence",
 "propValue": "true",
 "valueType": "boolean",
 "isTenantProperty": "false"
}
```

## Enabling Internal Error Handler Support

You can enable Internal Error Handler by configuring the values in ConfigValues\_OrchService.JSON file.

```
{
 "propName": "com.tibco.fom.orch.pcErrorHandlerType",
 "propDescription": "The Error Handler component to
be used in case of failed plan item",
 "allowedValues": [
 "ExternalErrorHandler",
 "InternalErrorHandler"
],
 "propValue": "ExternalErrorHandler",
 "valueType": "string",
 "isTenantProperty": "false"
}
```

This is a new property introduced for configuring Internal Error Handlers. We can have two values:

- ExternalErrorHandler (default)
- InternalErrorHandler

When it is configured as ExternalErrorHandler the user's implementation of error handler is considered, which means the on plan-item failure is handled by the error handler defined by the user.

When the property is configured as InternalErrorHandler, it invokes the plan-item failure response and newly created error handler in Order Management Server.

## Logging

Logging is used to record information about a program's execution. This information is typically used for debugging purposes, and additionally, depending on the type and detail of information contained in a trace log, to diagnose common problems with the software.

## How Logging Works

The types of logging for each component are as follows:

- Local Logging: This refers to writing the log output to a local log file for every component.

The logging can be effective by standardizing the contents of a log message for logging data across all the components.

The logging services for TIBCO Order Management are as follows:

- aopd (aopd.log)
- archival-service (archival-service.log)
- authorization-service (authorization-service.log)
- catalog-services (catalog-services.log)
- configurator (configurator.log)
- data-service (dataservice.log)

- encryptPWDUtility.log
- om-migration (migration.log, migration-starter.log)
- orchestrator (orchestrator.log)
- processcomponent (processcomponent.log)
- tmf-om-adapter (tmf-om-adapter.log)
- broker-service (broker-service.log)
- jeopardy (jeoms.log)

## Contents of the Log Message

The log message is composed of several log components that are required to explain the log message in its entirety. These log message components help you analyze the log.

Log Message Component	Description
Log level	The levels are as follows: DEBUG, INFO, WARN, ERROR, OFF.
BusinessTransactionId	Unique identifier for tracing purposes across function calls.
OrderRef	An identifier to identify the order for which this log message is written.
Component	Context information about the origin of the log (typically the engine name).
Service	Context information about the origin of the log (typically the class name).
Operation	Context information about the origin of the log (typically the method name).
StackTrace	Entire stack trace of the activity.
TimeStamp	Indicates when the message was logged.

## APIs for Changing log-level

Previously, when you wanted to change a log level for a class, you had to change it in the logback file and restart the server to get the updates. This was taking a lot of time and effort in this process.

To overcome this issue, the following log-level APIs are introduced in this release:

- **Get all logger details**

```
curl -X 'GET' \
'http://<host>:<port>/management/loggers' \
-H 'accept: */*'

```

- **Get logger for a specific class**

```
curl -X 'GET' \
'http://<host>:<port>/management/loggers/com.tib.fom' \
-H 'accept: */*'

```

- **Change log level for a specific class**

```
curl --location --request POST \
'http://<host>:<port>/management/loggers/com.tib.fom' \
--header 'Content-Type: application/json' \
--data-raw ' {"configuredLevel": "DEBUG"}'

```

## Configuring Logging for Java Components

Logging configuration for the following services applies to Automated Order Plan Development, authorization-service, catalog-services, data-service, orchestrator, configurator, process component, and tmf-om-adapter.

Logging is done using the log back framework. Each component has a separate file for log back configurations as explained as follows.

**Orchestrator:** \$OM\_HOME/roles/orchestrator/standalone/config/logback\_orch.xml is used to configure logging.

**Authorization-service:** \$OM\_HOME/roles/authorization-service/standalone/config/logback\_auth.xml is used to configure logging.

**catalog-services:** \$OM\_HOME/roles/catalog-services/standalone/config/logback\_catalog.xml is used to configure logging.

**Data-service:** \$OM\_HOME/roles/dataservice/standalone/config/logback\_dataservice.xml is used to configure logging.

**Tmf-om-adapter:** \$OM\_HOME/roles/tmf-om-adapter/standalone/config/logback\_tmf.xml is used to configure logging.

**Configurator:** \$OM\_HOME/roles/configurator/standalone/config/logback.xml is used to configure logging.

**Om-migration service:** \$OM\_HOME/roles/om-migration/standalone/config/logback\_migration.xml is used to configure logging.

**Broker service:** \$OM\_HOME/roles/broker-service/standalone/config/logback\_broker.xml is used to configure logging.

**Archival service:** \$OM\_HOME/roles/configurator/standalone/config/logback\_arch.xml is used to configure logging.

**Jeopardy service:** \$OM\_HOME/roles/configurator/standalone/config/logback\_jeopardy.xml is used to configure logging.

- The default logLevel is:
  - INFO for com.tib.fom package and its sub package
  - ERROR for all other packages
- The local log file used by the orchestrator is orchestrator.log. Published logs go into the logs folder for each service, the same as for the other components.
- The default maximum file size is 5 mb. After 5 mb, a new file is created.
- The default logging is orchestrator.log file. The next day those logs are moved to orchestrator-`<date>`.log file and current logging starts in orchestrator.log file.

**Automated Order Plan Development:** \$OM\_HOME/roles/aopd/standalone/config/Logback\_aopd.xml is used to configure logging.

- The default logLevel is:
  - INFO for com.tibco.aff, com.tibco.fom, and com.tibco.aff.models packages and its sub packages
  - ERROR for all other packages

- Local log file used by Automated Order Plan Development.

**i Note:** You can set the `catalogHibernateShowSql` property to control whether hibernate logs SQL statements. You can enable or disable SQL logging without changing the codes.

## Configuring Redis

Redis is supported only for the Order and Catalog services. A relational database is used for the Admin, Archival, and Jeopardy services.

To start the services in Redis, perform the following steps:

### Procedure

1. In the `$OM_HOME/externalLib/seed-data/app-properties/ConfigValues_common.JSON` file, update the `cmPluggableCache` and `omPluggableCache` properties to `redis`.
2. In the `$OM_HOME/externalLib/seeddata/app-properties/ConfigValues_CatalogService.JSON` file under the Redis Data Source Configuration category, update the following catalog Redis-related properties:

Property Name	Description
<code>catalogRedisClientName</code>	Redis server ClientName for Catalog datasource
<code>catalogRedisClusterEnabled</code>	Whether Redis is running in cluster mode
<code>catalogRedisDatabase</code>	Redis server database name for Catalog datasource
<code>catalogRedisHost</code>	Redis server host for storing Catalog models

Property Name	Description
catalogRedisPassword	Password to connect to Redis cluster/node
catalogRedisPort	Redis server port for storing Catalog models
catalogRedisSslEnabled	Connect to Redis cluster or node via SSL
catalogRedisUsername	User name to connect to Redis cluster/node
redisBlockedWhenExhausted	Enable connection blocking when the connection pool is exhausted
redisJmxEnabled	Enable JMX for connections
redisKeyStoreAbsoluteFileName	Redis SSL KeyStore absolute file name
redisKeyStorePassword	Redis SSL KeyStore password
redisKeyStoreType	Redis SSL KeyStore type
redisLifo	Enable LIFO behavior for idle objects, always returning the most recently used object from the pool
redisMaxIdle	Maximum number of idle connections in the pool
redisMaxTotal	Maximum number of connections that can be allocated by the pool at a given time
redisMinEvictableIdleTimeMillis	Minimum amount of time an object might sit idle in the pool, in milliseconds



Property Name	Description
redisMinIdle	Minimum number of idle connections to maintain in the pool
redisNumTestsPerEvictionRun	Maximum number of connections to examine during each eviction run
redisSoftMinEvictableIdleTimeMillis	Minimum amount of time an object might sit idle in the pool if <code>minIdle</code> instances are available, in milliseconds
redisTestOnBorrow	Enable connection validation before being borrowed from the pool
redisTestOnReturn	Enable connection validation before being returned to the pool
redisTestWhileIdle	Enable connection validation when idle in the connection pool
redisTimeBetweenEvictionRunsMillis	Number of milliseconds to sleep between runs of the idle object evictor thread
redisTrustStoreAbsoluteFileName	Redis SSL TrustStore absolute file name
redisTrustStorePassword	Redis SSL TrustStore password
redisTrustStoreType	Redis SSL TrustStore type

3. In the `$OM_HOME/externalLib/seed-data/app-properties/ConfigValues_AopdService.JSON` file under the Redis Data Source Configuration category, update the catalog Redis-related properties.

Property Name	Description
catalogRedisClientName	Redis server ClientName for Catalog datasource
catalogRedisClusterEnabled	Checks whether Redis is running in cluster mode
catalogRedisDatabase	Redis server database name for Catalog datasource
catalogRedisHost	Redis server host for storing Catalog models
catalogRedisPassword	Password to connect to Redis cluster/node
catalogRedisPort	Redis server port for storing Catalog models
catalogRedisSslEnabled	Connect to Redis cluster/node via SSL
catalogRedisUsername	User name to connect to Redis cluster/node
redisBlockedWhenExhausted	Enable connection blocking when the connection pool is exhausted
redisJmxEnabled	Enable JMX for connections
redisKeyStoreAbsoluteFileName	Redis SSL KeyStore absolute file name
redisKeyStorePassword	Redis SSL KeyStore password

Property Name	Description
redisKeyStoreType	Redis SSL KeyStore type
redisLifo	Enable LIFO behavior for idle objects, always returning the most recently used object from the pool
redisMaxIdle	Maximum number of idle connections in the pool
redisMaxTotal	Maximum number of connections that can be allocated by the pool at a given time
redisMinEvictableIdleTimeMillis	Minimum amount of time an object might sit idle in the pool, in milliseconds
redisMinIdle	Minimum number of idle connections to maintain in the pool
redisNumTestsPerEvictionRun	Maximum number of connections to examine during each eviction run
redisSoftMinEvictableIdleTimeMillis	Minimum amount of time an object might sit idle in the pool if <code>minIdle</code> instances are available, in milliseconds
redisTestOnBorrow	Enable connection validation before being borrowed from the pool

Property Name	Description
redisTestOnReturn	Enable connection validation before being returned to the pool
redisTestWhileIdle	Enable connection validation when idle in the connection pool
redisTimeBetweenEvictionRunsMillis	Number of milliseconds to sleep between runs of the idle object evictor thread
redisTrustStoreAbsoluteFileName	Redis SSL TrustStore absolute file name
redisTrustStorePassword	Redis SSL TrustStore password
redisTrustStoreType	Redis SSL TrustStore type

4. In the \$OM\_HOME/externalLib/seed-data/app-properties/ConfigValues\_OrchService.JSON file under the Redis Data Source Configuration category, update the catalog and order Redis-related properties .

Property Name	Description
orderRedisHost	Redis server host for storing order data
orderRedisPort	Port number of the Redis server for storing order models
orderRedisUsername	User name required to connect to the Redis cluster/node
orderRedisPassword	Password for connecting to the Redis cluster/node

Property Name	Description
orderRedisDatabase	Name of the Redis database for the order data source
orderRedisClientName	Name of the Redis client for the order data source
orderRedisSslEnabled	Enables SSL connection to the Redis cluster/node
orderRedisKeyStoreType	Type of the SSL KeyStore for Redis connections
orderRedisKeyStorePassword	Password for the SSL KeyStore
orderRedisTrustStoreType	Type of the SSL TrustStore for Redis connections
orderRedisTrustStorePassword	Password for the SSL TrustStore
orderRedisKeyStoreAbsoluteFileName	Absolute file name of the SSL KeyStore
orderRedisTrustStoreAbsoluteFileName	Absolute file name of the SSL TrustStore
redisStatsHost	Host address of the Redis server for storing order statistics
redisStatsPort	Port number of the Redis server for storing order statistics
redisStatsUsername	User name required to connect to the Redis cluster/node for statistics
redisStatsPassword	Password for connecting to the Redis cluster/node for statistics

Property Name	Description
redisStatsDatabase	Name of the Redis database for storing order statistics
redisStatsClientName	Name of the Redis client for storing order statistics
redisStatsSslEnabled	Enables SSL connection to the Redis cluster/node for statistics
redisStatsKeyStoreType	Type of the SSL KeyStore for Redis connections for statistics
redisStatsKeyStorePassword	Password for the SSL KeyStore for statistics
redisStatsTrustStoreType	Type of the SSL TrustStore for Redis connections for statistics.
redisStatsTrustStorePassword	Password for the SSL TrustStore for statistics
redisStatsKeyStoreAbsoluteFileName	Absolute file name of the SSL KeyStore for statistics
redisStatsTrustStoreAbsoluteFileName	Absolute file name of the SSL TrustStore for statistics
catalogRedisClientName	Redis server ClientName for Catalog datasource
catalogRedisClusterEnabled	Whether Redis is running in cluster mode
catalogRedisDatabase	Redis server database name for Catalog datasource

Property Name	Description
catalogRedisHost	Redis server host for storing Catalog models
catalogRedisPassword	Password to connect to Redis cluster/node
catalogRedisPort	Redis server port for storing Catalog models
catalogRedisSslEnabled	Connect to Redis cluster/node via SSL
catalogRedisUsername	User name to connect to Redis cluster/node
redisBlockedWhenExhausted	Enable connection blocking when the connection pool is exhausted
redisJmxEnabled	Enable JMX for connections
catalogRedisKeyStoreAbsoluteFileName	Redis SSL KeyStore absolute file name
catalogRedisTrustStorePassword	Redis SSL KeyStore password
catalogRedisKeyStoreType	Redis SSL KeyStore type
redisLifo	Enable LIFO behavior for idle objects, always returning the most recently used object from the pool
redisMaxIdle	Maximum number of idle connections in the pool
redisMaxTotal	Maximum number of connections that can be allocated by the pool at a given time

Property Name	Description
redisMinEvictableIdleTimeMillis	Minimum amount of time an object might sit idle in the pool, in milliseconds
redisMinIdle	Minimum number of idle connections to maintain in the pool
redisNumTestsPerEvictionRun	Maximum number of connections to examine during each eviction run
redisSoftMinEvictableIdleTimeMillis	Minimum amount of time an object might sit idle in the pool if minIdle instances are available, in milliseconds
redisTestOnBorrow	Enable connection validation before being borrowed from the pool
redisTestOnReturn	Enable connection validation before being returned to the pool
redisTestWhileIdle	Enable connection validation when idle in the connection pool
redisTimeBetweenEvictionRunsMillis	Number of milliseconds to sleep between runs of the idle object evictor thread
catalogRedisTrustStoreAbsoluteFileName	Redis SSL TrustStore absolute file name
catalogRedisTrustStorePassword	Redis SSL TrustStore password
catalogRedisTrustStoreType	Redis SSL TrustStore type

5. In the `$OM_HOME/externalLib/seed-data/app-properties/ConfigValues_DataService.JSON` file under the Redis Data Source Configuration category, update the order Redis-related properties.



Property Name	Description
orderRedisHost	Host address of the Redis server for storing order data
orderRedisClusterEnabled	Indicates whether Redis is running in cluster mode
orderRedisPort	Port number of the Redis server for storing order models
orderRedisUsername	User name required to connect to the Redis cluster/node
orderRedisPassword	Password for connecting to the Redis cluster/node
orderRedisDatabase	Name of the Redis database for storing order-related information
orderRedisClientName	Name of the Redis client for storing order-related information
redisTestOnBorrow	Enables connection validation before being borrowed from the pool
redisTestOnReturn	Enables connection validation before being returned to the pool
redisTestWhileIdle	Enables connection validation when idle in the connection pool

Property Name	Description
redisBlockedWhenExhausted	Enables blocking of new connection requests when the connection pool is exhausted
redisJmxEnabled	Enables Java Management Extensions (JMX) for monitoring connections
redisLifo	Enables Last In, First Out (LIFO) behavior for managing idle objects in the pool
redisMaxIdle	Sets the maximum number of idle connections in the pool
redisMinIdle	Sets the minimum number of idle connections to maintain in the pool
redisMaxTotal	Sets the maximum number of connections that can be allocated by the pool at any time
redisNumTestsPerEvictionRun	Sets the maximum number of connections to examine during each eviction run
redisSoftMinEvictableIdleTimeMillis	Sets the minimum amount of time, in milliseconds, an object might sit idle in the pool if `minIdle` instances are available
redisMinEvictableIdleTimeMillis	Sets the minimum evictable

Property Name	Description
	idle time, in milliseconds, for objects in the pool
redisTimeBetweenEvictionRunsMillis	Sets the time, in milliseconds, between eviction runs for idle objects in the pool
orderRedisSslEnabled	Enables SSL connection to the Redis cluster/node
redisKeyStoreType	Specifies the type of the SSL KeyStore for Redis connections
redisKeyStorePassword	Password for the SSL KeyStore
redisTrustStoreType	Specifies the type of the SSL TrustStore for Redis connections
redisTrustStorePassword	Password for the SSL TrustStore
redisKeyStoreAbsoluteFileName	Absolute file name of the SSL KeyStore
redisTrustStoreAbsoluteFileName	Absolute file name of the SSL TrustStore

## Configuring Microsoft SQL Server

Perform the following steps to configure Microsoft SQL Server.

### Admin Database

#### Procedure

1. Open the `$OM_HOME/db/dbscripts/sqlServer/admin/bin/sqlserver_admin_db.properties` file in a suitable editor and update the values.
2. Run the following scripts from the `$OM_HOME/db/dbscripts/sqlServer/admin/bin` directory:

**db-setup.sh**

**seed\_common\_authConfig\_db\_setup.sh**

## Archival Database

### Procedure

1. Open the `$OM_HOME/db/dbscripts/sqlServer/archival/bin/sqlserver_archival_db.properties` file in a suitable editor and update the values.
2. Run the following script from the `$OM_HOME/db/dbscripts/sqlServer/archival/bin` directory:

**db-setup.sh**

## Catalog Database

### Procedure

1. Open the `$OM_HOME/db/dbscripts/sqlServer/catalog/bin/sqlserver_catalog_db.properties` file in a suitable editor and update the values.
2. Run the following script from the `$OM_HOME/db/dbscripts/sqlServer/catalog/bin` directory:

**db-setup.sh**

## Jeopardy Database

### Procedure

1. Open the `$OM_HOME/db/dbscripts/sqlServer/jeopardy/bin/sqlserver_jeopardy_db.properties` file in a suitable editor and update the values.
2. Run the following script from the `$OM_HOME/db/dbscripts/sqlServer/jeopardy/bin` directory:

**db-setup.sh**

## Order Database

### Procedure

1. Open the `$OM_HOME/db/dbscripts/sqlServer/order/bin/sqlserver_order_db.properties` file in a suitable editor and update the values.
2. Run the following script from the `$OM_HOME/db/dbscripts/sqlServer/order/bin` directory:

**db-setup.sh**

## User Database

### Procedure

1. Open the `$OM_HOME/db/dbscripts/sqlServer/user/bin/sqlserver_user_db.properties` file in a suitable editor and update the values.
2. Run the following script from the `$OM_HOME/db/dbscripts/sqlServer/user/bin` directory:

**db-setup.sh**

Provide the SQL Server details in each of the configuration files. For more details, see the `$OM_HOME/samples/sqlServer-sample-property/sample_sqlServer_properties.properties` file.

# Configuring an External Identity Provider

### Before you begin

You must have registered an application in the external authentication provider such as Azure Active-Directory(ADD) or Google Identity with all the required details for the application role assignment to the user.

### Registering a Tenant

You can configure an external authentication provider with TIBCO® Order Management. See the "Multitenancy" topic in the *TIBCO® Order Management Administration* guide.

Register a tenant in the Authorization service using the POST method of the `/v1/tenant` API with the following sample payload, which shows all the mandatory fields.

```
{
 "tenantId": "cde6fa59-abb3-471-be01-2443c417cbda",
 "clientId": "ddaf41fb-3aef-4e30-879f-a188ba131abf",
 "clientSecret": "DI68Q~tljTkt4ABi7lZVztaz5AUN6A6r.CGJHbwd",
 "identityProviderType": "EXTERNAL",
 "supportAlgorithm": "RS256",
 "jwkSetUrl": "https://login.microsoftonline.com/cd-abb3-4971-be01-244bda/discovery/v2.0/keys",
 "issuer": "https://sts.windows.net/cde6fa59-abb3-4971-be01-2443c417cbda/",
 "oidcSchema": {
 "authUrl": "https://login.microsoftonline.com/cd-abb3-4971-be01-244bda/oauth2/v2.0/authorize",
 "accessTokenUrl": "https://login.microsoftonline.com/cd-abb3-4971-be01-244bda/oauth2/v2.0/token",
 "scope": "ddaf41fb-3aef-4e30-879f-a188ba131abf-serviceB/fosApplicationConsent"
 }
}
```

## Mapping of Keys from External OAuth2 Token

In the `$OM_HOME/seed-data/app-properties/ConfigValues_Common.json` file, update the following properties. These are the properties that need to be mapped to the KEY of CLAIMS in an externally generated OAuth2 token.

```
{
 "propName": "tenantIdMapping",
 "propDescription": "key in the token claims that refers to tenantId",
 "propValue": "TENANTID",
 "valueType": "string",
 "isTenantProperty": "false"
},
{
 "propName": "userNameMapping",
 "propDescription": "key in the token claims that refers to userName",
 "propValue": "user_name",
 "valueType": "string",
 "isTenantProperty": "false"
},
```

```
{
 "propName": "userRoleMapping",
 "propDescription": "key in the token claims that refers to
userRole",
 "propValue": "authorities",
 "valueType": "string",
 "isTenantProperty": "false"
}
```

## Role-Based Access Configurations

Irrespective of whether you want to use an external or Order Management's authentication provider, you have to modify the authorization for each API in Order Management.

Each API in Order Management has role-based access. API-related role mapping is available in the respective microservice's configuration. To configure API-specific roles, you must modify the following configurations for each of the mentioned microservices:

- **Authorization service:**

File name: \$OM\_HOME/roles/authorization-service/standalone/config/application.properties

Property name: allowedUserRoles

Description: All the user roles (comma separated) have access to the APIs exposed in the authorization service. This is not considered when you have used identityProviderType as EXTERNAL while registering the tenant. When identityProviderType is set as EXTERNAL, the authorization service is not used to generate the token.

- **Configurator:**

File name: \$OM\_HOME/roles/configurator/standalone/config/application.properties

Property name: configuratorAccessRoles

Description: All the user roles (comma separated) have access to the APIs exposed as part of the configurator microservice.

- **OMS UI:**

File name: \$OM\_HOME/seed-data/config-files/ConfigValues\_OMSUI.json

Property name: com.tibco.fom.orch.roles.piExecutionToComplete

Description: To force completing plan items in execution status on the basis of role.

- **Archival service:**

File name: \$OM\_HOME/seed-data/config-files/ConfigValues\_ArchivalService.json

Under the Application Security Configurations category name, update the following properties:

Property Name	Description
operation.roles.orderSummary	User Role for orderSummary API
operation.roles.ordersByCriteria	User Role for ordersByCriteria API
operation.roles.auditTrail	User Role for auditTrailForPlan API
operation.roles.planByCriteria	User Role for planByCriteria API
operation.roles.purgeOrders	User Role for purgeOrders API

Once you configure these mentioned properties, replies from all the above APIs would only be received if the user accessing the resource belongs to one of the user groups that have access to the specified resource.

- **Orchestrator:**

File name: \$OM\_HOME/seed-data/config-files/ConfigValues\_OrchService.json

Under the Application Security Configurations category name, update the following properties:



Property Name	Description
operation.roles.submitOrder	User Role for submitOrder Service
operation.roles.orderExecutionPlan	User Role for getOrderExecutionPlan Service
operation.roles.getOrderDetails	User Role for getOrderDetails Service
operation.roles.executePlanItemReply	User Role for planItemExecuteResponse Service
operation.roles.orderWithdraw	User Role for orderWithdraw Service
operation.roles.submitOrderExecutionPlan	User Role for submitOrderExecutionPlan Service
operation.roles.milestoneNotifyRequest	User Role for milestoneNotifyRequest Service
operation.roles.planItemSuspendResponse	User Role for planItemSuspendResponse Service
operation.roles.amendOrder	User Role for amendOrder Service
operation.roles.purgeOrder	User Role for purgeOrder Service
operation.roles.orderSuspend	User Role for orderSuspend

Property Name	Description
	Service
operation.roles.orderActivate	User Role for orderActivate Service
operation.roles.planItemErrorHandler	User Role for planItemErrorHandler Service
operation.roles.planItemBulkErrorHandler	User Role for planItemBulkErrorHandler Service
operation.roles.preQualificationFailedReply	User Role for preQualificationFailedReply Service
operation.roles.feasibilityReply	User Role for feasibilityReply Service
operation.roles.orderCancel	User Role for orderCancel Service
operation.roles.performBulkOrderAction	User Role for performBulkOrderAction Service
operation.roles.orderScXml	User Role for orderScXml Service
operation.roles.planFragments	Roles of the user used by getPlanFragments
operation.roles.submitPlanErrorNotification	Roles of the user used by planErrorNotification
operation.roles.opdErrorHandlerReply	Roles of the user used by opdErrorHandlerReply

- **Catalog service:**

File name: \$OM\_HOME/seed-data/config-files/ConfigValues\_CatalogService.json

Under the Application Security Configurations category name, update the following properties:

Property Name	Description
operation.roles.submitPlanFragmentModel	User Role to access submitPlanFragmentModel Service
operation.roles.submitProductModel	User Role to access submitProductModel Service
operation.roles.submitActionModel	User Role to access submitActionModel Service
operation.roles.purgePlanFragmentModel	User Role to access purgePlanFragmentModel Service
operation.roles.purgeProductModel	User Role to access purgeProductModel Service
operation.roles.purgeActionModel User	Role to access purgeActionModel Service
operation.roles.getProductModelRoles	User Role to access getProductModelRoles Service
operation.roles.getPlanFragmentModelRoles	User Role to access getPlanFragmentModelRoles Service
operation.roles.getActionModelRoles	User Role to access getActionModelRoles Service

Property Name	Description
operation.roles.getAllActionModelRoles	User Role to access getAllActionModelRoles Service
operation.roles.getAllPriceModelRoles	User Role to access getAllPriceModelRoles Service
operation.roles.getAllOfferIdsModelRoles	User Role to access getAllOfferIdsModelRoles Service
operation.roles.getAllDiscountModelRoles	User Role to access getAllDiscountModelRoles Service
operation.roles.getAllProductModelRoles	User Role to access getAllProductModelRoles Service
operation.roles.getAllPlanFragmentModelRoles	User Role to access getAllPlanFragmentModelRoles Service
operation.roles.getAllCategoryModelRoles	User Role to access getAllCategoryModelRoles Service
operation.roles.submitPriceModel	User Role to access submitPriceModel Service
operation.roles.submitDiscountModel	User Role to access submitDiscountModel Service
operation.roles.submitOfferIdsModel	User Role to access submitOfferIdsModel Service
operation.roles.submitCategoryModel	User Role to access submitCategoryModel Service
operation.roles.submitRuleModel	User Role to access

Property Name	Description
	submitRuleModel Service
operation.roles.purgePriceModel	User Role to access purgePriceModel Service
operation.roles.purgeDiscountModel	User Role to access purgeDiscountModel Service
operation.roles.purgeCategoryModel	User Role to access purgeCategoryModel Service
operation.roles.purgeOfferId	User Role to access purgeOfferId Service
operation.roles.purgeRuleModel	User Role to access purgeRuleModel Service

- **Broker service:**

File name: \$OM\_HOME/seed-data/config-files/ConfigValues\_BrokerService.json

Under the Application Security Configurations category name, update the following properties:

Property Name	Description
operation.roles.getPendingNotifications	User Role to access the getPendingNotifications Service
operation.roles.resumePendingNotifications	User Role to access the resumePendingNotifications Service
operation.roles.resumeAllPendingNotifications	User Role to access the resumeAllPendingNotifications Service

# Administration Tasks

---

This section covers all the administration tasks for TIBCO Order Management.

## Swagger API Reference

**Swagger version:** Open API - 3.0.1

**URL to access the Swagger UI:** `scheme://host[:port]/swagger-ui/index.html`



**Note:** To disable the Swagger UI, expose the following property either via an environment variable or configurator: `springdoc.api-docs.enabled=false`

## Docker

You can containerize TIBCO Order Management components and run them on hosts that support the Docker environment. The Docker files are delivered as part of the TIBCO Order Management installer. You can build images using those Docker files and then run them as containers.

This feature of TIBCO Order Management requires Docker version 25.0.x and Docker-Compose version 1.29.0 (or later).

It is required to have an internet connection on the machine where you install and run Docker.



**Note:** The term *Docker Context* refers to the directory where the Dockerfile is available. For example, the Docker context for the Order Management Server is `$OM_HOME/docker/orchestrator/6.1.0`.

Depending on your system configuration, you might need the following Docker containers:

- Configurator Service - through the Configurator-UI to make the configuration-related changes
- Authorization Service - to generate the token and this token is used across all Order Management Services to authorize and authenticate the users
- Catalog Service - API to load Product and Plan Fragment models
- Orchestrator Service - order-related APIs
- Automated Order Plan Development Service - API to generate plans for each order
- Data Service - API to modify User-Defined Fields at Plan or Plan Item level
- TM Forum Adapter Service - API to map TM Forum API to Order Management API
- Order Migration Service - to migrate the orders to the 6.1.0 version of Order Management
- Order Management System UI - provides operators a GUI to manage and track orders. Order Management System persists order data and allows operators to search, view, track, and trace orders.
- Archival Service - acts as the data backup for the Orchestrator and it uses messages to achieve this. For every status change in the order, the Orchestrator sends a JMS message.
- Broker Service: to resend pending notifications of an order.

After installation, all Docker-related files are located in the `$OM_HOME/docker` directory when the user's PWD is `$OM_HOME/docker`.

Before you start using the Docker feature in TIBCO Order Management, you must be familiar with the following Docker concepts:

- Docker architecture
- Using Docker in production
- Using Docker Volumes
- Docker commands
- Docker-Compose
- Using Docker-Compose in production

# Building a Docker Image Without an Internet Connection

Download and install `wget` and `unzip` utilities to build a Docker image without an internet connection.

In `$OM_HOME/docker/base/1.0/Dockerfile`, the "FROM `registry.access.redhat.com/ubi8/ubi-minimal`" instruction initializes a new build stage and sets the base image for subsequent instructions. You can accept the default base image or you can change the instruction and provide a valid source for a different base image. You can pull a valid base image from the Docker's public repository or you can create your base image, push it to a public or private Docker registry, and then use the newly created image as a base image. For more information about creating your base Docker image, see the [Docker documentation](#) related to Creating a Base Image.

In `$OM_HOME/docker/base/1.0/Dockerfile`, you can find instructions for downloading `wget` and `unzip` utilities. These instructions can be modified to pick up the installers of the utilities from the Docker context and install them in the image.



**Note:** In this case, the Docker context for `$OM_HOME/docker/base/1.0/Dockerfile` is `$OM_HOME/docker/base/1.0`.

## Copying Files to Docker Context

It is required to copy files to the Docker context before building the required Docker images.

### Procedure

1. Install Docker 25.x (or later), Docker-Compose 1.29 (or later), and TIBCO Order Management on the host machine.
2. Run the `$OM_HOME/roles/copyLib.sh` script. For more information, refer to the "Post-Installation Task: Copying Dependencies" topic in the *TIBCO Order Management and Configuration* guide.
3. Go to the `$OM_HOME/docker` directory and run `copy-required-files.sh` script.

This shell script copies all the required directories from `OM_HOME` to a specific Docker context. These files are required to build Docker images. This script works on `OM_`



HOME set in the environment or takes the value of OM\_HOME as user input.

## Building Docker Images

After the required files are copied at the Docker context, you can build the Docker images. Use Docker-Compose for building the Docker images.

### Procedure

1. Go to the `$OM_HOME/docker/base/1.0` directory and execute the following command:

```
docker build -t tibco/base:1.0 --rm=true .
```

2. Go to the `$OM_HOME/docker` directory and execute the following command:

```
docker-compose --file docker-compose-build-complete.yml build
```

3. Run the following command to check the images that are created:

```
$] docker images
```

You might get the following output:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
tibco/base	1.0	78745cb20c85	5 weeks ago	637MB
tibco/authorization-service	6.1.0	1bd76c5c9a54	5 weeks ago	714MB
tibco/dataservice	6.1.0	16eab77f5574	5 weeks ago	758MB
tibco/aopd	6.1.0	0ba6b17a086f	5 weeks ago	798MB
tibco/configurator	6.1.0	298896cc1698	5 weeks ago	721MB
tibco/tmf-om-adapter	6.1.0	f5c7d67f08d4	5 weeks ago	724MB
tibco/catalog-service	6.1.0	a64e23b78572	5 weeks ago	758MB
tibco/processcomponent	6.1.0	0d6927fce027	5 weeks ago	755MB
tibco/broker	6.1.0	89936c29baaf	5 weeks ago	758MB
tibco/orchestrator	6.1.0	ff65b877f462	5 weeks ago	780MB
tibco/jeopardy	6.1.0	d56f8bb14df3	5 weeks ago	801MB
tibco/configurator-ui	6.1.0	1e1c71795298	5 weeks ago	692MB
tibco/archival-service	6.1.0	1d7cfe285633	5 weeks ago	770MB
tibco/omsui	6.1.0	e1285958f064	5 weeks ago	716MB
registry.redhat.io/ubi8/ubi-minimal	latest	12c4198317ec	8 weeks ago	92.5MB

## Setting Up the .env File

Set the required variables, which varies according to the user's environment. All of these variables are required to be changed according to the user's environment.

All of these variables are in the .env file located in the \$OM\_HOME/docker directory.

## Configuring for Order Management Server Docker Containers

When you run the Order Management Server or the Order Management Server UI as Docker containers, you have to make configuration changes using the Configurator UI.

### Procedure

1. Start TIBCO Order Management configurator UI as a Docker container by running the following command:

```
$] docker-compose --file docker-compose-run-configurator-ui.yml up -d
```

You can access the Configurator UI on the port, which you had set for the HOST\_CONFIGURATOR\_UI\_PORT variable in the .env file.

2. On the TIBCO Order Management Configurator UI, make the configuration changes according to your environment. For example, make related configuration for Order Management Server, Automated Order Plan Development, and other related configurations.

All the changes that you do in the Configurator UI are uploaded to the database so other containers can read from it.

## Running the Docker Containers

After building the Docker images, you can run the images as containers to start containerized TIBCO Order Management.

## Running Different Containers for TIBCO Order Management Components

Start the Docker container by using the specific docker-compose file.

### Procedure

1. Start the Configurator Docker container.

```
$> docker-compose --file docker-compose-run-configurator.yml up -d
```

2. Start the Configurator UI Docker container.

```
$> docker-compose --file docker-compose-run-configurator-ui.yml up -d
```

3. Access the configurator UI and configure according to your environment and requirement.
4. Start the authorization service to fetch the token, which is then used across all order management services to authorize and authenticate the user.

```
$> docker-compose --file docker-compose-run-authorization-service.yml up -d
```

You can start any of the Docker services by using the following compose commands.

- a. Start Catalog Service container.

```
$> docker-compose --file docker-compose-run-catalog-service.yml up -d
```

- b. Start Automated Order Plan Development Service container.

```
$> docker-compose --file docker-compose-run-aopd.yml up -d
```

- c. Start Orchestrator Service container.

```
$> docker-compose --file docker-compose-run-orchestrator-service.yml up -d
```

- d. Start Data Service container.

```
$> docker-compose --file docker-compose-run-dataservice.yml up -d
```

- e. Start Archival Service container.

```
$> docker-compose --file docker-compose-run-archival-service.yml up -d
```

- f. Start Migration Service container.

```
$> docker-compose --file docker-compose-run-migration-service.yml up -d
```

- g. Start jeopardy Service container.

```
$> docker-compose --file docker-compose-run-jeopardy.yml up -d
```

- h. Start the omsui Service container.

```
$> docker-compose --file docker-compose-run-omsui.yml up -d
```

- i. Start the Broker Service.

```
$> docker-compose --file docker-compose-run-broker-service.yml up -d
```

- j. Start tmf-om adapter container. \$> docker-compose --file docker-compose-run-tmf-adapter-service.yml up.

- k. Run `$] docker ps -a` to check the containers that are started.

## Extend Docker-Compose Files

You can extend the Docker-Compose files provided as part of the TIBCO Order Management installation.

This is mainly done to handle different environments. For example, this is done in case you required a separate parameters for the containers based on your environment, such as a testing or production environment.

The suggested way to do this is to have multiple Docker compose files for each environment. For more information, see the [Docker documentation](#) on Multiple Compose Files.

## Modifying a Container Time-Zone

The default time-zone for any Docker container is UTC. In the case where you want the Docker container's time-zone to be in sync with the host machine's time-zone, you can apply these changes either in the Docker file or in the Docker-Compose YAML file.

Docker containers always use the system clock of the host machine but it sets its time-zone as UTC.

The following steps are an example of changing the time zone for an Order Management Server container.

### Procedure

1. You can modify a container's time zone with either of the following two ways:
  - This approach can be applied when you have not created any images. Open the `$OM_HOME/docker/orchestrator-service-context/6.1.0/Dockerfile` in a suitable editor and modify the file as shown:

```
FROM tibco/base:1.0
COPY orchestrator $OM_HOME/orchestrator
ENV TZ=Asia/Kolkata
RUN ln -snf /user/share/zoneinfo/$TZ etc/localtime "echo $TZ >
/etc/timezone
RUN chmod 777 $OM_HOME/orchestrator/standalone/bin/* \
 && chmod -R a+w $OM_HOME/orchestrator/standalone/logs \
 && chmod -R a+w $OM_HOME/orchestrator/standalone/config
USER root
ENTRYPOINT ["sh", "-c", "$OM_
HOME/orchestrator/standalone/bin/start.sh -
XX:MinRAMPercentage=$min_ram_percentage -
XX:MaxRAMPercentage=$max_ram_percentage --run=FG"]

EXPOSE 9093
```

In this example, the following has been modified:

```
ENV TZ=Asia/Kolkata
```

```
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ
/etc/timezone
```

Here you have to change the value of the TZ variable as per your time zone (in the example, the time zone is Asia/Kolkata).

- This approach can be applied if your images are already created and now you want to change the container time zone at runtime. Open `$OM_HOME/docker/docker-compose-run-orchestrator-service.yml` in a suitable editor and modify the file as shown:

```
version: "3"

services:
 tibco-orchestrator:
 image: "tibco/orchestrator:${OM_VERSION_TAG}"
 environment:
 min_ram_percentage: ${min_ram_percentage}
 max_ram_percentage: ${max_ram_percentage}
 ports:
 - "${HOST_ORCHESTRATOR_SERVICE_PORT}:9093"
 volumes:
 - "${HOST_LOG_ROOT_LOCATION_DIR_PATH}:/home/tibuser/tibco/
om/6.1/orchestrator/standalone/logs"
 deploy:
 resources:
 limits:
 cpus: '4'
 memory: 4G
 reservations:
 cpus: '0.2'
 memory: 512M
 environment:
 - "TZ=Asia/Kolkata"
 command: sh -c "ln -snf /usr/share/zoneinfo/$TZ
/etc/localtime && echo $TZ > /etc/timezone"
```

In this example, the following has been modified:

```
environment:
- "TZ=Asia/Kolkata"
```

```
command: >
 sh -c "ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo
$TZ > /etc/timezone"
```

Here you have to change the value of the TZ variable as per your time zone (in the example, the time zone is Asia/Kolkata).

## Reading Container Logs

When all the desired containers are up and running, it is best practice to check the logs for all the running services.

Logs for all the started and exited containers are available at the path you have mentioned for the LOG\_ROOT\_LOCATION\_DIR\_PATH variable in the .env file. So all the logs are preserved on your host machine.

## Troubleshooting Error from Building Docker Images

Troubleshoot an error when building Docker images with the following steps.

Complete the following step if the following error occurs when building Docker images:  
rm: cannot remove '/home/tibuser/tibco/om/6.1/configurator/standalone/config/backup':  
Directory not empty

### Procedure

1. Run the following command on the host machine:

```
$] docker info | grep 'Storage Driver' | awk -F':' '{print $2}'
overlay
$]
```

2. If the output is overlay, then apply the following workaround:
  - a. Stop the Docker engine.

- b. Changed DOCKER\_OPTS to set storage-driver value to device mapper, edit /etc/docker/daemon.json, and add "storage-driver" : "devicemapper" at the end of existing keys.
- c. Start the Docker engine.



**Note:** You can lose the existing Docker images due to the above change.

- d. Verify the fix by running the following command:

```
$] docker info | grep 'Storage Driver' | awk -F':' '{print $2}'
devicemapper
$]
```

## Order Sequencing

By default Order Sequencing feature is disabled. When Order Sequencing is disabled, all the incoming orders are processed in parallel. After the Order Sequencing is enabled, only a single order is processed at a time and any other incoming order by the same customer is stored in the ORDER\_IN\_SEQUENCE database table till the previous one is processed.

The order of a customer is processed in the sequence of order submission. Each order request has an element or a tag (called as custom property) in the request body, which is common into all the requests for a customer. Orders from one customer are processed in the sequence of order submission.

The following configuration properties are related to Order Sequencing:

Configuration Variable Name	Configuration Property Value	Description
Custom property JsonPath expression for order sequencing	com.tibco.fom.orch.sequencing.customerJsonPath	Custom property JsonPath for order sequencing



Configuration Variable Name	Configuration Property Value	Description
		that points to a unique customer identifier
Flag to enable or disable the Order Sequencing	<code>com.tibco.fom.orch.enableOrderSequencing</code>	Options to enable (for all or with a udf) or disable order sequencing

## Enabling or Disabling Order Sequencing

### Procedure

1. Define an JsonPath in `com.tibco.fom.orch.sequencing.customerJsonPath`, which points to a unique customer identifier in the order request.
2. Set the below are the enum values for `com.tibco.fom.orch.enableOrderSequencing`:
  - `Disable`: To disable order sequencing for all the orders.
  - `EnableForAll`: To enable order sequencing for all the orders.
  - `EnableWithUdf`: For the selected user-defined fields for which you want to enable order sequencing.

## Bulk Order Actions

Operations on an order are performed depending on the requirement. Performing the same action on individual orders are difficult and time-consuming. You can apply actions to the group of orders simultaneously using Bulk Order action.

The following operations can be performed on the group of orders:

- SUSPEND
- RESUME
- WITHDRAW
- CANCEL

These operations are exposed by the Order Management Service.

## Bulk Actions

The bulk order actions let administrators to cancel, suspend, resume, or withdraw a group of orders in a single invocation of a web service. This is useful:

- To perform a specific action on all orders in a particular region.
- To prevent repetitive intervention to perform similar actions.

The bulk order actions are based on the existing Order Management Server order service. This operation is called **BulkAction**.

The existing Order Management Server order service is modified to include a new operation. You can use this operation to specify the type of action to be performed along with the group of orders on which the action must be performed.

You can monitor the request status through:

- Event log - contains information about the status of the request.
- Order lists - show the change in the order status when refreshed.
- REST call - bulk order action can be made through BulkAction request.
- In progress jobs - contains bulk action performed tasks where each job contains job id, created date, action requested, total orders and processed orders along with the order IDs

All the errors that occur during this process are logged and handled individually.

## WSDL Location

This is the default location where all the WSDL files are copied after the installation.

- \$OM\_HOME/schemas/wSDL/orchestrator/OrderService.wSDL
- \$OM\_HOME/schemas/wSDL/orchestrator/OrderServiceJms.wSDL
- \$OM\_HOME/schemas/wSDL/aopd/AOPDService.wSDL

## Error Codes

The following table lists the error codes:

Error Code	Description
TIBCO-AFF-OMS-100046: INVALID_ACTION	Web service fault code for invalid values of action.
TIBCO-AFF-OMS-100047: NO ORDERS FOUND	Web service fault code when neither order id nor order reference is present in the request.
TIBCO-AFF-OMS-100048: BOTH ORDERID AND ORDERREF FOUND	Web service fault code when both order id and order reference are present in the request.
TIBCO-AFF-OMS-100020: ORDER {ORDERREF} NOT FOUND / ORDER {ORDERID} NOT FOUND	This exception is logged if an order to be canceled or withdrawn is not present in the Order Management Server component.

## Invoke Bulk Order Operation

The **BulkAction** bulk order operation requires the following input parameters to perform the selected action on all the orders contained in the request:

- Action type
- List of order IDs or order refs

The **BulkAction** order operation is an asynchronous operation and the consumer of the operation receives an acknowledgement immediately after the submission of the request. This acknowledgement is not an indication that the process is complete. This indicates that

the request is under process by the Order Management Server component. The operation can be invoked by a user with ADMIN role only.

## Tracking the Request Status

The request status for the invoked bulk order action can be tracked using:

- TIBCO Order Management UI (Dashboard, Order Screen and Activity logs)
- Logs in the Order Management Server and Orchestrator components

## Logging

TIBCO Order Management provides detailed logging and auditing capabilities to identify the system errors and key errors that can be gracefully handled by the calling system.

For bulk order actions, the logging is done using the AFFLogger APIs and the log file (`orchestrator.log`) is created in the corresponding location based on the configured Appender. The log location is `$OM_HOME/roles/orchestrator/standalone/logs`. The incoming bulk order request is validated and an INFO level log is generated. The log contains the action to be performed along with the number of orders in the request.

For all bulk order actions, if a particular order is not found in the Order Management Server component, an 'ERROR' level log is generated indicating that the order was not found.

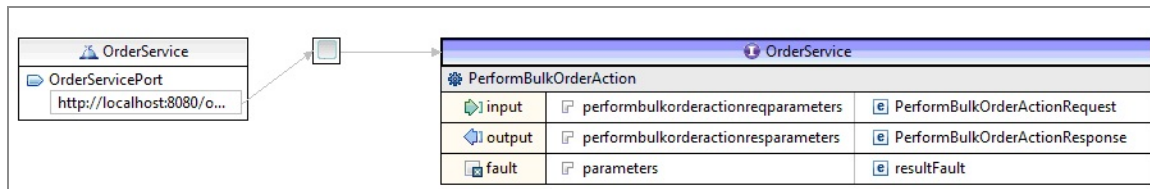
## Schema

A schema is an organization or structure for the `PerformBulkOrderAction` bulk order actions web service.

## Bulk Order Schema

The following figure depicts the action for a bulk order added to the Order Management Server order service.

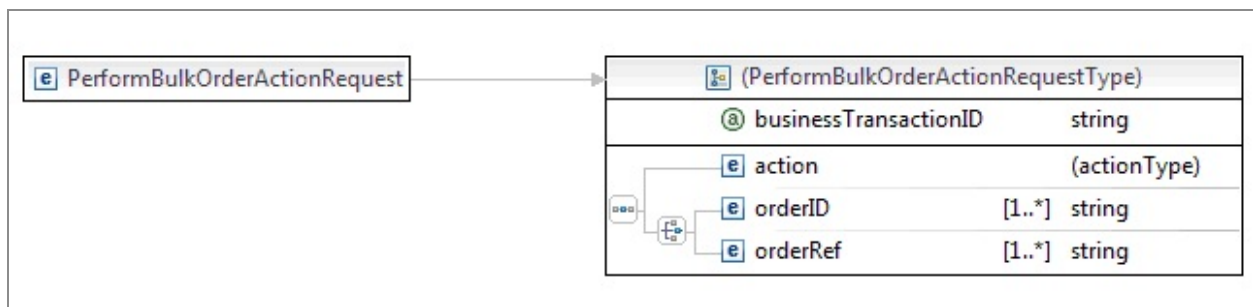
Action for a bulk order added to the service of the Order Management Server.



## Bulk Orders Operation Request Schema

The following figure depicts the bulk operation request schema.

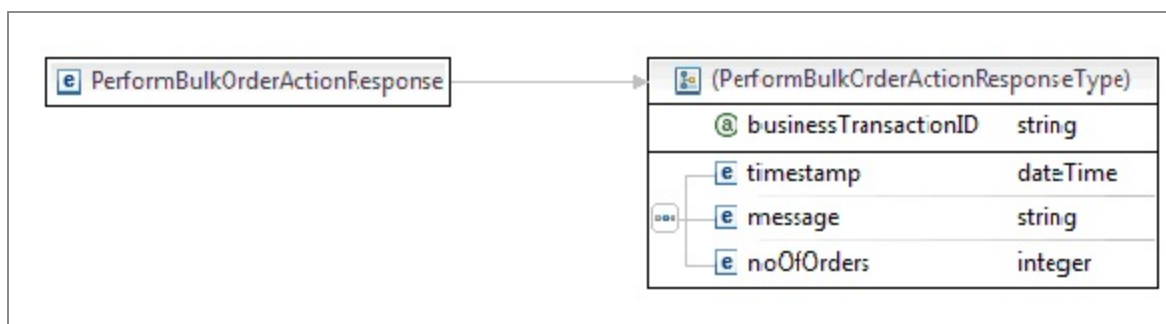
Request schema for bulk operation



## Bulk Orders Operation Response Schema

The following figure depicts the bulk operation response schema.

Response schema for bulk operation



## Sample Request

The sample request applicable to the bulk operation is as follows:

```
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-
envelope" xmlns:ord="http://www.tibco.com/aff/orderservice">
 <soapenv:Header/>
 <soapenv:Body>
 <ord:PerformBulkOrderActionRequest
businessTransactionID="bTranID">
 <ord:action>SUSPEND</ord:action>
 <ord:orderID>74</ord:orderID>
 <ord:orderID>56</ord:orderID>
 <ord:orderID>26</ord:orderID>
 <ord:orderID>30</ord:orderID>
 <ord:orderID>37</ord:orderID>
 <ord:orderID>88</ord:orderID>
 <ord:orderID>57</ord:orderID>
 <ord:orderID>27</ord:orderID>
 <ord:orderID>67</ord:orderID>
 <ord:orderID>35</ord:orderID>
 </ord:PerformBulkOrderActionRequest>
 </soapenv:Body>
</soapenv:Envelope>
```

## Sample Response

The sample response that is applicable to the bulk operation is as follows:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
 <soap:Body>
 <ns3:PerformBulkOrderActionResponse
xmlns="http://www.tibco.com/aff/order"
xmlns:ns2="http://www.tibco.com/aff/commontypes"
xmlns:ns3="http://www.tibco.com/aff/orderservice"
xmlns:ns4="http://www.tibco.com/aff/orderservice/result"
xmlns:ns5="http://www.tibco.com/aff/plan"
xmlns:ns6="http://www.tibco.com/aff/planfragments">
 <ns3:timestamp>2012-08-01T15:36:54.166+05:30</ns3:timestamp>
 <ns3:message>Request Submitted Successfully</ns3:message>
 <ns3:noOfOrders>10</ns3:noOfOrders>
 </ns3:PerformBulkOrderActionResponse>
```

```
</soap:Body>
</soap:Envelope>
```

## Performing Bulk Actions On error Plans Items


You can select a group of plan items on the error state and apply a bulk action on them simultaneously. The operations that can be performed are as follows:

- Retry
- Resume
- Complete

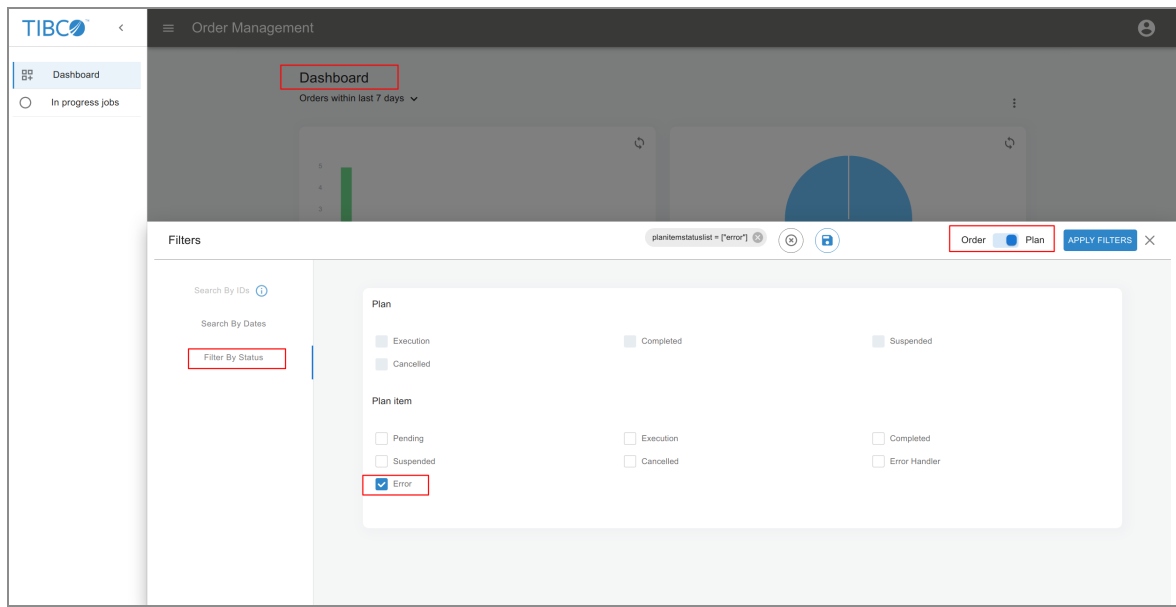


**Note:** This functionality is applicable only for handler type internal error handlers.

### Procedure

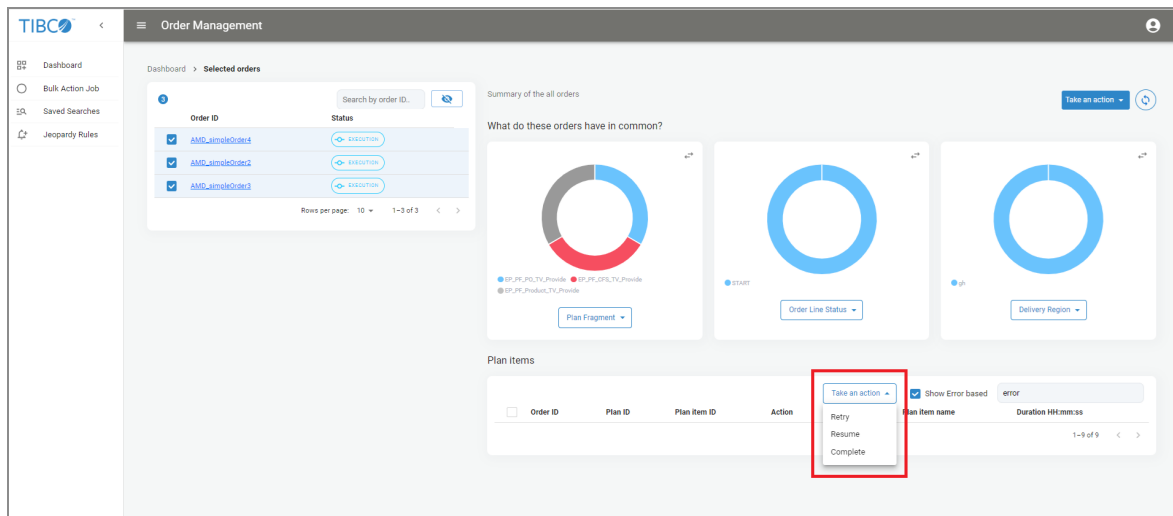
1. On Order Management System UI, filter error-based plans in **Dashboard > Search Orders >**  on the top-right corner of the **Find orders** table.

Find the orders table's filter drawer



2. To filter out the plans in the Find order table, switch the top-right available toggle button to 'Plan' and from the left section of the Filter drawer, select the **Filter by status** option.
3. Once the **Filter by status** section is visible, select the **Error** status checkbox under the **Plan items** section, and click the 'Apply Filters' button to get orders whose plan items are in the error state.
4. Select more than one order in the **Find orders** table, which comes after filtering. Selected orders are now added in the worktray. In the worktray, click the **Continue** button to move towards the bulk action screen with the selected orders. At the bottom of the page, the **Plan items** section must be visible.
5. Select the checkbox named **Show error based** to get the plan items, which are in error. Once the checkbox is selected, a **Take an action** dropdown populates next to it. From the **Take an action** dropdown, select the action that needs to be taken.





After the request is submitted, the list is cleared and closed.

## Multitenancy

The term multitenancy indicates an architecture in which a single running instance of an application simultaneously serves multiple clients or "tenants".

Isolating information, such as data and customizations, about the tenants is a particular challenge in these systems. This includes the data owned by each tenant. A single instance of the application can now support multiple tenants. Order Management Server REST API like catalog services, Automated Order Plan Development, data services, process component, and TM Forum adapter service can talk to the same TIBCO Order Management instance for processing and viewing the orders based on the tenant context. A default tenant "TIBCO" is supported by TIBCO Order Management without any configuration changes.

Multitenancy for TIBCO Order Management can be configured through the following steps.


1. [Creating and Configuring a Tenant](#)
2. [Authorizing a Tenant](#)

## Creating and Configuring a Tenant

### Procedure

1. Register a tenant by using the POST method `</v1/tenant>` and create a user with that tenant by using the `</v1/user>` API from the Authorization service. For more information, see [Create User](#).
2. Log in to the Configurator UI with the newly created tenant.
3. In the **Tenant Replication** window, enter the **Source TenantID** and click **REPLICATE**.

Here **Source TenantID** is a tenant ID that exists and you want to copy its properties to the current tenant.

 **Note:** When you log in with the default tenant or any other tenant with no data on the database, the data seeding option is enabled. When you log in with a non-default tenant and the database is not empty, the tenant replicate option is visible.

Alternatively, you can use the `http://<host_address>:<port_address>/v1/configuration/replicateTenantProperties` endpoint from the configuration service to replicate tenant properties by using the REST service.

4. Create an entry for the new tenant in the `order_lock` table in the Orders database by running the following query from the database client: `SELECT oms_seed_orderlock (10, 'new_tenant_id')`

This function accepts integers and `tenantId`. The integer value can be 7 to 12. The `tenantId` is the one that you want to create in Order Management.

## Authorizing a Tenant

Token-based authentication is used to authorize a tenant.

### Procedure

1. To authorize a tenant, generate token-based authentication. See [Generating an authorization token](#).
2. Pass the token-based authentication in the request header.

# Managing Health Check Endpoint

The TIBCO Order Management supports the health check endpoint to check the overall health status of the application resources like the EMS, Database, and Diskspace. You can check the health status of any service by putting the respective host and port number of that service in the following format.

You can find the number of database connection objects being used from the pool. You can set the `getTotalDsConnection` flag as true to enable the enhanced health check where you can see the number of database connections in the output.

For Authorization service, the `getTotalDsConnection` flag can be set in the `$OM_HOME/roles/authorization-service/standalone/config/application.properties` file.

For all other services, the `getTotalDsConnection` flag can be set in the `$OM_HOME/seed-data/app-properties/ConfigValues_Common.json` file.

The following list shows the health check endpoints:

- `http://<host>:<port>/management/health/readiness`
- `http://<host>:<port>/management/health/liveness`

Example of health check response:

```
{
 "status": "UP",
 "details": {
 "db": {
 "status": "UP",
 "details": {
 "name": "Configurator",
 "status": "RUNNING",
 "database": "PostgreSQL"
 }
 },
 "diskSpace": {
 "status": "UP",
 "details": {
 "total": 254720077824,
 "free": 93111214080,
 "threshold": 10485760
 }
 }
 }
}
```

## Implementation of LDAP

LDAP is a protocol through which Directory Service is connected. In Directory Service, the user's information is stored in a hierarchical structure.

The following properties are added in the `$OM_HOME/roles/authorization-service/standalone/config/application.properties` file to configure Directory Service:

- `directoryServiceDomainName=test`
- `directoryServiceRootDistinguishedName=DC=testad,DC=com`

- `ldapURLForDirectoryService=ldap://localhost:389`

## Authentication and Authorization

In TIBCO Order Management, support for authentication and authorization of all the available microservices is added. Authentication is used to authenticate someone's identity, whereas authorization is a way to permit someone to access a particular resource.

### Authentication Factors

Based on the security levels and the type of security that the application requires, there are different types of authentication factors. TIBCO Order Management supports Single-Factor authentication. This authentication mechanism requires users to provide a user name and password to access the system.

### Authorization Technique

The role-based access control technique is used to give users access to the TIBCO Order Management resources.

### Authorization Service

Authorization Service is a microservice available as part of TIBCO Order Management 6.1.0. This microservice has the following key features:

- This service generates JWT tokens based on OAuth2 specifications.
- `Grant_Type` used is `password`
- User credentials are entered in the request body when generating the OAuth2 token.

This service accepts encrypted passwords.

Once a user generates the token, it has to be entered as part of the header in each request (SOAP or REST). This token is used to verify the user's identity and authority.

### Resource Server


As per OAuth2 specification, a resource server is a server that hosts the protected resources and can accept and respond to protected resource requests by using access tokens.

Each TIBCO Order Management microservice is embedded with a resource-server library with the following capabilities:

- Verify token validity by using the same signing that was used to sign the token when it was generated by Authorization Service.

- Check token expiry.
- Extract claims from the token and set TenantId and authorities for the user.
- As part of the microservices configuration in TIBCO Order Management, each API exposed for the user has been protected with configurable role restrictions.

Example: The Orchestrator microservice uses the `operation.roles.submitOrder` property and `ROLE_ADMIN` property as the default values. It means that the users with a role as `ROLE_ADMIN` can access the submit-order API.

 **Note:** The role-based access is a fully configurable feature and can be modified.

## Generating User Token

The following methods describe the steps to generate a token if the user's information is stored with Authorization Service, in an external directory, or using a third-party external service:

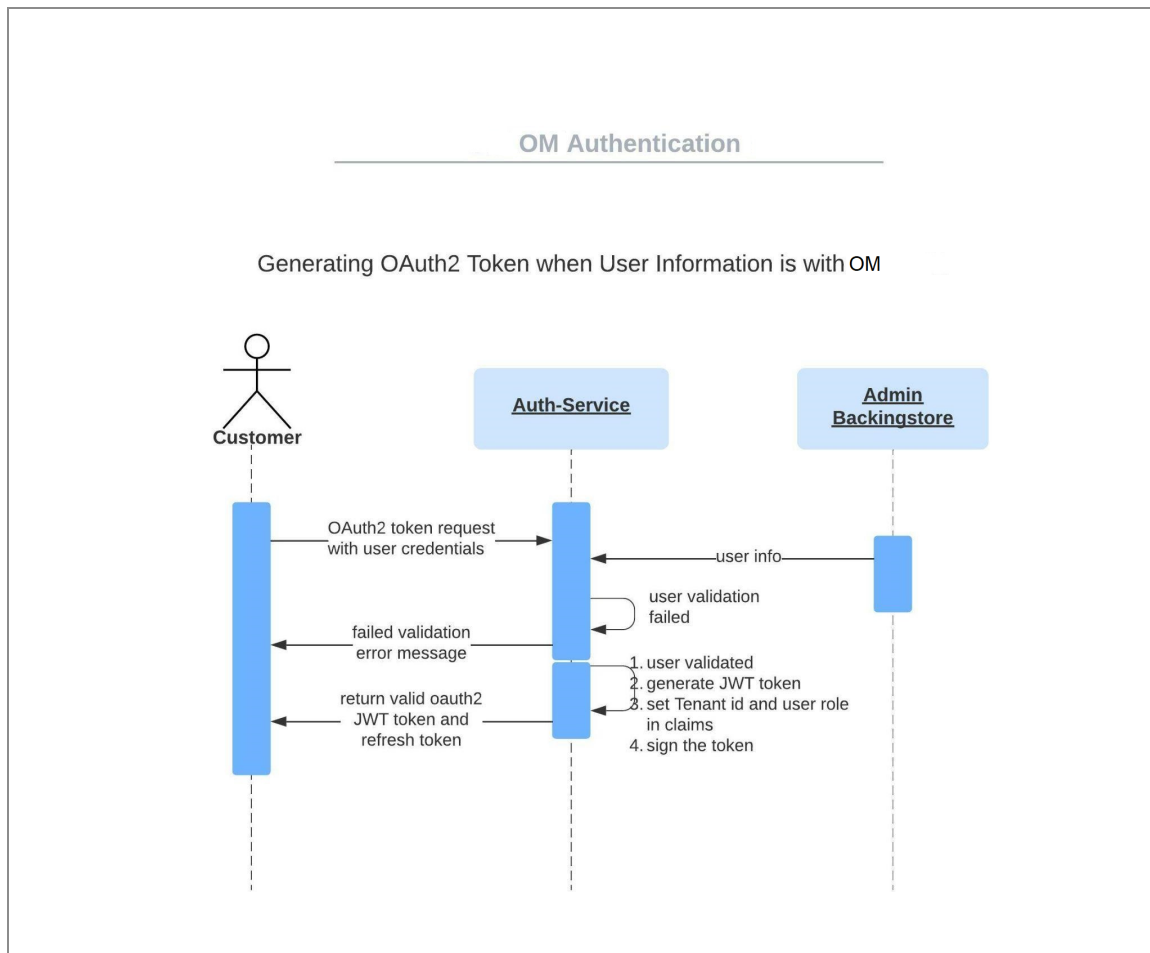
- [Information stored with Authorization Service](#)
- [Information stored in External Directory](#)
- [Using a third-party external service](#)

## Information stored with Authorization Service

Customers can choose to store Order Management user information in the Admin data store created and managed by Order Management.

The user credentials are encrypted and stored in the backing datastore.

The following sequence diagram illustrates token generation when the user's information is available in the Admin datastore:

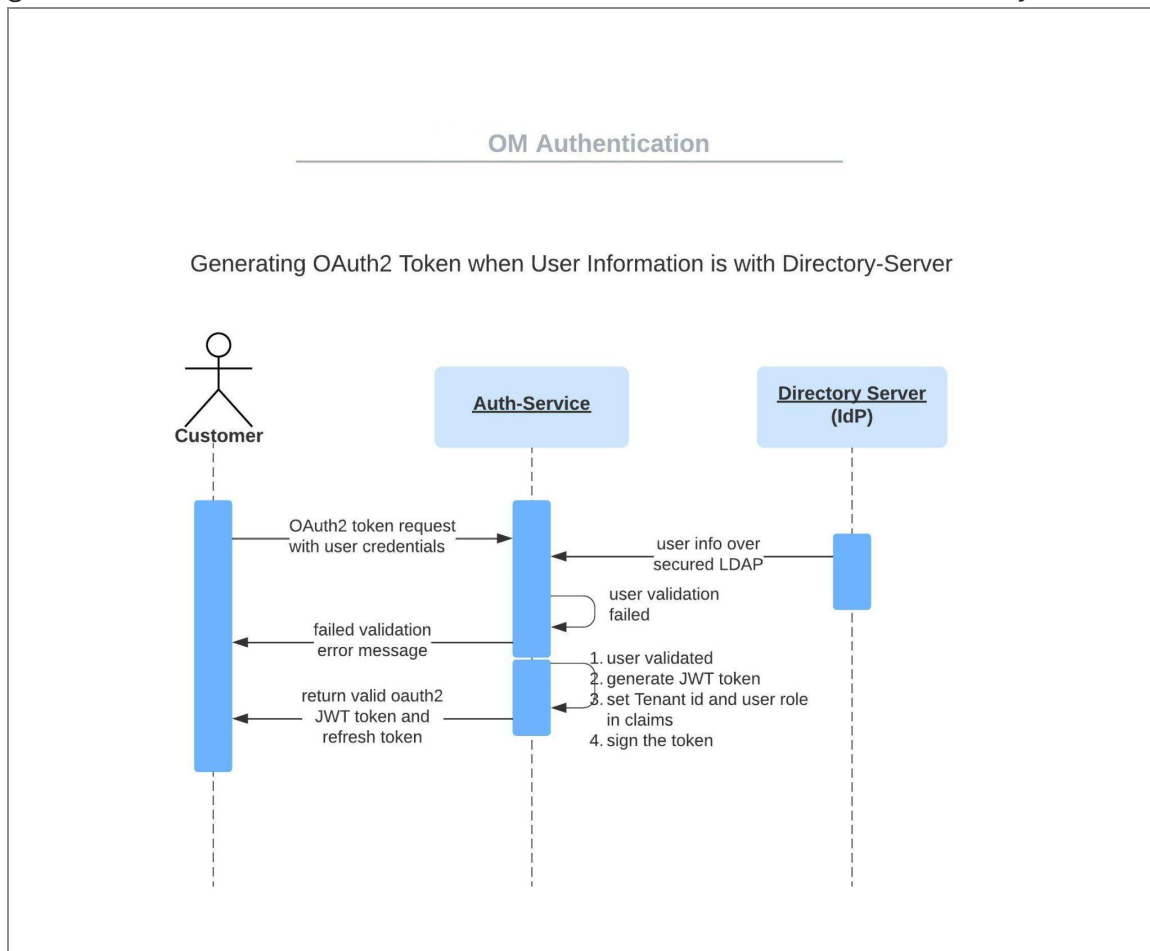


## Information stored in External Directory

Customers can choose to use the preexisting Directory Service(DS) which stores all the user information rather than replicating it in the Admin datastore.

Authorization Service can communicate with any external DS over LDAP or secured LDAP. Here, Directory Service acts as an IdP. The following sequence diagram illustrates token

generation when the user's information is available in an external Directory Service:



## Using a third-party external service

Customers might have an existing token generation mechanism (or service) and they might prefer using it for TIBCO Order Management 6.1.0

The Resource-Server library embedded in every microservice of Order Management expects OAuth2 JWT token must have a payload containing the following information:

- user\_name
- TENANTID
- an array of authorities containing roles of the user for whom this token was generated and these roles must have been configured for the API.
- The token must be signed with the secret string configured as a value of property `authentication.token.signing.key`





- Ds\_ActiveDirectory\_Relational
- Ds\_OpenLdap\_Relational
- Ds\_ActiveDirectory
- Ds\_OpenLdap

You need to map the Directory Service user with the Order Management user. You need to create the user in Order Management with at least `tenant Id`, valid `roles`, and `userName` as the mandatory fields. Leave the password field blank. The user name must match with the name provided in the ActiveDirectory.

You can use the **Create User** API ([http://<host\\_address>:<port\\_address>/v1/user](http://<host_address>:<port_address>/v1/user)) to create users who need to be mapped with the Directory Service users. This user roles must be same as the group roles present in Directory Service.

While creating an authorization token, the user name and password are validated by Directory Service. After successful validation, it checks the users table in the Order Management database, and an authorization token is generated as per the tenant Id that the user belongs.

Here, Directory Service is used for authentication and Order Management service is used for authorization.

## Types of retries

The following types of retries are supported in TIBCO Order Management:

- [Messaging level](#)
- [Feasibility retry](#)
- [Plan fragment based retry](#)
- [Plan Item failure retry configurations](#)
- [Web client retry](#)

### Messaging level

Messaging level retries are applicable wherever JMS is used. In the Archival, Data service, Orchestrator, Catalog, Jeopardy, and Migration services JMS is used. In case of any error, the retry mechanism is triggered as per the configuration.

## Feasibility retry

Feasibility retry takes place when `com.tibco.fom.orch.feasibilityRequired` and `retryFailedFeasibility` flags from the orchestrator application are set to true.

When an error occurred, it retries for a specified number of times (`feasibilityRetries`) in a specified interval (`feasibilityRetryInterval`) before the order goes in to the `preQualificationFailedReply`.

## Plan fragment based retry

Plan fragment based retry takes place when the `retryOverride` flag from the process component model is set to true.

When an error occurred, it retries for a specified number of times (`retryCount`) in a specified interval (`retryDelay`) before the plan item goes in to the `ERRORHANDLER/ERROR` state.

## Plan Item failure retry configurations

This is a backup of Plan fragment based retry. Plan Item failure retry configurations take place when the `retryOverride` flag from the plan fragment model is set to false.

When an error occurred, it retries for a specified number of times (`maxRetryCount`) in a specified interval (`retryInterval`) before the plan item goes in to the `ERRORHANDLER/ERROR` state. The system uses the default properties configured in the Orchestrator under the "Plan Item Failure Retry Configurations" category.

## Web client retry

TIBCO Order Management uses Spring's WebClient for inter-service communication on HTTP.

When an error occurred, it retries for a specified number of times (`*RetryCount`) in a specified interval (`*RedeliveryDelay`) before the HTTP communication failed.

# API Monitoring

Only some limited information is available through the existing monitoring systems. However with API monitoring, you can access more information.

Through the API monitoring system, you can access service level metrics, such as throughput, error and success rate, and response time for each API. You can also view the resource level metrics such as memory usage and CPU consumption. API monitoring is implemented for Catalog service, AOPD, Orchestrator, Data service, and Jeopardy.

You can use any of the following types of API monitoring tools:

- [JMX MBeans](#)
- [Prometheus](#)
- [Elasticsearch](#)
- [Dynatrace](#)



**Note:** For more information about these monitoring tools, you can visit the documentation pages of the respective tool.

## JMX MBeans

### Before you begin

Install visualVM tool and MBeans plug-in on your machine.

### Procedure

1. Set the following config values in the ConfigValues\_Common.json file:

Property Name	Value	Description
monitoringSystem	JMX	The monitoring system to view the application metrics.

Property Name	Value	Description
		(Default: Null)
		Set the value (such as JMX, prometheus, elastic, dynatrace) as per the required tool. You can also set multiple values here by comma separating them.

- Open the MBeans tab in the VisualVM tool to view the metrics.
- For the remote connection, you can add the following values in the `start.sh` script of OM services:
  - `Dcom.sun.management.jmxremote=true`
  - `Dcom.sun.management.jmxremote.port=port_no`
  - `Dcom.sun.management.jmxremote.authenticate=false`
  - `Dcom.sun.management.jmxremote.ssl=false`
  - `Djava.rmi.server.hostname=<hostname> or <host_ip>`
  - `Dcom.sun.management.jmxremote.rmi.port=port_no`

## Prometheus

### Procedure

- Set the following config values in the `ConfigValues_Common.json` file:

Property Name	Value	Description
monitoringSystem	prometheus	The monitoring system to view the application metrics.
		(Default: Null)
		Set the value (such as JMX, prometheus, elastic, dynatrace) as per the required tool.

Property Name	Value	Description
		You can also set multiple values here by comma separating them.

2. Add prometheus in the `management.endpoints.web.exposure.include` property value in the `ConfigValues_Common.json` file.
3. Open the OM management prometheus `http://<host>:<port>/management/prometheus` endpoint to view the metrics. You can also install the Prometheus application to view the prometheus metrics in a graphical representation.

## Elasticsearch

### Before you begin

Install the Kibana tool on your machine.

### Procedure

1. Set the following config values in the `ConfigValues_Common.json` file:

Property Name	Value	Description
monitoringSystem	elastic	<p>The monitoring system to view the application metrics.</p> <p><b>(Default: Null)</b></p> <p>Set the value (such as JMX, prometheus, elastic,</p>

Property Name	Value	Description
		dynatrace) as per the required tool. You can also set multiple values here by comma separating them.
management.metrics.export.elastic.enabled	true	Determines whether to enable the Elastic metrics or not
management.metrics.export.elastic.host	http://localhost:9200	Elastic search Url
management.metrics.export.elastic.index	micrometer-metrics	Management Metrics Elastic Index
management.metrics.export.elastic.step	1m	Time interval for sending metrics
management.metrics.export.elastic.userName	client's ElasticSearch User name (Default = "NULL")	ElasticSearch User name
management.metrics.export.elastic.password	client's ElasticSearch Password (Default = "NULL")	Encrypted ElasticSearch Password

Property Name	Value	Description
management.endpoints.jmx.exposure.include	*	Specifies the resource metric endpoints to expose, such as health and loggers.
management.endpoint.loggers.enabled	true	Specifies whether to enable the loggers endpoint. This is a Boolean property; it can be either true or false.
management.endpoints.jolokia.enabled	true	Specifies whether to enable the Jolokia endpoint. This is a Boolean property; it can be either true or false.

2. Open the Kibana tool to view the metrics.

## Dynatrace

### Before you begin

Install the Dynatrace tool on your machine.

## Procedure

1. Set the following config values in the `ConfigValues_Common.json` file:

Property Name	Value	Description
monitoringSystem	dynatrace	<p>The monitoring system to view the application metrics.</p> <p><b>(Default: Null)</b></p> <p>Set the value (such as JMX, prometheus, elastic, dynatrace) as per the required tool. You can also set multiple values here by comma separating them.</p>
management.metrics.export.dynatrace.uri	The host on which the Dynatrace tool is installed	The host on which the Dynatrace tool is installed
management.metrics.export.dynatrace.api-token	Access token generated from the Dynatrace tool	Access token generated from the Dynatrace tool
management.metrics.export.dynatrace.device-id	The Id of the device on which the Dynatrace tool is	The Id of the device on which the Dynatrace tool is installed



Property Name	Value	Description
	installed	
management.metrics.export.dynatrace.step	1m	Time interval for sending metrics
management.metrics.export.dynatrace.enabled	true	Determines whether to enable the dynatrace metrics or not.  (Default: false)

2. Add "dynatrace" as value for Dynatrace in the `management.endpoints.web.exposure.include` property value in the `ConfigValues_Common.json` file.
3. Open the Dynatrace tool to view the metrics.

## Debugging tools for production

### Read BLOB data from Database

To read the binary large object (BLOB) data from the database, perform the following procedures:

#### Procedure

1. For PostgreSQL, run the following command:
 

```
select encode(<column name>, 'escape') from <table name>;
```
2. For Oracle, perform the following steps:
  - a) Run the upgrade script which creates a function used to read BLOB data.
  - b) Run the following command to read the BLOB data:
 

```
select blob_to_clob(<column name>) from<table name> where
```

```
filename='<file name>'
```

## GET REST APIs in Catalog Service and AOPD

Through these APIs, you can verify the models that you have published.

**GET APIs in Catalog:** Fetch data from the database (Relational)

**Endpoint:** `http://<host_address>:<port_address>/v1/<actionModel>/bulk` or `<planFragmentModel>/bulk` or `<productModel>/bulk`

**Method:** GET

Fill the **actionModelIdList**, **planFragmentModelIdList**, or **productModelIdList** fields and click **Execute**.

All the models are retrieved from the database (Redis or Relational) that are published.

**GET APIs in AOPD:** Fetch data from the in-memory cache.

**Endpoint:** `http://<host_address>:<port_address>/<actionModels>` or `<productModels>`

**Method:** GET

Fill the **actionModelIdList** or **productModelIdList** fields and click **Execute**.

All the models are retrieved from the in-memory cache and displayed.

**GET API in Orchestrator:** Fetch data from the in-memory cache.

**Endpoint:** `http://<host_address>:<port_address>/<planFragments>`

**Method:** GET

Fill the **planFragmentIdList** field and click **Execute**. All the models are retrieved from the in-memory cache and displayed.

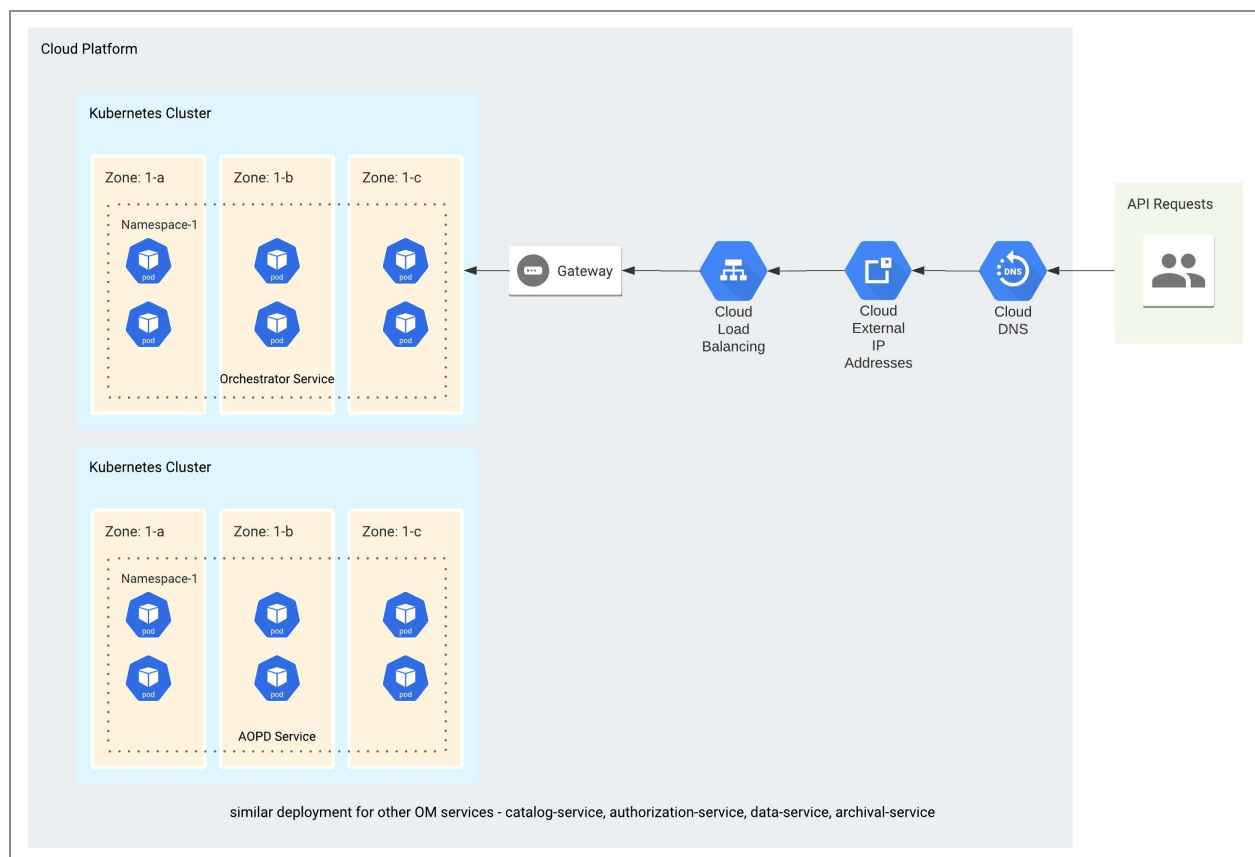
**i Note:** If the `enableProductModelGlobalCache`, `enableActionModelGlobalCache`, and `enablePlanFragmentGlobalCache` flag values are true, then only this GET API works.

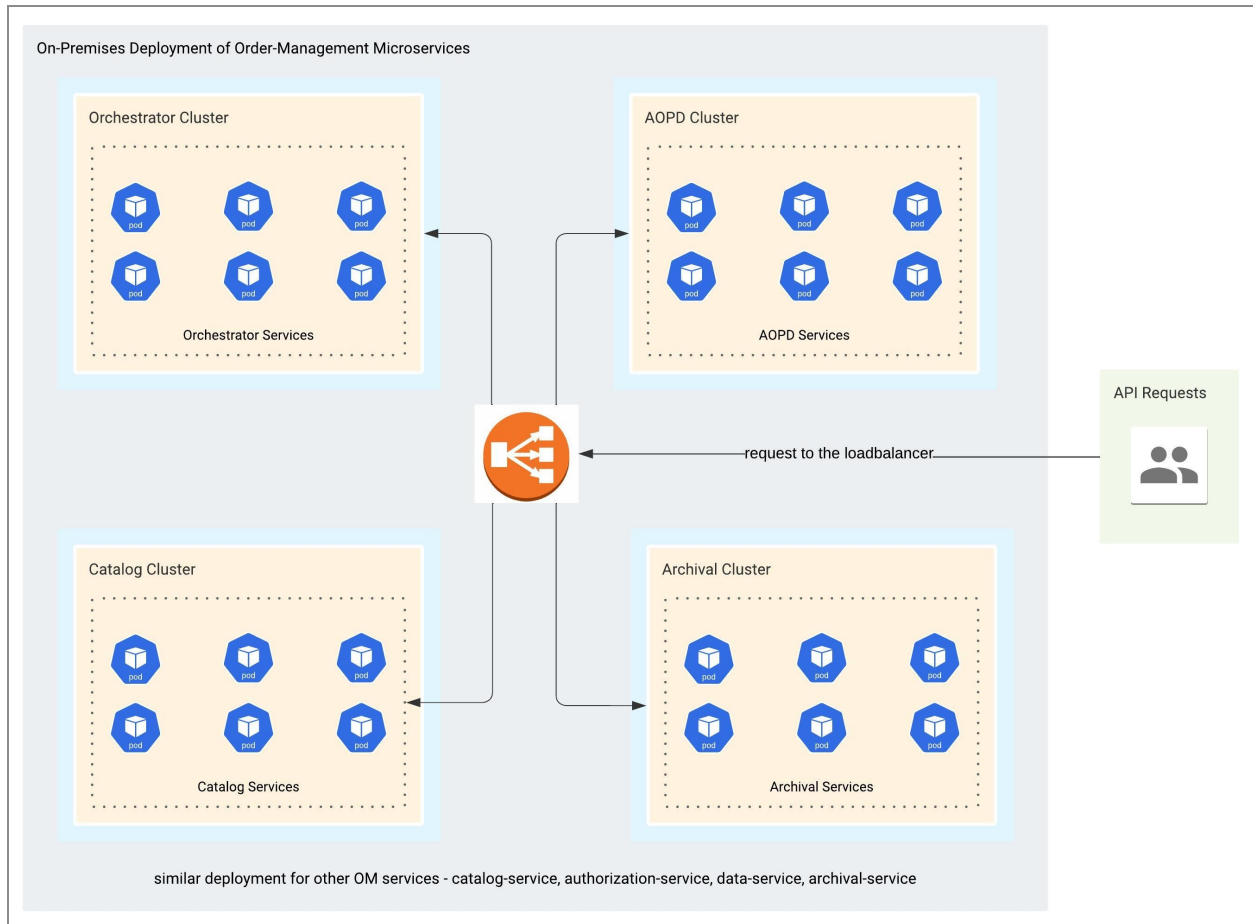
# Scaling of Order Management microservices

Previously, till the TIBCO® Order Management - Long Running 5.0.0 release, for each order management microservice node, member IDs were required to be registered in the domain member tables. The major drawback of this approach is that, whenever a new node is added to the existing cluster, you must restart the existing microservice nodes.

Now, you can scale any of the microservices as per the incoming load without restarting or configuring a separate new member. Any deployment topology can be used to and order management microservices can be started behind an external load balancer (when you choose to use SOAP over HTTP or RESTful interfaces). You can replicate microservice nodes at run time without the need to restart any of the existing microservices.

The following examples show the scaling of microservices:





## orderPriority

This section describes the orderPriority process.

orderPriority enables the user to set priority on submitted orders. This information is then used by the JMS broker to deliver the high priority orders to downstream components. orderPriority is also propagated to downstream process components. The priority value ranges from 0 to 9. The priority information or priority value to process any given order is set in the JMSHeader field of the JMS message that is sent to the Orchestrator Engine in Fulfillment Order Management.

The JMS broker then delivers the order based on a priority. The process of order prioritization works at a queue level and can be controlled by the JMS broker.

**i Note:** The orderPriority process or order prioritization cannot be controlled once the message is delivered to the orchestrator engine. The priority value can be changed before the JMS broker sends the order request to the engines.

## Order Schema Changes

You can use the order schema to submit the orderPriority information with the order. The orderPriority is at the orderHeader level and the same priority is applied to all the orderLines.

The schema snippet is as follows:

```
<xs:element name="orderPriority" minOccurs="0" default="4">
 <xs:simpleType>
 <xs:restriction base="xs:integer">
 <xs:minInclusive value="0" />
 <xs:maxInclusive value="9" />
 </xs:restriction>
 </xs:simpleType>
</xs:element>
```

The orderPriority can take values ranging from 0 to 9 to make them consistent and map with JMSPriority message header values.

**i Note:** The default value of the orderPriority field is 4.

## Lower Priority Orders

When any order is processed based on the given priority, it results in a situation where a lower priority order may never be processed because of high priority orders.

The orders with a lower priority can be processed by a mechanism known as Flow Control.

You can use Enterprise messaging Service (EMS) to control the flow of messages to a destination. Each destination can specify a maximum target size for storing all the pending messages. When the target is reached, EMS blocks message producers when new messages

are sent. This effectively slows down message producers until the message consumers can receive the pending messages.

## Tuning Data Source

You can use data source tuning to boost the performance of the relational data source in the server application. Allocating and deallocating resources for data sources is not so easy in terms of time and system resources. During the application startup, you can create a pool of database connections in advance and make these connections available to the application.

For TIBCO Order Management, you can configure the following properties for data source tuning as per your requirements. For description and more information about the properties, see the [Tomcat JDBC Connection Pool](#) documentation.

Microservice	Property	Description
AOPD	catalogDsTestOnBorrow	Enables connection validation before being borrowed from the pool
	catalogDsValidationInterval	Data source validation interval in milliseconds
	catalogDsTestWhileIdle	Enables connection validation while idle in the connection pool
	catalogDsTimeBetweenEvictionRunsMillis	Data source eviction

Microservice	Property	Description
		interval in milliseconds
	catalogDsMinEvictableIdleTimeMillis	Minimum time in milliseconds an object must sit idle in the pool before it is eligible for eviction
	catalogDsNumTestsPerEvictionRun	Data source tests per eviction run
	catalogDsDefaultAutoCommit	Default auto-commit state of connections created by this pool
	catalogDsRollbackOnReturn	Enables rollback of any pending transaction when a connection is returned to the pool
	catalogDsCommitOnReturn	Enables commit of any pending transaction when a connection is returned to the

Microservice	Property	Description
Orchestrator		pool
	catalogDsCustomProperty	Database custom property
	catalogDsInitializeSize	Number of connections established when the connection pool starts
	catalogDsMaxIdle	Maximum number of connections to keep in the idle pool
	catalogDsMaxActive	Maximum number of active connections that can be allocated from this pool at the same time
	catalogDsMaxWait	Maximum time in milliseconds the pool will wait when there are no available connections
	catalogDsTestOnBorrow	Enables



Microservice	Property	Description
		connection validation before being borrowed from the pool
	catalogDsValidationInterval	Data source validation interval in milliseconds
	catalogDsTestWhileIdle	Enables connection validation while idle in the connection pool
	catalogDsTimeBetweenEvictionRunsMillis	Data source eviction interval in milliseconds
	catalogDsMinEvictableIdleTimeMillis	Minimum time in milliseconds an object must sit idle in the pool before it is eligible for eviction
	catalogDsNumTestsPerEvictionRun	Data source tests per eviction run
	catalogDsDefaultAutoCommit	Default auto-commit state

Microservice	Property	Description
Archival		of connections created by this pool
	catalogDsRollbackOnReturn	Enables rollback of any pending transaction when a connection is returned to the pool
	catalogDsCommitOnReturn	Enables commit of any pending transaction when a connection is returned to the pool
	catalogDsCustomProperty	Database custom property
	archivalHibernateShowSql	Enables Hibernate to show queries
	archivalDsInitialSize	Number of connections established when the connection pool starts

Microservice	Property	Description
	archivalDsMaxWait	Maximum time in milliseconds the pool waits for a connection to be returned before throwing an exception
	archivalDsMaxActive	Maximum number of active connections that can be allocated from this pool at the same time
	archivalDsMaxIdle	Maximum number of connections to keep in the idle pool
	archivalDsMinIdle	Minimum number of established connections to keep in the pool at all times
	archivalDsTestOnBorrow	Pooled data source test on borrow

Microservice	Property	Description
	archivalDsValidationInterval	Pooled data source validation interval
Catalog	catalogDsMaxIdle	Maximum number of connections to keep in the idle pool
	catalogDsMaxActive	Maximum number of active connections that can be allocated from this pool at the same time
	catalogDsMaxWait	Maximum time in milliseconds the pool will wait when there are no available connections
	catalogDsTestOnBorrow	Enables connection validation before being borrowed from the pool
	catalogDsValidationInterval	Data source validation

Microservice	Property	Description
		interval in milliseconds
	catalogDsTestWhileIdle	Enables connection validation while idle in the connection pool
	catalogDsTimeBetweenEvictionRunsMillis	Data source eviction interval in milliseconds
	catalogDsMinEvictableIdleTimeMillis	Minimum time in milliseconds an object must sit idle in the pool before it is eligible for eviction
	catalogDsNumTestsPerEvictionRun	Data source tests per eviction run
	catalogDsDefaultAutoCommit	Default auto-commit state of connections created by this pool
	catalogDsRollbackOnReturn	Enables rollback of any pending transaction

Microservice	Property	Description
		when a connection is returned to the pool
	catalogDsCommitOnReturn	Enables commit of any pending transaction when a connection is returned to the pool
	catalogDsCustomProperty	Database custom property

## Catalog Caching

Catalog caching improves the performance and scalability of applications by reducing the need to repeatedly fetch the same data from the database. You can configure the following properties for catalog caching according to your requirements:

Microservice	Property	Description
AOPD	maxNoProductcached	Maximum number of product catalogs stored in cache
	enableProductModelGlobalCache	Enables global caching for the product catalog
	productCacheExpiryPeriod	Product catalog cache expiry period in seconds

Microservice	Property	Description
	maxNoActioncached	Maximum number of action catalogs stored in cache
	enableActionModelGlobalCache	Enables global caching for the action catalog
	actionCacheExpiryPeriod	Action catalog cache expiry period in seconds
Orchestrator	enablePlanFragmentGlobalCache	Enables global caching for the plan-fragment catalog
	maxNoPlanFragmentcached	Maximum number of plan-fragment catalogs stored in cache
	planFragmentCacheExpiryPeriod	Plan-fragment catalog cache expiry period in seconds
	globalCacheCleanupTopicName	Global cache cleanup topic name
Common	enableProductScoringAndLedger	Enables product scoring. To enable model scoring and ledger, set the enableProductScoringAndLedger value to true for the product model in the ConfigValues_Common.xml file
	cmPluggableCache	Decides whether to use Redis or Relational in the application
	enablePriceScoringAndLedger	Enables price scoring. To enable model scoring and ledger, set the enablePriceScoringAndLedger value to true for the price model in the ConfigValues_Common.xml file.
	enableDiscountScoringAndLedger	Enables discount scoring. To enable model scoring and ledger, set the

Microservice	Property	Description
		enableDiscountScoringAndLedger value to true for the discount model in the ConfigValues_Common.xml file.
Catalog	globalCacheCleanNotificationTopic	The name of the global cache cleaning topic, used for sending notifications about the model that has been modified or purged.

## Integrate Inventory Information in AOPD Plan Generation

In the Automated Order Plan Development (AOPD), integrating inventory information into plan generation is a critical process. The following properties determine how inventory information is integrated into the plans.

Property	Description
mergeInventory	<p>Determines whether the functionality for merging inventory is activated during plan generation.</p> <p>Possible values: true or false</p> <p>Default: false</p>
inventoryUsername	Username used for authentication with the services of TIBCO Product Service and Inventory.
inventoryPassword	Password used for authentication with the services of TIBCO Product Service and Inventory.



Property	Description
inventoryBaseUrl	Base URL containing the host and port for accessing TIBCO Product Service and Inventory.

When submitting an order request, including `PartyId` is mandatory. This identifier can be provided as either `CustomerId` or `SubscriberId`. TIBCO Order Management evaluates the `mergeInventory` property to decide on the integration of inventory information into plan generation.

- When `mergeInventory` is true

Order management retrieves inventory information from Product Service and Inventory using the credentials and base URL specified by `inventoryUsername`, `inventoryPassword`, and `inventoryBaseUrl`. The system integrates the retrieved inventory data into the plan, customizing it according to the specifics of the order request.

- When `mergeInventory` is false

The inventory merging process is skipped. Plan generation continues without incorporating live inventory data, relying on the information available at the request time.

## Integrate TIBCO OPE with Order Submission Process

Activating the `isEnabledOfferValidation` property integrates TIBCO Offer and Price Engine with the order submission process. This ensures integration so that the order adheres to specified criteria before completion.

With `isEnabledOfferValidation` set to true, Offer and Price Engine initiates the validation process. The process examines the order ID to verify the validity of the offer associated with the order.

- Valid orders: Orders that meet the validation criteria are submitted.
- Invalid orders: For orders that do not meet the criteria, the system takes actions as defined by the validation failure handling protocol.

# Schema References

---

You can find JSON schema definitions on the following URL for all the respective services.

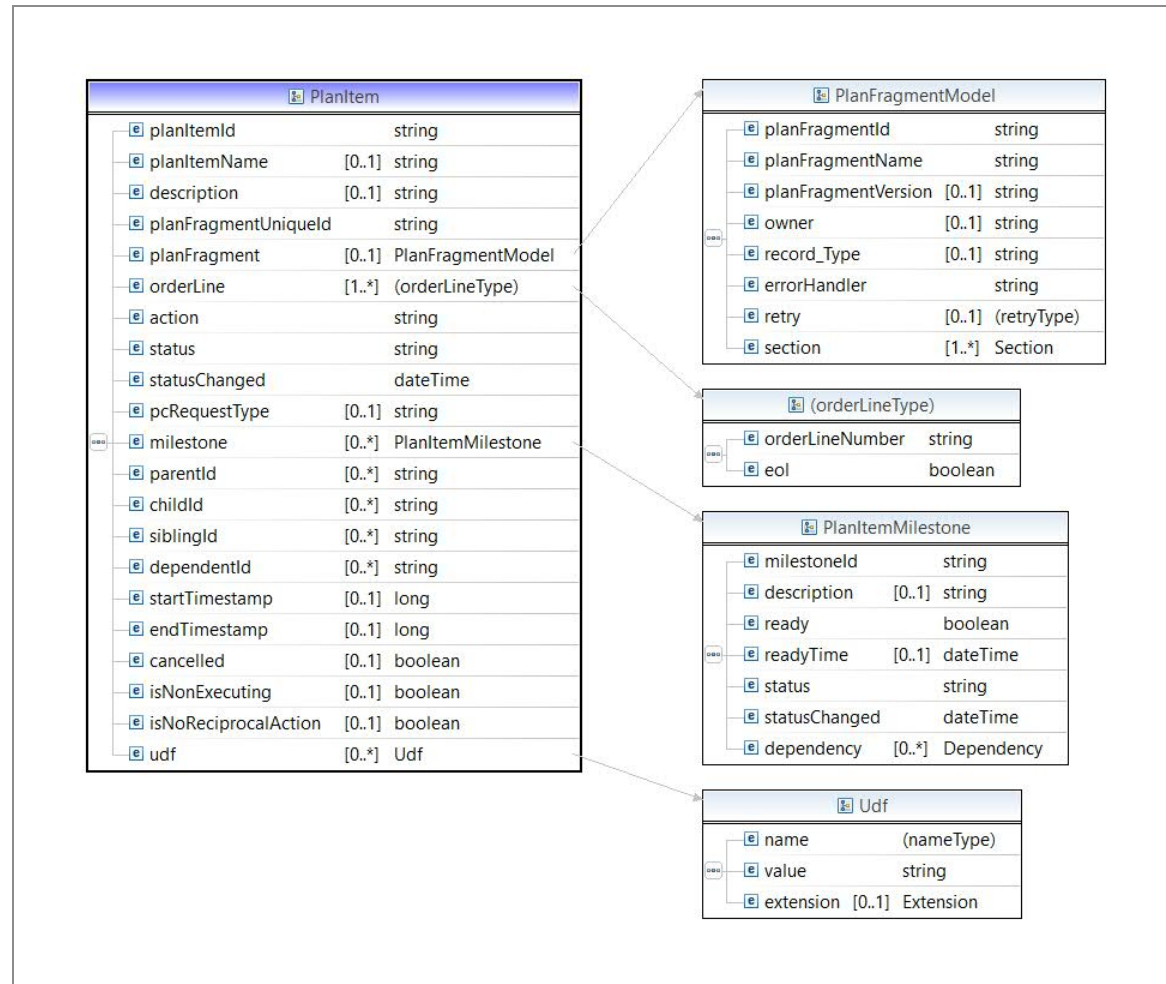
`http://<host>:<port>/v3/api-docs` (Example: If the Orchestrator is running on `localhost` and port `9093`(default), then the API doc is available on `http://localhost:9093/v3/api-docs`).

The following list represents the common schema definition present in Order Management services:

- [Plan Item](#)
- [Product Model](#)
- [Result Status](#)
- [Message](#)
- [Order Request](#)
- [Order Request Header](#)
- [Order Request Line](#)
- [Process Component Model](#)

# Plan Item

## Plan Item



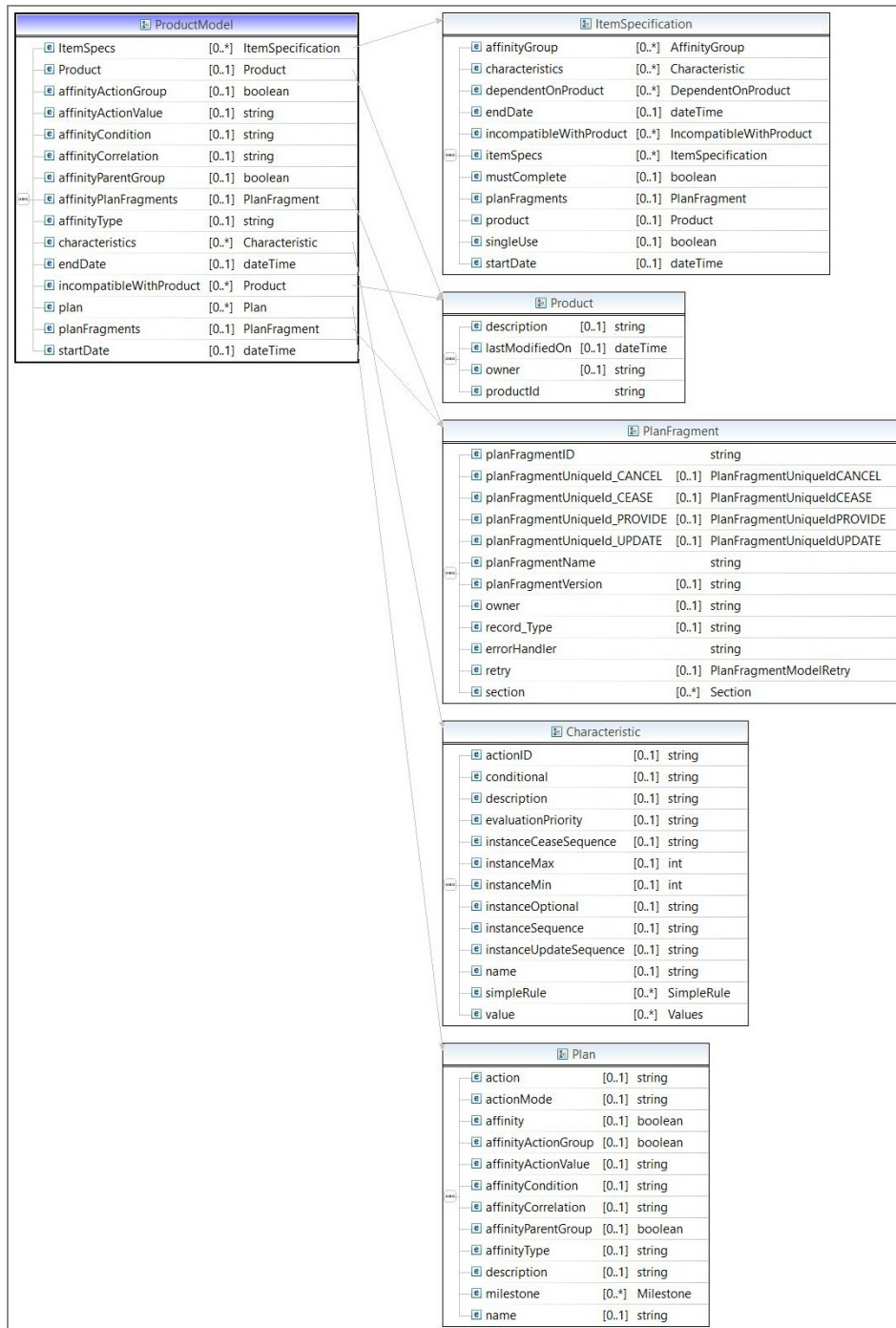
Element	Type	Cardinality	Description
planItem/planItemId	String	Required	A unique identifier for the plan item within the plan to be executed.
planItem/description	String	Optional	Description for the plan item to be executed.
planItem/processComponentID	String	Required	A unique identifier for

Element	Type	Cardinality	Description
			the Process Component to be executed.
planItem/processComponentName	String	Required	Process component name for the Process Component to be executed.
planItem/processComponentVersion	String	Optional	Process component version for the Process Component to be executed.
planItem/processComponentType	String	Optional	Process component type for the Process Component to be executed.
planItem/processComponentRecordType	String	Optional	Class of processComponentType.
planItem/orderLine	Type	1-M	Order line type for the plan item to be executed.
planItem/orderLine/orderLineNumber	String	Required	Order line number for the order line of the plan item to be executed.
planItem/orderLine/productID	String	Required	Product ID for the order line of the plan item to be executed.
planItem/orderLine/productVersion	String	Optional	Product version for the order line of the plan item to be executed.

Element	Type	Cardinality	Description
planItem/orderLine/action	String	Required	Order line action for the order line of the plan item to be executed.
planItem/orderLine/actionMode	String	Optional	Order line action mode for the order line of the plan item to be executed.
planItem/orderLine/quantity	Long	Required	Quantity for the order line of the plan item to be executed.
planItem/orderLine/uom	String	Required	Unit of measure for the order line of the plan item to be executed.
planItem/orderLine/subscriberID	String	Optional	Subscriber ID for the order line of the plan item to be executed.
planItem/orderLine/linkID	String	Optional	Link ID for the order line of the plan item to be executed.
planItem/orderLine/inventoryID	String	Optional	Inventory ID for the order line of the plan item to be executed.
planItem/orderLine/eol	Boolean	Required	End of line flag for the order line of the plan item to be executed. This indicates that this plan item is the final plan item for the order line.

Element	Type	Cardinality	Description
planItem/action	String	Required	Plan item action for the plan item to be executed.
planItem/actionMode	String	Optional	Plan item action mode for the plan item to be executed.

# Product Model



Element	Type	Cardinality	Description
ItemSpecs	Item Specification	Optional	Relationship information tag.
Product	Product	Mandatory	product information tag.
affinityActionGroup	Boolean	Optional	Valid for Conditional type only. A Boolean field containing the value true or false.
affinityActionValue	String	Optional	AffinityActionValue is considered for grouping when AffinityActionGroup is set to true. This is valid for Conditional type only.
affinityCondition	String	Optional	Valid for Conditional type only. A String field containing an XPATH expression that evaluates to true or false based on data is in the order.
affinityCorrelation	String	Optional	Valid for Conditional type only. The XPATH is evaluated on the Plan data and the



Element	Type	Cardinality	Description
			order data.
affinityParentGroup	Boolean	Optional	Valid for Conditional type only. A Boolean field containing the value true or false.
affinityPlanFragments	Plan Fragment	Optional	Affinity plan fragment.
affinityType	String	Optional	For a plan this is the type of affinity grouping required.
characteristics	Characteristic	Mandatory	Characteristic type.
endDate	Date Time	Optional	End Date for the record to be effective.
incompatibleWithProduct	Product	Optional	Incompatible relationship
plan	Plan	Optional	Planfragment relationship tag.
planFragments	Plan Fragment	Optional	It provides configuration information for a Process Component/Plan Fragment.
startDate	Date Time	Optional	start Date

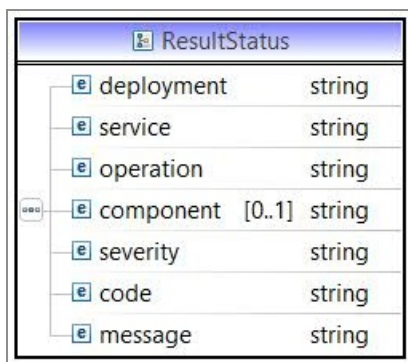
Element	Type	Cardinality	Description
affinityGroup	Affinity Group	Optional	Affinity Group
dependentOnProduct	Dependent On Product	Optional	Reverse relationship
mustComplete	Boolean	Optional	Must complete flag for provisioning
singleUse	Boolean	Optional	Single use flag for provisioning
description	String	Optional	Description of the product model
lastModifiedOn	String	Optional	Last modified date of the record
owner	String	Optional	owner
productId	String	Optional	Identifier of product record
planFragmentID	String	Optional	Identifier of planFragment record
planFragmentUniqueId_CANCEL	planFragmentUniqueIdCANCEL	Optional	Planfragment identifier for CANCEL action
planFragmentUniqueId_CEASE	planFragmentUniqueIdCEASE	Optional	Planfragment identifier for CEASE action
planFragmentUniqueId_PROVIDE	planFragmentUniqueIdPROVIDE	Optional	Planfragment identifier for PROVIDE action

Element	Type	Cardinality	Description
planFragmentUniqueld_UPDATE	planFragmentUniqueldUPDATE	Optional	Planfragment identifier for UPDATE action
planFragmentName	String	Optional	Name of process component
planFragmentVersion	String	Optional	Version of process component
record_Type	String	Optional	Record type
errorHandler	String	Optional	Error handler to use in case of failure
retry	PlanFragmentModelRetry	Optional	Retry type.
section	String	Optional	Product model section type.
actionID	String	Optional	Unique identifier of Action record
conditional	String	Optional	conditional
evaluationPriority	String	Optional	Evaluation Priority
name	String	Optional	Name of the product model.
action	String	Mandatory	Action for the plan Item. Valid values are: <ul style="list-style-type: none"> <li>• PROVIDE</li> <li>• UPDATE</li> </ul>

Element	Type	Cardinality	Description
			<ul style="list-style-type: none"> <li>• CEASE</li> <li>• CANCEL</li> </ul>
actionMode	String	Optional	Action mode. This is optional supplemental information to go with OrderLineAction. Valid value is: MOVE
affinity	Boolean	Optional	Affinity value true or false
milestone	Milestone	Optional	Milestone of the plan

## Result Status

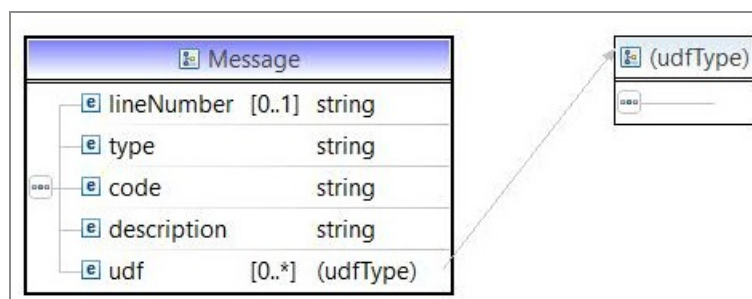
### Result Status



Element	Type	Cardinality	Description
deployment	String	Required	Engine deployment that returned this result.
service	String	Required	Service name that returned this result
operation	String	Required	Operation within the service that returned this result.
component	String	Optional	Component within the operation and service that returned this result.
severity	String	Required	Severity result. Valid values are: <ul style="list-style-type: none"> <li>1. S - Success</li> <li>2. W - Warning</li> <li>3. E - Error</li> </ul>
code	String	Required	Message code for this result.
message	String	Required	Message details for this result.

## Message

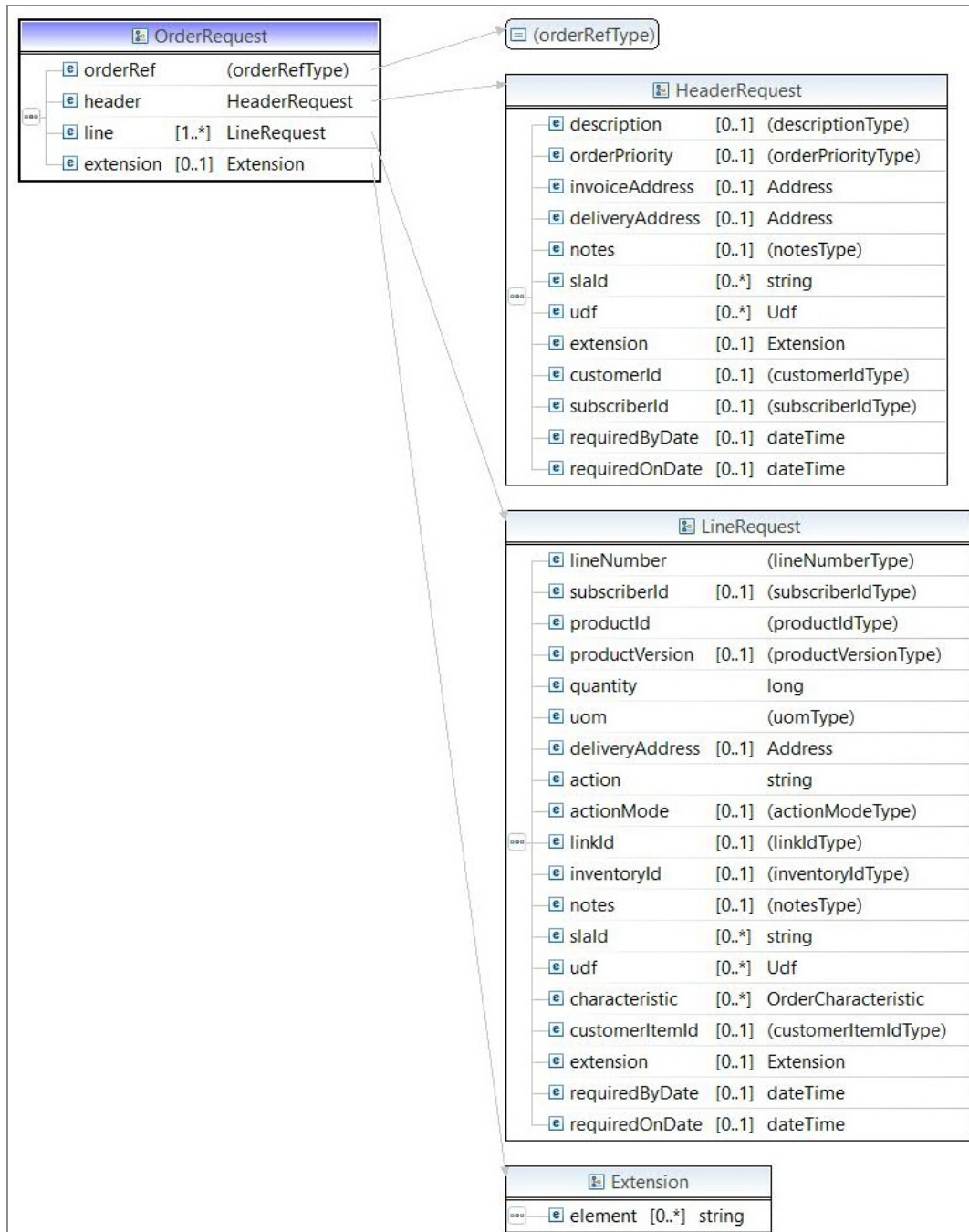
### Message



Element	Type	Cardinality	Description
lineNumber	String	Optional	Order line number that this message refers to.
type	String	Required	Message type. Valid values are: <ol style="list-style-type: none"><li>1. Information</li><li>2. Warning</li><li>3. Error</li></ol>
Code	String	Required	Message code for this message.
Description	String	Required	Message text for this message.
udf	Type	0-M	User defined field type.
udf/name	String	Required	User defined field name.
udf/value	String	Required	User defined field value.

# Order Request

## Order Request



Element	Type	Cardinality	Description
orderRef	String	Required	External unique identifier for an order.
header	Type	Required	Order request header type. Refer to the Order Request Header definition for details.
line	Type	1-M	Order request line type. Refer to the Order Request Line definition for details.
extension	Type	Optional	Extension attributes for user-defined fields.
extension/#any	Any	Required	Any data



# Order Request Header

## Order Request Header



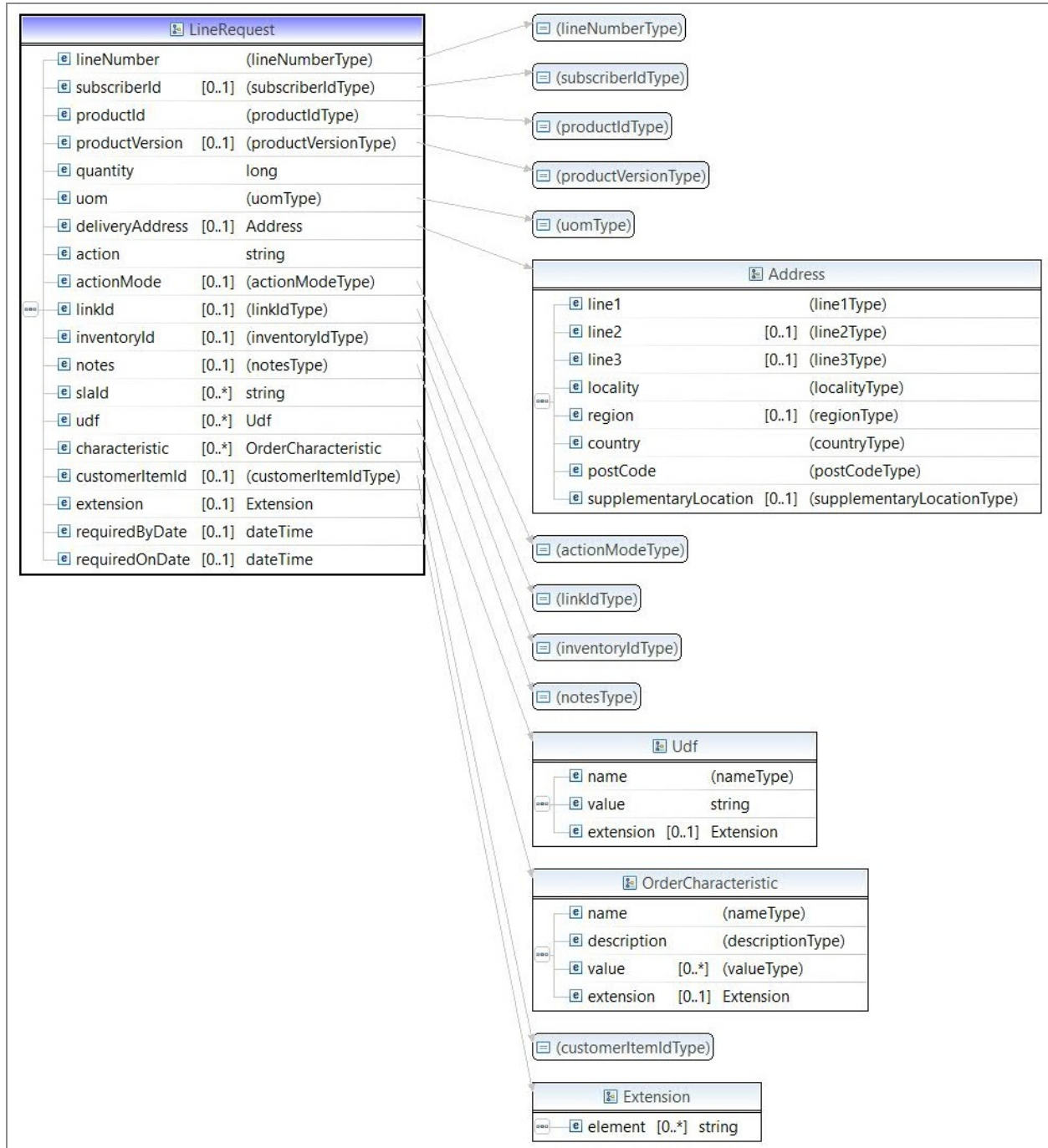
Element	Type	Cardinality	Description
description	String	Optional	Description for the order.
customerID	String	Required	Unique identifier for the customer for this order.
subscriberID	String	Required	Unique identifier for the subscriber for this order.
requiredByDate	DateTime	Optional, Choice	Date and time when this order is required to start.

Element	Type	Cardinality	Description
requiredOnDate	DateTime	Optional, Choice	Date and time by which this order is required to complete.
invoiceAddress	Type	Required	Invoice address type.
invoiceAddress/line1	String	Required	Invoice address line 1.
invoiceAddress/line2	String	Optional	Invoice address line 2.
invoiceAddress/line3	String	Optional	Invoice address line 3.
invoiceAddress/locality	String	Required	Invoice address locality.
invoiceAddress/region	String	Optional	Invoice address region.
invoiceAddress/country	String	Required	Invoice address country.
invoiceAddress/postcode	String	Required	Invoice address post code.
invoiceAddress/supplementaryLocation	String	Optional	Invoice address supplementary location.
deliveryAddress	Type	Required	Delivery address type.
deliveryAddress/line1	String	Required	Delivery address line 1.
deliveryAddress/line2	String	Optional	Delivery address line 2.
deliveryAddress/line3	String	Optional	Delivery address line 3.
deliveryAddress/locality	String	Required	Delivery address locality.
deliveryAddress/region	String	Optional	Delivery address region.
deliveryAddress/country	String	Required	Delivery address country.
deliveryAddress/postcode	String	Required	Delivery address post code.

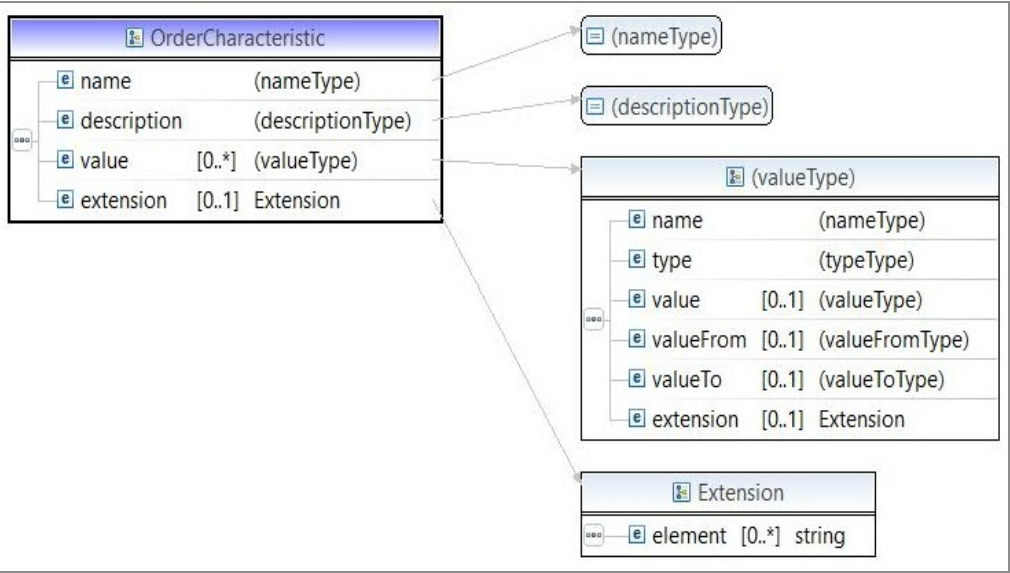
Element	Type	Cardinality	Description
delivery address/ supplementaryLocation	String	Optional	Delivery address supplementary location.
notes	String	Optional	Order notes.
slalD	String	0-M	Unique identifier for an SLA that is applied to this order.
udf	Type	0-M	User defined field type.
udf/name	String	Required	User defined field name.
udf/value	String	Required	User defined field value.
udf/extension	Type	Optional	Extension attributes for user- defined fields.
udf/extension/#any	Any	Required	Any data
extension	Type	Optional	Extension attributes for user- defined fields.
extension/#any	Any	Required	Any data

# Order Request Line

## Order Request Line



Order Line Characteristics



Element	Type	Cardinality	Description
lineNumber	String	Required	Unique identifier for this order line within this order.
subscriberID	String	Optional	Unique identifier for the subscriber for this order line.
productID	String	Required	Product identifier for this order line.
productVersion	String	Optional	Product version for the product for this order line.
quantity	Integer	Required	Quantity of the product being ordered.

Element	Type	Cardinality	Description
uom	String	Required	Unit of measure of the product being ordered.
deliveryAddress	Type	Required	Delivery address type.
deliveryAddress/line1	String	Required	Delivery address line 1.
deliveryAddress/line2	String	Optional	Delivery address line 2.
deliveryAddress/line3	String	Optional	Delivery address line 3.
deliveryAddress/locality	String	Required	Delivery address locality.
deliveryAddress/region	String	Optional	Delivery address region.
deliveryAddress/country	String	Required	Delivery address country.
deliveryAddress/postcode	String	Required	Delivery address post code.
deliveryAddress/supplementaryLocation	String	Optional	Delivery address supplementary location.
action	String	Required	Action for this order line. Valid values are:  1. Provide

Element	Type	Cardinality	Description
			2. Update 3. Cease
actionMode	String	Optional	Supplementary action mode for the action.
requiredByDate	DateTime	Optional, Choice	Date and time by which this order line is required to start.
requiredOnDate	DateTime	Optional, Choice	Date and time by which this order line is required to complete.
linkID	String	Optional	Unique identifier used to link across order lines on this order.
inventoryID	String	Optional	Unique identifier that identifies this order line product with an entry in an external inventory management system.
notes	String	Optional	Order line notes.
slalD	String	0-M	Unique identifier for an SLA that is applied to this

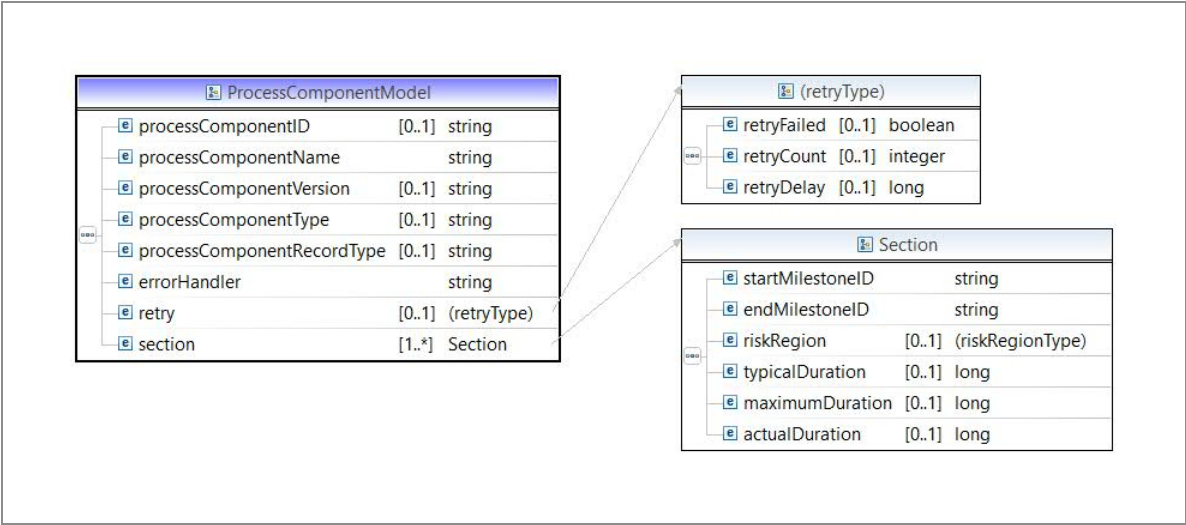
Element	Type	Cardinality	Description
			order line.
udf	Type	0-M	User defined field type.
udf/name	String	Required	User defined field name.
udf/value	String	Required	User defined field value.
udf/extension	Type	Optional	Extension attributes for user-defined fields.
udf/extension/#any	Any	Required	Any data
characteristic	Type	Required	Characteristic type.
characteristic/name	String	Required	Characteristic name.
characteristic/description	String	Required	Characteristic description.
characteristic/value	Type	0-M	Characteristic value type.
characteristic/value/name	String	Required	Characteristic value name.
characteristic/value/type	String	Required	Characteristic value type.
characteristic/value/value	String	Optional	Characteristic



Element	Type	Cardinality	Description
			value.
characteristic/value/valueFrom	String	Optional	Characteristic value from.
characteristic/value/valueTo	String	Optional	Characteristic value to.
characteristic/value/extension	Type	Optional	Extension attributes for user-defined fields.
characteristic/value/extension/#any	Any	Required	Any data
characteristic/extension	Type	Optional	Extension attributes for user-defined fields.
characteristic/extension/#any	Any	Required	Any data
customerItemID	String	Optional	Customer item unique identifier.
extension	Type	Optional	Extension attributes for user-defined fields.
extension/#any	Any	Required	Any data

# Process Component Model

## Process Component Model



Element	Type	Cardinality	Description
processComponentID	String	Required	A unique identifier for the Process Component to be executed.
processComponentName	String	Optional	Process component name for the Process Component to be executed.
processComponentVersion	String	Optional	Process component version for the Process Component to be executed.
processComponentType	String	Optional	Process component type for the Process Component to be executed.
processComponentRecordType	String	Optional	Class of processComponentType.

Element	Type	Cardinality	Description
errorHandler	String	Optional	Error handler to use in the event of the Process Component returning an incomplete or unsuccessful execution response message.
retry	Type	Optional	Retry type.
retry/retryFailed	Boolean	Required	The flag indicating that the orchestrator might retry failed plan items.
retry/retryCount	Integer	Required	Number of times the Orchestrator might retry the failed plan item before referring it to the Plan Item Failed Handler for manual intervention.
retry/retryDelay	Long	Required	Delay in msec between calls to the Process Component if the plan item is retired.
section	Type	1-M	Process component model section type.
section/startMilestoneID	String	Required	Unique identifier for the start milestone that describes this section.
section/endMilestoneID	String	Required	Unique identifier for the end milestone that describes this section.
section/typicalDuration	Long	Required	Typical duration for this

Element	Type	Cardinality	Description
			section is in msec.
section/maximumDuration	Long	Required	Maximum duration for this section is in msec.

# TIBCO Documentation and Support Services

---

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

## How to Access TIBCO Documentation

Documentation for TIBCO products is available on the [Product Documentation website](#), mainly in HTML and PDF formats.

The [Product Documentation website](#) is updated frequently and is more current than any other documentation included with the product.

## Product-Specific Documentation

The documentation for this product is available on the [TIBCO® Order Management Product Documentation](#) page.

## How to Contact Support for TIBCO Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our [product Support website](#).
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the [product Support website](#). If you do not have a username, you can request one by clicking **Register** on the website.

## How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature

requests from within the [TIBCO Ideas Portal](#). For a free registration, go to [TIBCO Community](#).

# Legal and Third-Party Notices

---

SOME CLOUD SOFTWARE GROUP, INC. (“CLOUD SG”) SOFTWARE AND CLOUD SERVICES EMBED, BUNDLE, OR OTHERWISE INCLUDE OTHER SOFTWARE, INCLUDING OTHER CLOUD SG SOFTWARE (COLLECTIVELY, “INCLUDED SOFTWARE”). USE OF INCLUDED SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED CLOUD SG SOFTWARE AND/OR CLOUD SERVICES. THE INCLUDED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER CLOUD SG SOFTWARE AND/OR CLOUD SERVICES OR FOR ANY OTHER PURPOSE.

USE OF CLOUD SG SOFTWARE AND CLOUD SERVICES IS SUBJECT TO THE TERMS AND CONDITIONS OF AN AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER AGREEMENT WHICH IS DISPLAYED WHEN ACCESSING, DOWNLOADING, OR INSTALLING THE SOFTWARE OR CLOUD SERVICES (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH LICENSE AGREEMENT OR CLICKWRAP END USER AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE SAME TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of Cloud Software Group, Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, ActiveMatrix BusinessWorks, TIBCO Runtime Agent, TIBCO Administrator, and Enterprise Message Service are either registered trademarks or trademarks of Cloud Software Group, Inc. in the United States and/or other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only. You acknowledge that all rights to these third party marks are the exclusive property of their respective owners. Please refer to Cloud SG’s Third Party Trademark Notices (<https://www.cloud.com/legal>) for more information.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: <https://scripts.sil.org/OFL>

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

Cloud SG software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the “readme” file for the availability of a specific version of Cloud SG software on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. CLOUD SG MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S), THE PROGRAM(S), AND/OR THE SERVICES DESCRIBED IN THIS DOCUMENT AT ANY TIME WITHOUT NOTICE.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "README" FILES.

This and other products of Cloud SG may be covered by registered patents. For details, please refer to the Virtual Patent Marking document located at <https://www.cloud.com/legal>.

Copyright © 2010-2024. Cloud Software Group, Inc. All Rights Reserved.