



TIBCO® Order Management

User Guide

Version 6.1.0 | October 2024

Contents

Contents	2
Orchestrator	8
Architecture	10
Instance Registration	11
Process for Running Order-related Requests Using EMS	11
Submission of Order Assignment to Instance	12
Orchestrator Gateway Behavior	12
Message Routing On EMS	12
Instance Specific EMS Listeners	14
Internal Order Processor	15
Snapshot Saving Enhancement	16
Orchestrator and AOPD Communication	17
Database Updates	18
Division of State Machine	26
Order Content Caching	26
Cache Management via EMS	27
Southbound Replies On REST	28
XPath Evaluation Caching in AOPD	29
Processing Future-Dated Orders in the Orchestrator	29
Audit Trail and Recovery Notification Handling	30
Namespace Handling for Southbound Replies	31
Routing of REST Requests to EMS	33
Batch Notification	35
Synchronous Event Processing	36
Notification	37
Time Dependency	39

Non-Executing Plan Item	40
Process Component Destination	41
Order Types	42
Amend Order	42
Suspend and Activate Order	43
Order Submission	44
Execution Plan	44
Plan Tasks with Associated Process Components	45
Actions	45
Dependencies	45
Order Header	45
Order Line	47
Global Variables	48
Feasibility Providers	49
Feasibility Request	50
Feasibility Response	51
Feasibility Retry	54
OPD Error Handler	54
Process Components	57
Plan Item Execute Request Event	58
Plan Item Milestone Release Request Event	62
Plan Item Milestone Notify Request Event	65
Plan Item Execute Response Event	67
Plan Item Suspend Request Event	72
Plan Item Suspend Response Event	75
Plan Item Activate Request Event	77
Pre-qualification Failed Handlers	81
Pre-Qualification Failed Request Event	83
Pre-qualification Failed Response Event	85
Plan Item External Error Handlers	88
Plan Item Failed Request Event	91

Plan Item Failed Response Event	93
Broker Service	97
Feature Descriptions	97
Design and Implementation	98
Automated Order Plan Development	145
Overview	145
Model Deployment	145
Product Models Purging	146
Configuration	146
Main Configuration	147
Logs	149
Features	149
Autoprovision	149
Time Dependency	152
Product Specification Field Decomposition	153
Sequencing	155
Delta Provisioning	160
Product Affinity (Plan Item Level)	164
Configurable Handling of CrossLink + ProductComprisedOf Conflicts and Single Use + ProductComprisedOf Conflicts	181
Sort Plan	182
Attribute-Based Decomposition	182
ProductDependsOn and ProductRequiredFor Relationships	188
Dependent and Sibling Products	193
Shared Attributes	195
Intermediate Milestones Dependencies	198
Order Amendment	205
Custom Action	230
Product Id and Product Id Ext.	230
Jeopardy Management System	231

Jeopardy Management	233
Jeopardy Events	235
Understanding Plan	237
Understanding Critical Path	238
Understanding Dependencies	239
Jeopardy Management for Execution Plans	240
Jeopardy Management for Plan Task	240
Must Start On Dependencies	241
Consequential Actions	242
Predictive Jeopardy	242
Jeopardy Services	244
Design and Implementation	247
Jeopardy Detection Cycle	266
Adding Jeopardy Rules On OMS UI	272
Internal Error Handler	279
Internal Error Handler Data Flow Diagram	279
Understanding Data Flow in Internal Error Handler	280
Internal Error Handler Sequence Diagram	282
Searching for Plans with planItem in ERROR State	282
Modifying the Plan Item State	283
Submit the Error Resolution	286
Order Management System User Interface	288
Navigation	289
Dashboard	291
Data Access Interfaces	319
Get Plan	319
Get Plan Request	319
Get Plan Response	322
Get Plan Messages and Message Codes	327

Get Plan Items	328
Get Plan Items Request	328
Get Plan Items Response	330
Get Plan Items Messages and Message Codes	332
Set Plan	333
Set Plan Request	333
Set Plan Response	336
Set Plan Messages and Message Codes	337
Set Plan Item	338
Set Plan Item Request	338
Set Plan Item Response	341
Set Plan Item Messages and Message Codes	341
Best Practices for TIBCO Order Management	343
Exception Handling Guidelines	343
General Approach	343
Example Approach	344
Pre-Qualification Failed Handler	347
Technical Exception Handling	348
Schema References	354
Plan Item	354
ResultStatus	357
Message	358
Order Request	360
Samples	362
Sample Order XML	362
Sample Plan Item XML	363
Sample XPATHs	365
TIBCO Documentation and Support Services	366

Legal and Third-Party Notices	368
--	------------

Orchestrator

This section describes the functions of the Orchestrator component of TIBCO Order Management.

Orchestrator is a Java based micro-service and it can be easily scaled up and down behind a load balancer. This approach simplifies the deployment and administration. The Orchestrator's Java implementation is multi-threaded, which improves performance.

The Orchestrator communicates with the Automated Order Plan Development component using RESTful APIs.

Orchestrator and southbound systems can communicate by using the following modes. This mode can be set in Orchestrator's defaultAckMode property. The modes are as follows:

- [REST](#)
- [MESSAGING](#)

REST: Orchestrator invokes a RESTful API exposed by the southbound system (for example: Processcomponent). In the Orchestrator, when the default acknowledgment mode is set to REST, the Orchestrator invokes the `planitem/planItemExecuteRequest` endpoint implemented by the processcomponent. When you use the acknowledgment mode as REST, the Orchestrator expects the processcomponent to implement RESTful APIs and expose the following endpoints:

- * `/pqf`
- * `/planitem/suspendrequest`
- * `/planitem/milestonerelease`
- * `/planitem/executionrequest`
- * `/planitem/errorhandlerrequest`
- * `/planitem/activaterequest`
- * `/plan/opdErrorHandlerRequest`
- * `/feasibility`

After the request is processed by the southbound system, it invokes an API exposed by Orchestrator corresponding to the request that was received by it. The corresponding REST endpoints are exposed by the Orchestrator and are used by the process-component. The RESTful endpoints are as follows:

- * /v1/order/preQualificationFailedReply
- * /v1/planitem/suspendreply
- * /v1/planitem/milestonenotify
- * /v1/planitem/executionreply
- * /v1/planitem/errorHandlerreply
- * /v1/order/feasibilityReply
- * /v1/plan/error

MESSAGING: When the default acknowledgment mode is Messaging, the Orchestrator sends the outbound notification for the processcomponent over Messaging. There are various categories of notifications that Orchestrator sends over different queues.

Message Type	Queue Name
FeasibilityRequest	tibco.aff.orchestrator.provider.order.feasibility.request
PlanItemExecuteRequest	tibco.aff.orchestrator.planItem.execute.request
PlanItemFailedRequest	tibco.aff.orchestrator.provider.planItem.failed.request
PreQualificationFailedRequest	tibco.aff.orchestrator.provider.order.prequal.failed.request
PlanItemActivateRequest	tibco.aff.orchestrator.planItem.activate.request
PlanItemSuspendRequest	tibco.aff.orchestrator.planItem.suspend.request
MilestoneReleaseRequest	tibco.aff.orchestrator.planItem.milestone.release.request
OPDRequest	tibco.aff.orchestrator.provider.order.opd.request

The processcomponent listens on the respective queues and processes the notifications that are sent by the Orchestrator. Once processed, the processcomponent replies on the

following queues.

Message Type	Queue Name
FeasibilityReply	tibco.aff.orchestrator.provider.order.feasibility.reply
MilestoneNotifyRequest	tibco.aff.orchestrator.planItem.milestone.notify.request
OPDReply	tibco.aff.orchestrator.provider.order.opd.reply
PlanItemExecuteReply	tibco.aff.orchestrator.planItem.execute.reply
PlanItemFailedReply	tibco.aff.orchestrator.provider.planItem.failed.reply
PlanItemSuspendReply	tibco.aff.orchestrator.planItem.suspend.reply
PreQualificationFailedReply	tibco.aff.orchestrator.provider.order.prequal.failed.reply

All failed replies from the southbound system are sent to `orchestratorInboundQueueDead` queue instead of individual queues.

In the request for all the modes, the Orchestrator includes specific headers in its requests, which are expected to be included in the response headers.

- **acknowledge:** The communication mode used by the southbound system to determine the processing semantics.

Architecture

Orchestrator is a Java-based component, which by default runs in standalone mode. This simplifies the deployment and administration. The Java implementation of the Orchestrator is multi-threaded, which improves the performance.

The orchestrator needs to communicate with Automated Order Plan Development. The orchestrator can invoke the Automated Order Plan Development component interface directly.

Instance Registration

Start the Broker service before you start the Orchestrator service. For more information, see [Broker Service](#).

After you start the Orchestrator service, it registers by calling the `/v1/instance/register` API of the Broker service. On successful registration, the Broker service returns the instance ID to the registering instance. This instance ID is used by the Orchestrator service for further communication with other Orchestrator instances or southbound systems.

In cases where an instance encounters a client-side issue (indicated by a 4xx error) during the registration process with the Broker service, the Orchestrator implements a retry mechanism to ensure successful registration.

Process for Running Order-related Requests Using EMS

This section explains the process for routing and running order-related requests using TIBCO Enterprise Messaging Service (EMS).

1. After receiving an order-related request, the system first identifies the instance ID (also known as `instance_id`) associated with the order.
2. The request is then dispatched to EMS. It includes a JMS header named `originator`, with the instance ID specified as its value.
3. The owner instance listens for this request by creating a listener with a selector matching the `originator` value.
4. The owner instance proceeds with processing the request as required.

When making requests to southbound systems, the system includes a header named `originator`, with the instance ID as its value, in one of the following ways:

- For requests sent through EMS, the system includes this information in the JMS header.
- For requests made through REST, the system includes this information in the HTTP Request header.

Submission of Order Assignment to Instance

The Orchestrator processes submit order requests through multiple channels: REST API, SOAP over HTTP, and SOAP over JMS. These request channels are positioned behind a load balancer to ensure efficient distribution of requests.

i Note: During the order submission request, the orderRef must be unique.

The instance receiving submit order request assumes ownership of the order, by assigning its ID to the order.

Orchestrator Gateway Behavior

When AOPD submits a plan to an instance that is unregistered, the Orchestrator acts as a gateway until the instance completes registration with the Broker service.

In this situation, the orchestrator stores the order content in the database. It then sends the plan item execute request to the southbound system, omitting the originator header. Consequently, the instance ID for the order is set to null.

For time-bound orders, where the plan has time-dependency, the `time_scheduler` table does not have an instance ID.

The system expects the southbound system to respond without altering the original headers. The plan item Execute Reply does not contain the originator header in this scenario.

The Broker service monitors for southbound replies that are missing an originator header. On detection, it assigns an instance to the order and updates the instance ID in both the order and `time_scheduler` tables.

Message Routing On EMS

The following table describes about the EMS configuration properties.

Configurations

Property	Description	Default Value
orchestratorInboundQueue	Queue for the Orchestrator to listen to all southbound replies with an originator header.	tibco.aff.orchestrator.inbound.queue
orchestratorInboundNoOriginatorQueue	Queue for the Broker Service to listen to all southbound replies without an originator header.	tibco.aff.orchestrator.inbound.no.originator.queue

Bridges are introduced at the EMS level to enhance message handling between the southbound service and the orchestrator. These bridges enable efficient routing of replies based on the presence or absence of an originator header in the message.

Southbound Message Routing

Message Bridging

When the southbound service sends a reply to the reply queue, EMS bridges the message from the source queue to a designated target queue using a selector.

- **With Originator Header:** Replies with an originator header are routed to `orchestratorInboundQueue`.
- **Without Originator Header:** Replies lacking an originator header are routed to `orchestratorInboundNoOriginatorQueue`.

For example, if the southbound service sends a Plan Item Execute Reply to the `tibco.aff.orchestrator.planItem.execute.reply` queue, depending on whether the originator header is present the service is routed to either the `orchestratorInboundQueue` queue or the `orchestratorInboundNoOriginatorQueue` queue.

This bridging mechanism allows the system to adapt seamlessly, requiring no modifications from the customer. The orchestration and broker services are configured to listen on their respective queues:

- Orchestrator Service: Listens on `orchestratorInboundQueue`.
- Broker Service: Listens on `orchestratorInboundNoOriginatorQueue`.

The introduction of message bridging offers several benefits over the previous model, which relies on microservices having a fixed number of listeners on each queue:

- Thread Utilization: Having fixed listeners on multiple queues can lead to blocked threads when there are no messages, which waste resources.
- Load Balancing: Employing a fixed number of threads on a single queue to process all incoming messages ensures efficient load balancing and optimal resource utilization.

By centralizing message processing to a single queue and employing intelligent routing based on message headers, the system significantly improves efficiency and flexibility in managing southbound messages. This approach minimizes thread blocking and enhances the system's ability to handle varying message volumes effectively.

Instance Specific EMS Listeners

Each instance is configured to exclusively listen to southbound replies that include an 'originator' EMS header, facilitated by JMS selectors. The JMS selectors create these listeners. Every instance creates listeners with the header name 'originator' and value as the instance ID of the instance. An orchestrator, for instance, processes only the messages for orders that it owns.

After the message is picked, it is queued for the `InternalOrderProcessor` worker thread. Each order is processed exclusively by one `InternalOrderProcessor` thread, ensuring that all related messages or events are handled by a single thread and eliminating the need for order locking.

If a southbound system's reply lacks the 'originator' header, the Broker service routes the message to the correct owner instance. For more information, see 'Listener for Southbound Reply Queues (With no Originator)' topic in the [Broker Service](#).

Internal Order Processor

A worker thread pool processes all events of an order in a sequence as follows:

1. Picks tasks from a common queue: In a basic thread pool, threads pick tasks from a common queue, where any thread can process any task. Conversely, in the Orchestrator's worker thread pool, threads are created at startup, each with its own dedicated queue. Each worker thread continuously monitors its queue for tasks to process.
2. Summary of the process: When a message is picked by EMS listeners, it is submitted to the BatchProcessor.
3. Determines the OrderID: The system determines the order ID from the message. When the Orchestrator sends a request to the southbound system, it includes Originator, OrderId, and TenantID in the JMS header, expecting the southbound system to include this header in the replies. If the OrderId is not present in the JMS header, the system determines it from the message content.
4. Finds the worker thread (InternalOrderProcessor) to process the order event:
 - If an InternalOrderProcessor is already assigned, the order event is added to its queue.
 - If an InternalOrderProcessor is not assigned, a new InternalOrderProcessor is assigned to the order. This is done on a round-robin basis.
5. Saves the message or event in the order_event table: After the message is added to the worker thread's queue, it is saved in the order_event table to track incoming messages or events. This ensures that if an Orchestrator instance becomes inactive, these messages can be reassigned to another active instance by the Broker Service.
6. Deletes the message or event from the order_event table after successful processing.

InternalOrderProcessorSize Calculation

- In development mode: The orchestrator sets the internalProcessorSize to 1 by

default.

- In non-development mode: The `internalProcessorSize` is determined as follows:

```
internalProcessorSize = 2 × Runtime.getRuntime  
( ).getAvailableProcessors()
```

This sets the internal processor size to twice the number of available processors.

Overriding `InternalOrderProcessorSize`

The `InternalOrderProcessorSize` value can be overridden by setting the `internalProcessorSize` property through any of the following methods:

- Environment Variable: Override by setting the `internalProcessorSize` as an environment variable.
- Configurator: Use the configurator to define the `internalProcessorSize` property.
- `application.properties` file: Specify the `internalProcessorSize` in the `application.properties` file.

This approach allows for customization flexibility, enabling users to tailor the internal processor size according to their specific needs and configurations.

Snapshot Saving Enhancement

The `InternalOrderProcessor` saves the latest snapshot under specific conditions, rather than after each transaction by the orchestrator.

- No Pending Messages: The snapshot is saved when the `InternalOrderProcessor` work queue has no pending messages.
- Batch Processing: The snapshot is saved after processing every 1000 events.

Orchestrator and AOPD Communication

Configurations

Property	Description	Default Value
planGenerationQueue	Queue on which AOPD listens for all plan generation requests from Orchestrator.	tibco.aff.orchestrator.plan.generation.queue
orchestratorInboundQueue	Queue on which the Orchestrator listens for all southbound replies that contain an originator header.	tibco.aff.orchestrator.inbound.queue
orchestratorInboundNoOriginatorQueue	Queue on which the Broker Service listens for all southbound replies that do not contain an originator header.	tibco.aff.orchestrator.inbound.no.originator.queue

The communication between the Orchestrator and AOPD takes place using EMS.

Submitting plan generation request

The Orchestrator submits the plan generation request to the `planGenerationQueue`.

- **Gateway Mode:** The Orchestrator submits the plan generation request to the `planGenerationQueue` without originator headers. In this scenario, AOPD sends the generated plan to the Broker Service on the `orchestratorInboundNoOriginatorQueue` queue. The Broker service then assigns a new owner for the order and routes the message to the `orchestratorInboundQueue`.
- **Registered Mode:** The orchestrator submits the request with the originator header. Consequently, AOPD sends the generated plan back to the Orchestrator on the `orchestratorInboundQueue` queue.

This modification utilizes EMS to boost both the reliability and efficiency of the communication flow between the Orchestrator and AOPD.

Database Updates

The order content is stored in the `plan_item_data` database.

Note: The database tables given here are for the PostgreSQL. Similarly, you can refer to the `<OM_HOME>/db/dbscripts/oracle` path for the Oracle database values.

The impacted tables are as follows:

- [Order_Data](#)
- [Plan_Item_Data](#)
- [Order_Event](#)

Order_Data

This table stores the information regarding orders.

Column Name in Database	Data Type	Description
partitiondate	Date	Stores the

Column Name in Database	Data Type	Description
		partition date of the order.
orderid	Varchar (128)	Stores the order Id.
orderref	Varchar (128)	Stores the order ref of the order.
order_ser	Text	Stores the latest order request. In case of amendment, this stores the latest order request submitted during the amendment.
plan_ser	Text	Stores the plan generated by the AOPD. This plan does not reflect the latest statuses of plan and plan items.
org_order_ser	Text	In case of amendment, this column stores the original order request submitted

Column Name in Database	Data Type	Description
		during the submit order request.
tenantid	Varchar (128)	Stores the tenant id to which this order belongs.
planid	Varchar (128)	Stores the plan id of the order.
status	Varchar (128)	Stores the order status.
data	Text	<p>Stores the state machine of the order in XML format. This does not represent the entire state machine of the order. Individual Plan Item state machines are saved in the plan item data table.</p> <p>The data type of this column has been changed to CLOB (Character</p>

Column Name in Database	Data Type	Description
		Large Object)/Text.
instance_id	Varchar (250)	Stores the instance Id to which these orders belong.
pre_amendment_scxml	Text	Stores the interim sc XML required during amendment processing.
sequencing_enabled	Boolean	Whether the order is sequenced based on the order sequencing feature.
customer_key	Varchar (250)	Stores the customer key of the sequenced order.
feasibility_request_retry_count	Integer	Stores the feasibility retry count of the order.
businesstransactionid	Varchar (128)	Transaction Id generated by Order Management Server.

Plan_Item_Data

It comprehensively stores all the necessary information related to the plan item of an order.

Column Name in Database	Data Type	Description
id	Varchar (524)	Serves as the unique identifier for the plan item. It is generated by concatenating the orderId, tenantId, and planItemId.
orderid	Varchar (256)	Stores the order id to which this plan item belongs.
tenantid	Varchar (256)	Stores the tenant id to which this plan item belongs.
planitemid	Varchar (256)	Stores the plan item id of the plan item.
data	Text	Stores the state machine of the plan item in XML format.
dependentplanitemids	Text	Stores the plan item ids of other plan items whose

Column Name in Database	Data Type	Description
		milestones are dependent on this plan item.
partitiondate	Date	This field holds the creation date of the data partition.
status	Varchar (256)	This column represents different stages like pending, completed, execution, and more.
execute_request_retry_count	Integer	Indicates how many attempts have been made to retry an execution request.
suspend_request_retry_count	Integer	This field tracks the retry attempts for a suspension request.

Order_Event

This table stores the information regarding the messages or events received by the orchestrator on EMS. It tracks incoming messages or events in case an Orchestrator instance goes inactive. These messages are reassigned to another active instance by the Broker Service.

The older order_event table is renamed as order_event_temp table. This table is no longer required by the orchestrator. Previously, it contained the names of events that failed to

acquire a lock. Until 5.1.0 and 6.0.0 releases, PendingOrderProcessor retrieves pending events from this table. After it has been renamed, you can trigger a migration API to transfer these events back to EMS. After migration, you can drop `order_event_temp` table.

Column Name in Database	Data Type	Description
partitiondate	Date	Date of partition
tasktrackingid	Varchar (250)	Stores the unique identifier of this message. This is used to purge the message once it is processed by the Orchestrator.
originator	Varchar (10)	Stores the instance id to which this order belongs.
orderid	Varchar (250)	Stores the order id of the order.
tenantid	Varchar (250)	Stores the tenant id of the order.
message	Text	Stores the entire message or event received by the Orchestrator.
creationtime	Numeric	Marks the

Column Name in Database	Data Type	Description
		moment the order event is initiated.
nodeid	Varchar (250)	Identifies the specific node associated with the order event uniquely.
status	Varchar (10)	This column represents different stages such as pending, completed, and execution.

The following tables remain unchanged:

- order_in_sequence
- order_in_play
- notification
- order_amendment
- order_messages
- time_scheduler
- Time_scheduler_error

The following tables are dropped as they are no longer required by the orchestrator:

- order_lock
- dead_order_event

Division of State Machine

The SCXML is divided into the following parts:

- Order SCXML: Manages events related to Order, OrderLine, Plan, and Order Amendment.
- Plan Item SCXML: Manages events for individual plan items, with each plan item having its own SCXML stored in the `plan_item_data` table.

The order events are processed in the following sequence:

1. Determines if the event is for the order, order line, plan, amendment, or for a specific plan item.
2. Fetches the required SCXML from the database.
3. Unmarshals the XML to a StateMachine object.
4. Fires the order event.
5. Marshalls the latest snapshot of the StateMachine to XML.
6. Saves this XML back to the database.

By dividing the SCXML into smaller segments, the process of marshaling and unmarshalling becomes more efficient, leading to reduced processing time.

Order Content Caching

All requests for a specific order are processed by a single instance, known as the owner instance. You can use it to cache the order content within the Orchestrator instance.

When a request is received, the orchestrator fetches the order details from its cache. After processing the request, it updates the cache with the most recent snapshot of the order and then synchronizes these updates with the database.

The cache greatly decreases the time needed to load order content for processing requests but is exclusively used for POST requests. For GET requests, any instance can handle them because the database maintains the most current snapshot of the order content.

Orchestrator clears the caches in the following scenarios:

- When an order reaches any of the final states (Complete, Canceled, or Withdrawn)
- Orchestrator cache clear scheduler:
 - The orchestrator utilizes a scheduler that runs according to the cron expression provided in the `orchestratorCacheClearInterval` property.
 - You can utilize this job to clear the cache at regular intervals.
 - This functionality is beneficial for handling long-running orders.

i Note: Order content is only cached after the order reaches the Execution state. Before this state, the order content is managed directly from the database.

Cache Management via EMS

The Orchestrator uses Spring Cache for cache management, you can use it to manage cache contents via EMS queues through a request-response mechanism.

- EMS request queue: `tibco.aff.orchestrator.cache`
- EMS response queue: `tibco.aff.orchestrator.cache.reply`

Message Structure

Requests sent to the `tibco.aff.orchestrator.cache` queue must contain the following headers:

- Originator (Mandatory): Each microservice instance has a unique instance ID. This ID ensures that each microservice instance listens to cache requests specifically meant for it by utilizing the JMS selector feature. Example: `instance-12345`
- Operation (mandatory): Specifies the operation to be performed on the cache.

Supported operations:

- GET: Retrieves the content of the cache for a specified OrderID and TenantID.
- CLEAR: Clears the content of the cache for a specified OrderID and TenantID.
- CLEARALL: Clears the content of all orders in the cache for a specified TenantID.
- ORDER_ID (Conditional): Specifies the OrderID for operations that target specific

orders. It is required for GET and CLEAR operations.

- **TENANTID (Mandatory):** Specifies the TenantID for operations. It is required for all operations (GET, CLEAR, and CLEARALL).

Operations

- **GET Operation**
 - **Description:** Retrieve the content of the cache for the specified OrderID and TenantID.
 - **Required headers:** originator, Operation=GET, ORDER_ID, TENANTID
 - **Response:** The cached content for the specified OrderID and TenantID is sent to the response queue `tibco.aff.orchestrator.cache.reply`.
- **CLEAR Operation**
 - **Description:** Clear the content of the cache for the specified OrderID and TenantID.
 - **Required headers:** originator, Operation=CLEAR, ORDER_ID, TENANTID
 - **Response:** A confirmation message indicating the cache has been cleared for the specified OrderID and TenantID is sent to the response queue `tibco.aff.orchestrator.cache.reply`.
- **CLEARALL Operation**
 - **Description:** Clear the content of all orders in the cache for the specified TenantID.
 - **Required headers:** originator, Operation=CLEARALL, TENANTID
 - **Response:** A confirmation message indicating the cache has been cleared for all orders for the specified TenantID is sent to the response queue `tibco.aff.orchestrator.cache.reply`.

Southbound Replies On REST

The order processing approach has been updated to ensure that all requests for a specific order are exclusively managed by the owner instance. This is accomplished by utilizing the JMS selector feature.

Despite the modifications, REST APIs remain accessible. After receiving an order event through REST, the Orchestrator initially fetches the instance ID of the order from the database. Then, it routes the request to EMS by appending the originator, orderID, and tenantID headers.

For every REST API, a corresponding EMS queue and set of listeners are established. When the owner instance receives a request on EMS, this request is then queued for further processing to the `InternalOrderProcessor`, also referred to as the worker thread.

This modification results in all communications via REST being asynchronous.

XPath Evaluation Caching in AOPD

During plan generation, AOPD evaluates XPath expressions for product decomposition, which are specified in the `DECOMPOSITION_REQUIRED_FOR` field of the product model. These XPath expressions might change alongside modifications to the product model, which is also cached in AOPD.

During plan generation, AOPD evaluates XPath expressions specified in the `DECOMPOSITION_REQUIRED_FOR` field of the product model for product decomposition. These XPath expressions might change with modifications to the product model, which AOPD also caches.

To optimize plan generation time, AOPD now caches the evaluation results of these XPath expressions.

Whenever you modify the product model, AOPD does the followings:

- Removes the modified product model from the cache.
- Also, remove any associated XPath evaluation results to ensure data integrity and accuracy.

Processing Future-Dated Orders in the Orchestrator

The orchestrator enables the submission of future-dated orders by specifying the execution time in the `requiredByDate` field. Plans generated by AOPD have a time dependency on the

eligible plan items. These plan items remain in the pending state until all of their dependencies, including time dependencies, are completed.

Every order, including those scheduled for the future, is processed only by its owner instance. On receiving a future-dated order, Orchestrator:

- Logs it in the `time_scheduler` table with a timestamp and the owner instance's ID.
- The `TimeDependencyScheduler` then checks for time dependencies associated with the owner instance, filtering by the instance ID.
- If relevant dependencies are found, they are marked as completed.

During gateway mode, the instance ID is null in both the `order` and `time_scheduler` tables. The Broker's `Time Dependency Monitor` is responsible for assigning the owner to the order and its time dependencies.

Audit Trail and Recovery Notification Handling

Configure the following properties for the processing of audit trail and recovery notifications.

Configurations

Property	Description	Default Value
<code>auditTrailNotificationQueue</code>	Queue on which the Orchestrator dispatches the audit trail notifications.	<code>tibco.aff.orchestrator.audit.trail.notification.queue</code>
<code>orderEventNotificationQueue</code>	Queue on which the Orchestrator dispatches the recovery notifications.	<code>tibco.aff.orchestrator.order.event.notification.queue</code>

The orchestrator no longer directly saves audit trails and notifications. Instead, it dispatches these notifications to EMS for archival and processing by the Broker Service.

Audit Trail Updates

- **Dispatching Audit Trail Notifications:** The Orchestrator now dispatches audit trail notifications to the Archival Service using the queue specified by the `auditTrailNotificationQueue` property.
- **Archival Service Responsibility:** The Archival Service receives these notifications and saves them in its database.

Impacts

- **API changes:** The following APIs are now managed by the Archival Service:
 - Submit Audit Trail: POST `/order/audit`

Note: For Submit Audit Trail request from TIBCO EMS, the date format must be as follows: 2023-10-09T15:20:28.773Z

- Get Audit Trail: GET `/v1/orders/audit`
- **JMS Listeners:** The JMS listeners for custom audit trails have been moved to the Archival Service.

Recovery Notification Updates

- **Dispatching Recovery Notifications:** The Orchestrator now dispatches recovery notifications to the Broker Service using the queue specified by the `orderEventNotificationQueue` property.
- **Broker Service Responsibility:** The Broker service receives these notifications and saves them in its database.
- **No Functional Impact:** There is no impact on the existing functionality of the Orchestrator or Broker Service.

This new approach improves the separation of concerns and enhances the efficiency of handling audit trails and notifications by leveraging EMS for better scalability and performance.

Namespace Handling for Southbound Replies

The orchestrator listens on a single queue, specified by the `orchestratorInboundQueue` property, for all replies.

Namespaces are essential to correctly identify the type of message. Southbound systems are expected to include namespaces in their replies as part of the JMS header, with the header name `_nm_` and the following values:

JMS Header Property Name	Value
Pre-Qualification Failed Response	PreQualificationFailedResponseEvent
Plan Item Execute Response	PlanItemExecuteResponseEvent
Plan Item Execute Request	PlanItemExecuteRequestEvent
Plan Item Activate Request	PlanItemActivateRequestEvent
Plan Item Suspend Reply	PlanItemSuspendResponseEvent
Milestone Notification Request	PlanItemMilestoneNotifyRequestEvent
Plan Item Error Handler Response / Plan Item Failed Response	PlanItemFailedResponseEvent
Feasibility Response	FeasibilityResponseEvent

Additional Messages Handled by Orchestrator Inbound Queue

Apart from the southbound replies, the `orchestratorInboundQueue` also handles messages routed by the orchestrator from REST to EMS. For these messages, the Orchestrator attaches the appropriate namespace header (`_ns_`) in the JMS message. The namespaces for these messages are as follows:

Header Property Name	Value
Withdraw Order Request	WithdrawOrderEvent
Suspend Order Request	SuspendOrderEvent
Cancel Order Request	CancelOrderEvent
Activate Order Request	ActivateOrderEvent

Header Property Name	Value
Amend Order Request	AmendOrderEvent
Purge Order Request	PurgeOrderEvent
Plan Submission from AOPD	AopdRequestEvent
Plan Generation Failure	AopdPlanErrorNotificationEvent

This namespace handling ensures that the orchestrator can correctly identify and process each type of message, improving efficiency and accuracy in message processing.

For more information, refer to the `$OM_HOME/samples/JMS_Request_Response_Samples.zip` file.

Routing of REST Requests to EMS

All order requests are processed sequentially by the same instance using EMS channels. The orchestrator uses EMS channels to achieve this.

For REST requests, there exists the possibility that a request is received by an instance that is not the owner of the order. Therefore, all POST requests are routed to the EMS with an originator header. These requests are no longer processed directly by the REST API. Instead, the REST API routes the requests to the EMS, where they are processed by the owner instance.

The following APIs are affected by this change:

Operation	API	Queue Name	Remarks
Amend Order	/v1/order/amend	tibco.aff.orchestrator.order.amend	
Withdraw Order	/v1/order	tibco.aff.orchestrator.order.withdraw	

Operation	API	Queue Name	Remarks
Suspend Order	/v1/order/suspend	tibco.aff.orchestrator.order.suspend	
Activate Order	/v1/order/activate	tibco.aff.orchestrator.order.activate	
Cancel Order	/v1/order/cancel	tibco.aff.orchestrator.order.cancel	
Purge Order	/v1/order/purgeOrder	tibco.aff.orchestrator.order.purge	
Bulk Action	/v1/order/bulkaction		Based on the action, value the request is redirected to the corresponding queue
Plan Item Execute Response	/v1/planitem/executionreply	tibco.aff.orchestrator.planItem.execute.reply	
Milestone Notify	/v1/planitem/milestonenotify	tibco.aff.orchestrator.planItem.milestone.notify.request	
Plan Item Suspend Response	/v1/planitem/suspendreply	com.tibco.fom.orch.planitem.suspend.response.queue	
Plan Item Error	/v1/planitem/errorHandler/reply	tibco.aff.orchestrator.provider.planItem.failed.reply	

Operation	API	Queue Name	Remarks
Handler			
Plan Item Bulk Error Handler	/v1/planitem/bulkErrorHandlerReply		Individual Plan Item Error Handler Reply message is dispatched to the EMS with originator headers.
Feasibility Reply	/v1/order/feasibilityReply	tibco.aff.orchestrator.provider.order.feasibility.reply	
Pre-Qualification Failed Reply	/v1/order/preQualificationFailedReply	tibco.aff.orchestrator.provider.order.preQualification.failed.reply	

Batch Notification

Following are the notifications sent by the Orchestrator:

1. Orchestrator sends the JMS notification to the Process Component for running, suspending, and activating the plan items. Orchestrator also needs to notify the Milestone waiting in the Process components
2. Orchestrator also publishes notifications for the state changes of entities such as Order, Order Line, Order Amendment, Plan, and Plan Item. Third-party applications can listen to these notifications.
3. Orchestrator publishes Plan Development notification and Bulk Action Notification as

well. But they are not state change notifications.

The Orchestrator needs to run the actions triggered by the state change. The actions are configured internally to dispatch the JMS Messages, process Database notifications and logging.

Synchronous Event Processing

Events are consumed by the state machine and processed sequentially. Following is the list of the events that are processed by the Orchestrator:

1. Events that are primarily from process components and orchestrator Components.

Below is the sequence of activities involved:

- a. State Machine receives the events from the either Process Component.
- b. Events are consumed by the State machine.
- c. State machine generates the actions to be run. The actions are configured internally to dispatch the Messages for Process Components, Outbound Notifications process, Database notifications and logging.
- d. Actions are run.
- e. Final state orders and checkpoints are cleaned up.

2. Time-Dependent Events that are triggered using Timer.

Below is the sequence of activities involved:

- a. State Machine receives the events from the Timer Event.
- b. State machine generates the actions to be run. The actions are configured internally to dispatch the Messages for Process Components, Outbound Notifications process, Database notifications and logging.
- c. Actions are run.
- d. Final state orders and checkpoints are cleaned up.

Notification

External clients can listen to the notifications about the state changes that are sent by the Orchestrator. The users can filter the following state change notifications:

- [JMS notification](#)

JMS notification

Type of state change	Property name	Default value
Order Status Change	com.tibco.fom.orch.order.statusChange.filter	*
OrderLine Status Change	com.tibco.fom.orch.orderLine.statusChange.filter	*
Plan Status Change	com.tibco.fom.orch.plan.statusChange.filter	*
PlanItem Status Change	com.tibco.fom.orch.planItem.statusChange.filter	*
Order Amendment Status Change	com.tibco.fom.orch.orderAmendment.filter	*
Order State Change Notification topic	com.tibco.fom.orch.outbound.notification.destination	tibco.aff.orchestrator.outbound.notification
OrderLine status change	com.tibco.fom.orch.orderLine.statusChange.destination	tibco.aff.orchestrator.notification.orderLine

Type of state change	Property name	Default value
destination		
Plan status change destination	com.tibco.fom.orch.plan.statusChange.destination	tibco.aff.orchestrator.notification.plan
PlanItem status change destination	com.tibco.fom.orch.planItem.statusChange.destination	tibco.aff.orchestrator.notification.planItem
Order Amendment status change destination	com.tibco.fom.orch.orderAmendment.statusChange.destination	tibco.aff.orchestrator.notification.orderAmendment
ORDER, ORDERLINE, PLAN, PLANITEM, PLANDEVELOPMENT State Change Notification Queue	archivalNotificationQueue	tibco.aff.orchestrator.archival.notification

By default, all notifications are dispatched to the `tibco.aff.orchestrator.outbound.notification.queue`. To disable, set the `enableNotification` property to `false`.

```
{
  "propName": "com.tibco.fom.orch.plan.statusChange.destination",
  "propDescription": "Plan status enableNotificationchange destination",
  "propValue": "tibco.aff.orchestrator.notification.plan",
  "valueType": "string",
  "isTenantProperty": "false"
}
```

com.tibco.fom.orch.noreciprocalaction.planfragmentID: This property is a tenant specific property. The value provided in this property is set as the planfragmentID for the plan items that directly got canceled from the pending state.

The default value for this property is NO_RECIPROCAL_ACTION.

Whenever the order status changes, the Orchestrator sends an EMS message. The Archival system receives this message and saves the necessary information in a database. The Archival database has several tables and does not need locks to process notifications, allowing it to update the relevant tables directly.

Out-of-Sequence Notifications

Since Archival does not process notifications in a sequence, a notification can arrive before an entry is created for that order. In that case, the notification is retried and sent back to the source queue (tibco.aff.orchestrator.archival.notification). The retry settings are:

- archivalNotificationRedeliveryCount (default: 5)
- archivalNotificationRedeliveryDelay (default: 5000 ms or 5 seconds)

Amendment Notifications

When the first status change notification for an amendment is received, the Archival service updates the tables with the amendment changes. During this process, other notifications are not processed. They get retried and sent back to the source queue (tibco.aff.orchestrator.archival.notification) based on the retry configuration until the Archival database updates.

Exception Handling

In other exception scenarios, retries follow the same configuration. If the issue continues, the notification goes to the dead queue (tibco.aff.orchestrator.archival.notification.dead).

Time Dependency

Time dependency in plan items is satisfied when a certain time period has elapsed, or a certain absolute date and time has been reached. Time dependencies take the form of an absolute date time and once the time has reached or passed, then the dependency is considered satisfied.

Time dependencies of a planItem are scheduled to be EXECUTED at the specified absolute time and only executed once the time is reached. If execution fails then the Orchestrator tries to execute it until maximum retries (`timedep.numRetries`) is reached. If it fails during max retries then the Orchestrator puts the order into `time_scheduler_error` for future reference and the time dependencies are not scheduled and not executed by the Orchestrator.

The following properties play a crucial role in time dependency:

Property	Description
<code>timedep.bufferInterval</code>	This parameter specifies the buffer time available for completing a planItem. It represents the buffer period remaining for a future-dated planItem before its execution phase begins. The interval is used to identify dependencies among active plan items.
<code>timedep.numRows</code>	This attribute counts the active time dependencies that still have buffer time remaining. Essentially, it returns the row count of active time dependencies with available buffer time left.
<code>timedep.pollingInterval</code>	This setting establishes the periodic execution frequency of a time scheduler. It helps in scheduling the execution timing of the scheduler.

Non-Executing Plan Item

A non-executing plan item does not have to be submitted to a process component service.

A comma-separated string of planFragment IDs is defined with the property name "Non-Executing Plan-Fragment IDs", present under the "AOPD Functional Configuration" category of the `ConfigValues_AopdService.json` application properties file, which indicates a planItem having these PlanFragments is treated as non-executing planItem.

```
{
  "propName": "nonexecutingPlanfragmentID",
  "propDescription": "Plan-Fragment IDs whose Plan Items are auto
  completed",
  "propValue": "NON_EXECUTING",
  "valueType": "string",
  "isTenantProperty": "true"
}
```

The orchestrator does not send any notification to the process component service and it completes the planItem along with its milestone dependencies.

Process Component Destination

Currently, process component notifications are sent to a default destination if there is no owner defined in the process component or to a destination with the owner name if the owner is defined in the process component. This destination can be overridden and a new destination can be used as the destination for process component notifications. This can be configured by using a property `overridePlanfragmentDestination`. If this property is set to `true`, then the destination is picked from the *ProcessComponent.props* file for the respective process component ID. The content of this property file is loaded when a file is modified and the updated value is used for sending notifications. This is applicable only when the error handler type is internal. This properties file has a destination defined for the process component as follows:

```
<Process Component ID>.destinationName=<Destination value>
```



Note: The destination value is a string.

If there is no destination defined for a process component in a file `ProcessComponent.props` or property `overridePlanfragmentDestination` is configured as *false* then the default behavior is used to, and all the process component-related notifications are sent to a predefined queue. If the property is *true* and the value is not configured, the default destination is used. The default value is *false*.

For an alternate process component, you have to configure `<Process Component ID>` and `<Alternate Process Component ID>` in `ProcessComponent.prop` file.

<Process Component ID>.alternateProcessComponent=<Alternate Process Component ID> and overridePlanfragmentDestination property must be true.

You can track the change details in the **Activity Log** on Order Management Server UI.

Order Types

TIBCO Order Management supports the following order fulfillment modes:

- Amend Order
- Cancel Order
- Suspend Order
- Activate Order
- Withdraw Order

Amend Order

An order amendment is the process of making changes to a previously submitted order. An order can be amended by sending through the new order with the same orderID and orderRef as the previously submitted order.

Orders might only be amended in certain lifecycle states.

Amendable	Not Amendable
<ul style="list-style-type: none">• Plan Development• Execution• Suspended	<ul style="list-style-type: none">• Complete• Canceled• Withdrawn• Feasibility• Pre-Qualification Failed• OPDERROR

Amendments prior to creation of a plan take the form of updating the order in the database and then restarting the order lifecycle back from submitted. At this point a plan

has not been generated and does not have to be modified. When the fulfillment process reaches the Plan Development step, the updated order as it exists from the amendment is used to generate the plan.

For amendments that occur after a plan has been created, but when the plan is still pending, then the order is updated in the database, the existing plan is discarded and the order starts back from submitted. This applies to order status of execution, but with a plan status of pending. However, this is a very rare scenario as the plan immediately goes into EXECUTION state.

For amendments that occur after a plan has started executing only certain aspects of an order might be amended. These are outlined below.

Any other aspects of an order not explicitly detailed here are not amendable. This applies to order and plan status of Execution.

- There is no limitation on the number of amendments that are possible for any given order, but only one amendment might be active at any one time.



Note: It is a best practice to keep the maximum number of amendments under 30 for a particular order.

- Once the amendment has been completed, and the order resumes execution then it is possible to amend the order again.
- If the order goes to Pre-Qualification Failed state from Order Management Server, it cannot be amended.

For more information, see [Order Amendment](#).

Suspend and Activate Order

The order can be suspended at any point during the fulfillment process.

1. If the order is in any of the pre-EXECUTION states, it is suspended immediately.
2. If the order is in EXECUTION state, the Orchestrator sends the suspend requests to all the process components associated with the plan items that are in execution state. At this time, order is moved to SUSPENDING state. These process components might either respond with an execution suspends response, if they can suspend the processing or execution complete response, if they cannot. Based on the response,

the running plan items go into SUSPENDED or COMPLETE state. Finally, the order and plan state is changed to SUSPENDED.

3. The orders that are in final states such as COMPLETE or CANCELLED or already in SUSPENDED state cannot be suspended again.

The suspended orders can be activated back into the EXECUTION state to proceed ahead with the fulfillment.

1. If the order was in any of the pre-EXECUTION states before suspension, it is activated immediately, and processing carry on further.
2. If the order was in EXECUTION state before suspension, the Orchestrator activates it by sending the activation requests for all the process components associated with the plan items that were SUSPENDED. Finally, the order and plan state is changed to EXECUTION.

Order Submission

A customer order is received from an external order capture or request injection system, for example, a CRM system or a Business-to-Business (B2B) gateway, and Web services. The order must be in the JSON format and is received through a RESTful service.

Execution Plan

Execution Plan is a process model, which is developed for a concrete order and can also be termed as a collection of the activities that have to be completed to fulfill a customer order. Execution plans usually specify how the process components must be arranged to fulfill the order.

An execution plan consists of the following:

- Plan Tasks or Plan Items with an associated process component and action
- Actions
- Dependencies on the plan items

Plan Tasks with Associated Process Components

One or more plan tasks or items comprise an execution plan. Each plan item is created to fulfill a particular product against a particular action. The process component specified in plan item is invoked for the fulfillment.

Actions

Each plan task has an action associated with it. These are the possible actions you can select for each plan task:

- Provide
- Cease
- Update
- Cancel

A plan task manages a particular item. Each action defines what needs to be done for a particular item. An action serves as an annotation to make the execution plan more understandable.

Dependencies

Plans are automatically generated by the system based on the product model for a given order.

A dependency can be defined as a relationship between milestones in the plan items.

Order Header

The table below lists the information contained in an order header:

Type	Description
Order Ref ID	A unique identifier supplied by the system that submits the order. The Order

Type	Description
	<p>Reference is used to determine whether the order is a new request. Order Management Server does this by checking if an order with the same Order Reference is already stored in the cache.</p> <p>Note: The value of Order Ref ID must not contain ":"</p>
Order ID	<p>Internal ID of the order generated and assigned by Orchestrator.</p> <p>Note: The value of Order ID must not contain ":"</p>
Status	Current status of an order. For instance, COMPLETE.
Execution Plan	Execution Plan ID.
Required By Date	Indicates the date and time when the order must start fulfillment.
Notes	Notes about the order. Basically, this is any additional text that might be supplied by the summiteer or submitting system.
Subscriber ID	Reference ID of the subscriber.
Customer ID	Used to retrieve the current customer profile and to identify the customer to other systems interested in the order.
Changed Date	Date when the order is changed.
Execution Status	Execution status of an order. For instance, COMPLETE.
Required On Date	Currently not supported.
Invoice Address	The address to invoice for the order, if different from the customer address.

Type	Description
Delivery Address	The address to deliver the order, if different from the customer address.
Service Level Agreements (SLA)	This is a list of the identifiers of any service level agreement that applies to a particular order.

Order Line

An Order contains order lines. Each order line has the following information:

Type	Description
Line	Line no. of an order.
Product ID	The identifier of the specification of the product to be provided.
Inventory ID	Inventory ID.
Action	<p>The action required for the specific product referred to in the order line. You can enter one of the following actions:</p> <ul style="list-style-type: none"> • Provide: The customer has requested a new service. • Cease: The customer has requested that an existing service must cease. • Update: The customer has requested that an existing service be updated in some way. • Cancel: The customer has requested that an existing service must cancel.
Required By Date	Indicates the date and time when the order line must start fulfillment.
Quantity	The number of the product required.

Type	Description
Required On Date	Currently not supported.
Subscriber ID	Reference ID of the subscriber.
Product Version	Version of a particular product.
Link ID	Link reference ID.
Status	The current status of the order. This is automatically filled in and you cannot amend it. The status changes with the order item's lifecycle.
Status Changed	The date and time that the order line status last changed. This is automatically filled in and you cannot amend it. It initially shows the date and time the order line was created and is updated to reflect later status changes.
UOM (Unit of Measurement)	The unit of measure of the product required.
Delivery Address	The address to deliver the order, if different from the customer address.
Characteristics	A list of product characteristics that are supplied as input parameters to the order to provide additional information to the product specification. For instance, this might be the color of a mobile telephone.

Global Variables

The following table lists the important global variables for TIBCO Order Management Orchestrator configuration related to feasibility enabling, retries, and error handling.

Global Variable	Description
com.tibco.fom.orch.feasibilityRequired	Flag that enables/disables an order feasibility request to a feasibility

Global Variable	Description
	provider from the Orchestrator.
com.tibco.fom.orch.enableFeasibilityErrorHandling	Flag that determines whether or not to refer the failed feasibility request to a pre-qualification failed error handler.
enableOpdErrorHandling	Flag that determines whether or not to refer the failed order and plan development request to a opd error handler.
com.tibco.fom.orch.pcErrorHandlerType.	Specifies the name of the default error handler for the process component.
<pre>"allowedValues": ["ExternalErrorHandler", "InternalErrorHandler"]</pre>	

Feasibility Providers

Feasibility Provider is a customer-specific implementation that checks whether an order is feasible for fulfillment. Feasibility might involve network inventory capacity analysis, stock checks, order line validation, or any other number of checks.

Feasibility checking is an optional step in the fulfillment process within Orchestrator. By disabling feasibility checking, no Feasibility Provider is required. However, if feasibility is enabled, then a Feasibility Provider must be available for orders to proceed. Feasibility Providers must conform to the following requirements to be a valid implementation.

Specification

- Receive event messages on a JMS queue or receive events on a REST call.
- Interpret the Feasibility Request event and determine if the order is feasible for fulfillment.

- Create and send a response event on a JMS queue or REST call.
- In the response event, specify whether the feasibility check has completed successfully by setting the completed flag to TRUE if it can determine feasibility or FALSE if it cannot conclude feasibility.
- In the response event specify whether the order has passed feasibility by setting the passed flag to TRUE if the order is feasible, or FALSE if it is not feasible.
- In the response event, optionally specify a list of warning or error messages whether the order has passed feasibility or not.

Since the feasibility checking process is a customer-implemented component, the functional specification for this process is out of scope of this document.

Feasibility Request

Feasibility Request is an event sent by Orchestrator to the Feasibility Provider to request order feasibility checking. It is a request/reply event to a JMS/REST that returns a reply to the default reply for Feasibility Response.

If an exception occurs during feasibility checking, then it must be logged to the Feasibility Provider log. The details of the exception are returned in the response.

Response for JMS

Event Type	Asynchronous request event
Queue or Topic	Queue
Destination	tibco.aff.orchestrator.provider.order.feasibility.request

The event has the following payload:

Feasibility Request



The following table lists the details of the elements.

Element	Type	Cardinality	Description
businessTransactionID	String	Optional	Unique identifier for tracing purposes across function calls.
correlationID	String	Optional	Unique identifier to correlate the request message with a response message.
orderID	String	Required	Order ID for the order to feasibility check.
orderRequest	Type	Required	Order Request type. See Schema References for the specification of this type.

Feasibility Response

Feasibility Response is an event sent by Feasibility Provider back to the Orchestrator in response to a Feasibility Request event. It is a reply event to a JMS/REST.

The response for feasibility has **completed** and **passed** flags. Orchestrator routes the order lifecycle based on the returned value of these flags. The two flags can be used to distinguish between technical and business exceptions. For example, a failure to complete generally indicates a technical exception, so complete is `false`. A validation failure indicates a business exception, where complete is `true`, but passed is `false`.

Completed	This flag indicates whether the feasibility call completed. If this is set to <code>true</code> , the passed flag becomes relevant.
Passed	This flag indicates whether the order has passed feasibility.

Based on different scenarios, these flags must be set as follows:

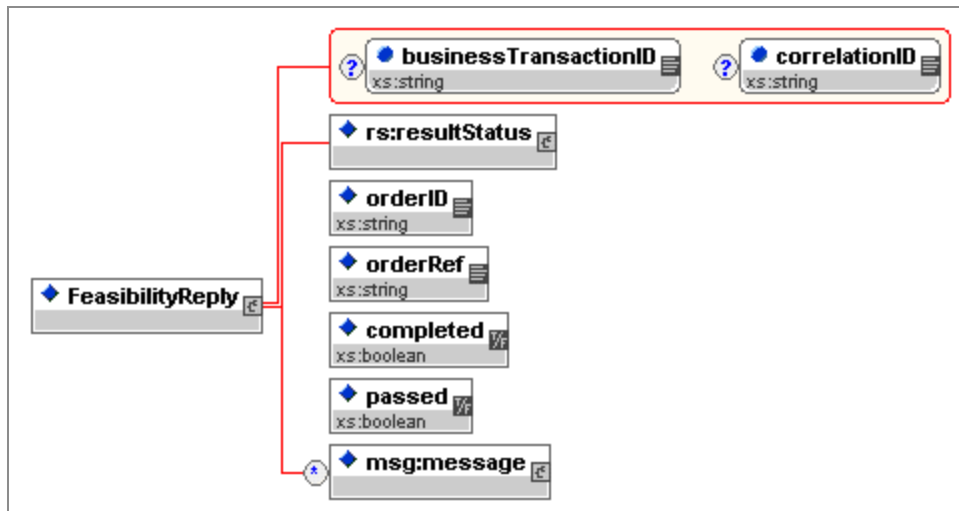
	Completed	Passed	Description
Technical Error	False	False True	Orchestrator refers the order to the Pre-Qualification Failed Handler if error handling is enabled for feasibility, or the error is withdrawn if error handling is not enabled.
Business Error	True	False	Orchestrator refers the order to the Pre-Qualification Failed Handler if error handling is enabled for feasibility, or the error is withdrawn if error handling is not enabled.
Success	True	True	The processing continues as normal.
Event Type	Reply event		
Queue or Topic	Queue		
Destination	tibco.aff.orchestrator.provider.order.feasibility.reply		

The event has the following property:

Property	Type	Cardinality	Description
originator	String	Optional	The value of the originator property in the FeasibilityRequest message, received from the Orchestrator, which must be mapped and sent back in the response message.

The event has the following payload:

Feasibility Response



The following table lists the details of the elements.

Element	Type	Cardinality	Description
businessTransactionID	String	Optional	Unique identifier for tracing purposes across function calls.
correlationID	String	Required	Unique identifier to correlate the request message with a response message. Even though this field is marked as optional in the response schema, it is required for Orchestrator can correlate the response with the correct version of the submitted order. Populate this field the same as correlationID in the request message.
resultStatus	Type	Required	Result status type. See Schema References for the specification of this type.
orderID	String	Required	Order ID for the order that was feasibility checked.

Element	Type	Cardinality	Description
orderRef	String	Required	Order ref for the order that was feasibility checked.
completed	Boolean	Required	Flag indicating whether the feasibility call completed.
passed	Boolean	Required	Flag indicating whether the order has passed feasibility.
message	Type	0-M	Message type. See Schema References for the specification of this type. This list of messages is passed to the Pre-Qualification Failed Handler if invoked.

Feasibility Retry

In case of any technical error when the **completed** flag value is set as `false`, the feasibility retry mechanism ensures that before the feasibility fails, the orders are resubmitted for certain number of times in a certain interval.

You can configure the retry count and retry interval by setting the following property values:

- `com.tibco.fom.orch.feasibilityRetries` for retry count
- `com.tibco.fom.orch.feasibilityRetryInterval` for retry interval (in milliseconds)

OPD Error Handler

Overview


An OPD Error handler is a customer-implemented component used to manage the plan development failure in the orchestrator. During the fulfillment process, the orchestrator always calls out to a plan development provider.

The plan development provider designs a plan from the order. If a plan cannot be designed, then the fulfillment process cannot proceed and, if the `enableOpdErrorHandling` flag is true, the order might be referred to the OPD Error handler for manual intervention.

Specification

The OPD Error handler must conform to the following requirements to be a valid implementation:

- Receive event messages on a JMS or REST queue.
- Interpret the `PreQualificationFailedRequest` event and determine the best way to route the failed order for further processing.
- Create and send a response event on a JMS or REST queue.
- In the response event, specify `RetryOPD` or `Withdraw` as the possible actions that the orchestrator can take in response to the error.

 **Note:** For an amendment request, only `withdraw` action is supported.

The actions that can be specified are as follows:

- **RetryOPD:** The OPD Error handler confirms that the order can be processed further. In this case, the orchestrator sends an order plan generation request to AOPD and moves the order to the OPD state.
- **Withdraw:** The order is withdrawn from the orchestrator and not fulfilled. It is deleted from the engine and Transient Data Store.

The details of the OPD Error handler are left as a customer-specific implementation. An example of a handler can be the following:

1. Receive a `PreQualificationFailedRequest` event message on a JMS queue or REST by a BusinessWorks process.
2. BusinessWorks starts a process instance in a customer build implementation.
3. The TIBCO iProcess® process model creates a manual task and displays a form in a work queue for operations support. The form displays the order and the details of the failure.
4. Operations support reviews the task and chooses one of the following options:

- a. Make changes to the order that ensure that the order passes plan development. The plan development must be retried.
- b. Update configurations in back-end systems or product catalog that ensure that the order passes plan development. The plan development must be retried.
- c. Determine whether the order is invalid and withdraw the order.

The task is completed.

5. TIBCO iProcess® then invokes a BusinessWorks service that creates the PreQualificationFailedReply event, populates the event with the appropriate information, and flags the response to retryopd or withdraw as appropriate. This event is then sent on a JMS or REST queue to the orchestrator. The TIBCO iProcess® procedure then ends.
6. Orchestrator receives the PreQualificationFailedReply and processes it accordingly.

PreQualificationFailedRequest Event

The PreQualificationFailedRequest event is sent by the orchestrator in case the plan fails to generate. The event is received by the OPD Error handler for manual processing. It is an asynchronous event sent to a JMS or REST queue.

Event	Destination Type	Destination	Event Type
PreQualificationFailedRequest	POST (REST)	/v1/pqf	Asynchronous
PreQualificationFailedRequest	JMS Queue	tibco.aff.orchestrator.provider.order.prequal.failed.request	Asynchronous

PreQualificationFailedReply Event

PreQualificationFailedReply event is sent by the OPD Error handler as a response to the failed plan generation. Orchestrator receives the result and interprets it accordingly.

Event	Destination Type	Destination	Event Type
PreQualificationFailedReply	POST (REST)	/v1/preQualificationFailedReply	Synchronous
PreQualificationFailedReply	JMS Queue	tibco.aff.orchestrator.provider.order.prequal.failed.reply	Asynchronous

You can set the following properties in the `ConfigValues_OrchService.JSON` file as per your requirements:

- `com.tibco.fom.orch.retryFailedOPD`: Flag to enable retry of failed OPD request.
- `com.tibco.fom.orch.OPDRetries`: Retry count for failed OPD request.
- `com.tibco.fom.orch.opdRetryInterval`: Interval in millisecond to wait before retrying failed OPD Request.

Process Components

A Process Component is the implementation of a series of steps that are required to fulfill a plan item. Process components must be implemented by using any REST and JMS-enabled technology provided they meet the requirements outlined here.

These are required components in the architecture.

Specification

Process Components must conform to the following requirements to be a valid implementation.

1. Receive event messages on REST endpoint or on a JMS queue.
2. Receive the following event types:
 - a. Plan Item Execute Request
 - b. Plan Item Suspend Request
 - c. Plan Item Activate Request

3. For plan item execute requests, perform a series of tasks that are required to fulfill the product and action specified in the plan item. Once it has completed, send a plan item execute response.
4. When instructed to do so, halt execution at certain milestones until notified by Orchestrator it might continue.
5. For plan item suspend requests, halt execution of an in-progress process component. This might or might not be possible so it is valid to send back a plan item execute response if execution completed, or plan item suspend response if execution was suspended.
6. For plan item activate requests, resume execution of a previously suspended process component. This resume takes the form of one of the following cases:
 - a. Resume execution from the point where it was previously suspended.
 - b. Cancel execution and roll back previously completed tasks.
 - c. Cancel execution and do not roll back previously completed tasks.
7. Create and send response events on a JMS queue.
8. Respond with the following event types:
 - a. Plan Item Execute Response
 - b. Plan Item Suspend Response

Plan Item Execute Request Event

Plan Item Execute Request Event is sent by the Orchestrator to a Process Component to request the fulfillment of a particular plan item. It is received by the Process Component and a series of tasks are run. It can be an asynchronous or synchronous event by a REST service or JMS. The response is another asynchronous or synchronous event on a different orchestrator endpoint or JMS queue.

Event	Destination Type	Destination	Event Type
PlanItemExecute Request	POST (REST)	/v1/planitem/executionrequest	Asynchronous/Synchronous event

Event	Destination Type	Destination	Event Type
PlanItemExecute Request	JMS Queue	tibco.aff.orchestrator.planItem.execute.request	Asynchronous event

Note: The destination name `tibco.aff.orchestrator.planItem.execute.request` is valid only if the owner value is `""`. Otherwise, the destination would be as follows: *(If the owner value is defined), the destination would be* `tibco.aff.orchestrator.planItem.<ownertype>.execute.request`.

For example, if the owner value in the plan fragment model is BPM, the destination would be `tibco.aff.orchestrator.planItem.BPM.execute.request`.

Note: In the case of the REST service, you can check the owner in the header property `processComponentName`.

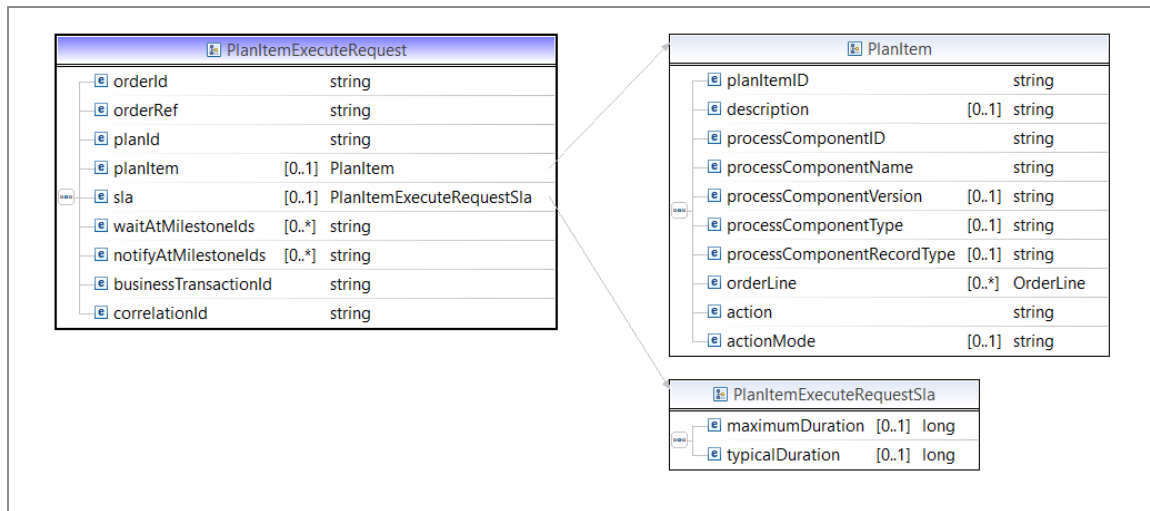
Orchestrator sends the below properties in the header according to their technology (HTTP header, JMS header).

Property	Type	Cardinality	Description
<code>processComponentID</code>	String	Required	Unique identifier for the Process Component to be run.
<code>processComponentName</code>	String	Required	Name of the Process Component to be run. This is the name as configured in the Process Component Model for the specified <code>processComponentID</code> . If a model is not specified, then this field is null.
<code>processComponentVersion</code>	String	Required	Version of the Process

Property	Type	Cardinality	Description
			Component to be executed. This is the version as configured in the Process Component Model for the specified processComponentID. If a model is not specified, then this field is null.
processComponentType	String	Required	Type of the Process Component to be executed. This is the type as configured in the Process Component Model for the specified processComponentID. If a model is not specified, then this field is null.
processComponentRecordType	String	Required	It is a class of processComponentType. This is the processComponentRecordType as configured in the Process Component Model. If a model is not specified, then this field is null.
JMSPriority	Integer	Required	It is the standard JMS message priority to be sent in the outbound message to support order priority.

The payload specification is as follows:

Plan Item Execute Request



The following table lists the details of the elements.

Element	Type	Cardinality	Description
businessTransactionID	String	Optional	A unique identifier for tracing purposes across function calls.
correlationID	String	Optional	A unique identifier to correlate the request message with a response message.
orderID	String	Required	Internal unique identifier for the order associated with the plan containing the plan item to execute.
orderRef	String	Required	External unique identifier for the order associated with the plan containing the plan item to execute.
planID	String	Required	Internal unique identifier for the plan that contains the plan item to execute.
planItem	Type	Required	Plan item type for the plan item to execute. See Schema References for

Element	Type	Cardinality	Description
			the specification of this type.
sla	Type	Optional	Service level agreement type.
sla/typicalDuration	Long	Required	Typical duration in msec for this execution when SLAs are implemented in the Process Component.
sla/maximumDuration	Long	Required	Maximum duration in msec for this execution when SLAs are implemented in the Process Component.
waitAtMilestoneID	String	0-M	Milestone ID for a milestone within the Process Component where execution must wait until notified by the Orchestrator that it must proceed.
notifyAtMilestoneID	String	0-M	MilestoneID for a milestone within the Process Component where the Process Component must notify the Orchestrator that the milestone has been passed during execution.

Plan Item Milestone Release Request Event

Plan Item Milestone Release Event is sent by Orchestrator to a Process Component to instruct it to continue execution when stopped at a particular milestone. It might be possible that this notification occurs before the Process Component has reached the milestone during execution. Therefore, it is necessary for the Process Component to maintain a state of the milestone at any time during execution. There is no response to this interface.

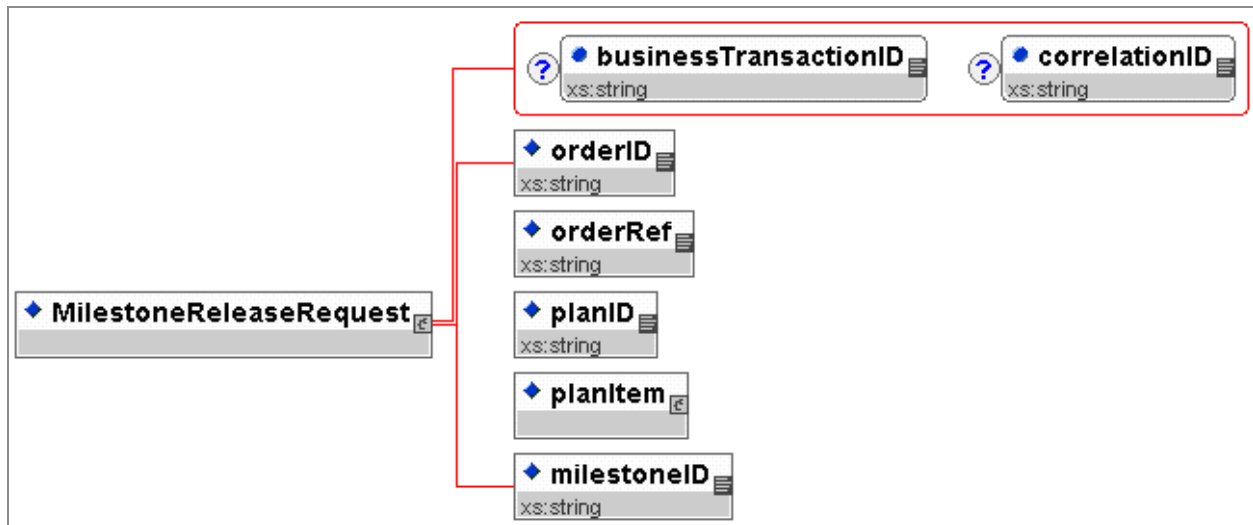
Event	Destination Type	Destination	Event Type
MilestoneReleaseRequest	POST (REST)	/v1/planitem/milestonerelease	Asynchronous/Synchronous event
MilestoneReleaseRequest	JMS Queue	tibco.aff.orchestrator.planItem.milestone.release.request	Asynchronous event

Orchestrator sends the below properties in the header according to their technology (HTTP header, JMS header).

Property	Type	Cardinality	Description
Originator	String	Optional	The value of the NODE_ID that is assigned to the instance. This property is sent by the Orchestrator in all the outbound JMS messages and is expected to be mapped back by the external systems (process components, feasibility providers, pre-qualification failure handlers, and error handlers) in the corresponding response messages.

The payload specification is as follows:

Plan Item Milestone Release Request



Element	Type	Cardinality	Description
businessTransactionID	String	Optional	A unique identifier for tracing purposes across function calls.
correlationID	String	Optional	A unique identifier to correlate the request message with a response message.
orderID	String	Required	Internal unique identifier for the order associated with the plan containing the plan item with the milestone to be released.
orderRef	String	Required	External unique identifier for the order associated with the plan containing the plan item with the milestone to be released.
planID	String	Required	Internal unique identifier for the plan that contains the plan item with the milestone to be released.
planItem	Type	Required	Plan item type for the plan item with

Element	Type	Cardinality	Description
			the milestone to be released. See Schema References for the specification of this type.
milestoneID	String	Required	A unique identifier for the milestone within the plan item and plan to be released.

i Note: If `com.tibco.fom.orch.enableMilestoneReleaseDuringActivation` is set, the milestone is released during the amendment, provided the related dependencies do not change during the amendment. If `com.tibco.fom.orch.enableMilestoneReleaseDuringActivation` is not set, the milestone is not released during amendment in the activated plan item, if it was already released before amendment.

Plan Item Milestone Notify Request Event

Plan Item Milestone Notify Event is sent by a Process Component to the Orchestrator to notify the orchestration engine that a particular milestone has been passed during execution. This event enables the Orchestrator to release the milestone, which was waiting on the current milestone that was notified.

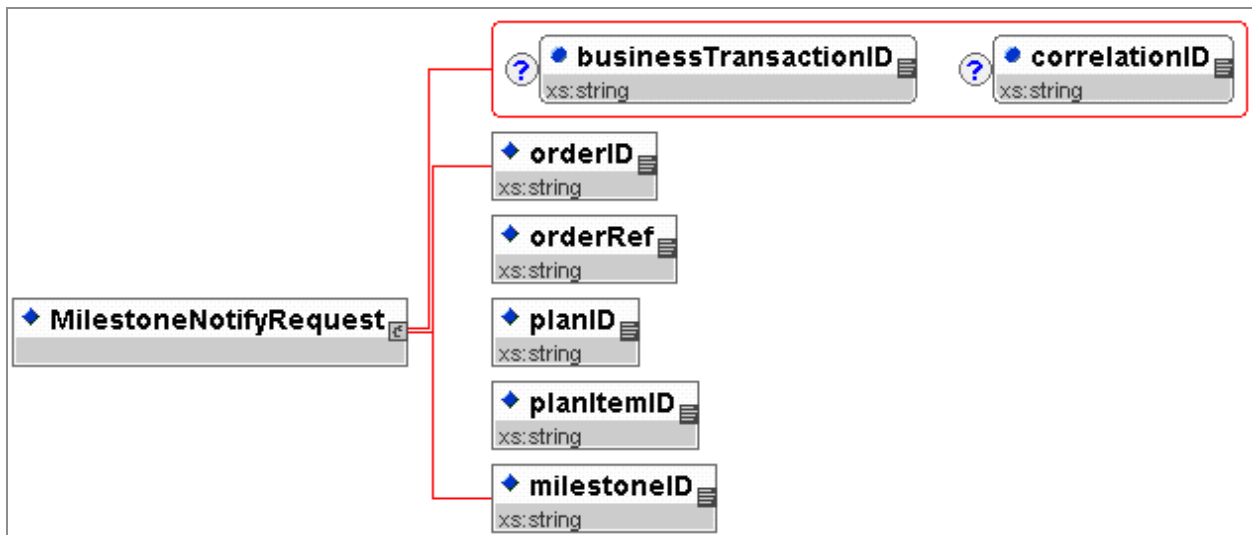
Event	Destination Type	Destination	Event Type
MilestoneNotifyRequest	POST (REST)	/v1/planitem/milestonenotify	Asynchronous/Synchronous event
MilestoneNotifyRequest	JMS Queue	tibco.aff.orchestrator.planItem.milestone.notify.request	Asynchronous event

Orchestrator sends the below properties in header according to their technology (HTTP header, JMS Header).

Property	Type	Cardinality	Description
originator	String	Optional	The value of the originator property in the PlanItemExecuteRequest message, received from the Orchestrator, which must be mapped and sent back in this response message.

The payload specification is as follows:

Plan Item Milestone Notify Request Event



Element	Type	Cardinality	Description
businessTransactionID	String	Optional	Unique identifier for tracing purposes across function calls.
correlationID	String	Optional	Unique identifier to correlate the request message with a response message.
orderId	String	Required	Internal unique identifier for the order associated with the plan containing the plan item with the milestone to notify.

Element	Type	Cardinality	Description
orderRef	String	Required	External unique identifier for the order associated with the plan containing the plan item with the milestone to notify.
planID	String	Required	Internal unique identifier for the plan that contains the plan item with the milestone to notify.
planItemID	String	Required	Unique identifier for the plan item within the plan with the milestone to notify.
milestoneID	String	Required	Unique identifier for the milestone within the plan item in the plan to notify.



Note: `com.tibco.fom.orch.milestone.allowMultipleSpaces` allows milestone to process with multiple spaces in the milestone id.

Plan Item Execute Response Event

Plan Item Execute Response Event is sent by a Process Component as a response to a Plan Item Execute Request Event or a Plan Item Activate Event. The orchestrator receives the result and interprets the result accordingly.

The response for Plan Item Execute has success, completed, and canceled flags. The orchestrator does not act in response to the canceled flag. However, it does route plan items to either Plan Item Internal Error Handler or External Error Handler Component if either completed or success is set to false. Functionally, the orchestrator handles both of these the same. Plan Item Failed Handlers might choose to handle the exception differently depending on completed or failure status.

The two flags can be used to distinguish between technical and business exceptions. For example, a failure to complete is generally indicated a technical exception, so the

completed flag is false here. A validation failure indicates a business exception, where complete is true, but success is false.

Completed	This flag indicates that the Process Component completed. If this is set to true, then the Success flag becomes relevant. If this is false, then the Process Component did not complete and the Success flag is automatically considered to be false as well.
Success	This flag indicates whether the Process Component was successful. This is only relevant if the complete flag is set to true.

The possible response scenarios are:

	Complete	Passed	Description
Technical Error	False	False True	Orchestrator retries the Process Component call for the defined number of retries with the defined retry interval. If the Process Component call continues to fail, then it refers the plan item to the Plan Item Failed Handler.
Business Error	True	False	The orchestrator refers the plan item to the Plan Item Failed Handler.
Success	True	True	Processing continues as normal.

In addition to completed and success values, the Plan Item Execute Response Event also allows returning a canceled flag. This is only valid if responding to a Plan Item Activate Request Event and it indicates whether the cancellation was completed successfully or whether a rollback was requested. completed and success values retain the same definitions in the event of an activation request as in an execution request.

The possible response scenarios are:

	Canceled	Description
Execute Request	False	No cancellation occurred.

Activate Request with Rollback	True	Cancellation requested with rollback.
Activate Request without Rollback	True	Cancellation requested with no rollback.

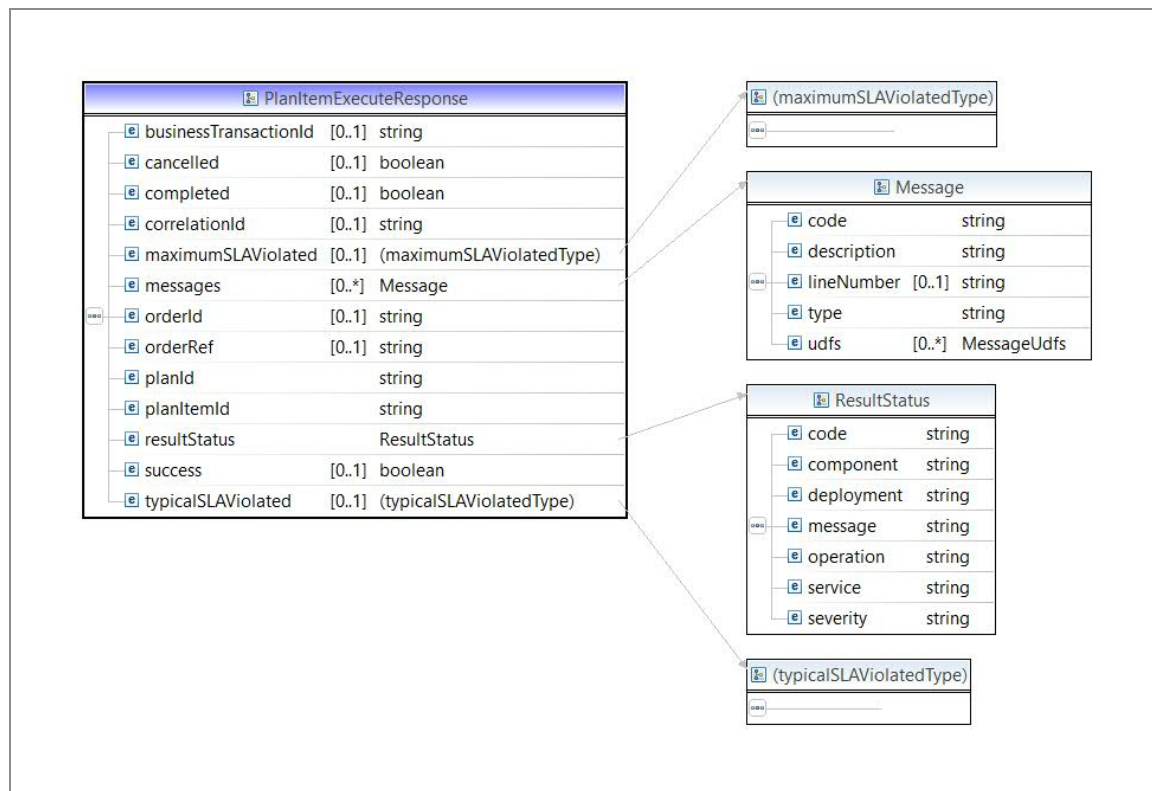
Event	Destination Type	Destination	Event Type
PlanItemExecuteResponse	POST (REST)	/v1/planitem/executionreply	Asynchronous/Synchronous event
PlanItemExecuteResponse	JMS Queue	tibco.aff.orchestrator.planItem.execute.reply	Asynchronous event

The event has the following property:

Property	Type	Cardinality	Description
Originator	String	Optional	The value of the originator property in the PlanItemExecuteRequest message, received from the Orchestrator, which must be mapped and sent back in the response message.

The payload specification is as follows:

Plan Item Execute Response



The following table lists the details of the elements.

Element	Type	Cardinality	Description
businessTransactionID	String	Optional	Unique identifier for tracing purposes across function calls.
correlationID	String	Optional	A unique identifier to correlate the request message with a response message.
resultStatus	Type	Required	Result status type. See Schema References for the specification of this type.
orderId	String	Required	Internal unique identifier for the order associated with the plan containing the plan item to

Element	Type	Cardinality	Description
			execute.
orderRef	String	Required	External unique identifier for the order associated with the plan containing the plan item to execute.
planID	String	Required	Internal unique identifier for the plan that contains the plan item to execute.
planItemID	String	Required	A unique identifier for the plan item within the plan to be executed.
Completed	Boolean	Required	The flag indicating if the Process Component completed processing.
Success	Boolean	Required	The flag indicating if the Process Component completed successfully.
Canceled	Boolean	Required	Flag indicating that the Process Component successfully canceled previously completed tasks.
Message	Type	0-M	Message type. See Schema References for the specification for this type.
typicalSLAViolated	Type	Optional	Flag indicating that the execution time of the Process Component violated the typical SLA duration.
maximumSLAViolated	Type	Optional	Flag indicating that the execution time of the Process Component violated the maximum SLA duration.

Plan Item Suspend Request Event

Plan Item Suspend Request Event is sent by the orchestrator to a process component to request suspension of execution of a particular plan item. It is received by the process component, which then either suspends execution or completes execution.

Event	Destination Type	Destination	Event Type
PlanItemSuspend Request	POST (REST)	/v1/planitem/suspendrequest	Asynchronous/Synchronous event
PlanItemSuspend Request	JMS Queue	tibco.aff.orchestrator.planItem.suspend.request	Asynchronous event



Note: The destination name

`tibco.aff.orchestrator.planItem.suspend.request` is valid only if the owner value is "". Otherwise, the destination is as follows: *(If the owner value is defined), the destination is*

`tibco.aff.orchestrator.planItem.<ownertype>.suspend.request`.

For example, if the owner value in the plan fragment model is BPM, the destination is `tibco.aff.orchestrator.planItem.BPM.suspend.request`.

In the case of REST service, you can check the owner in the header property `processComponent name`.

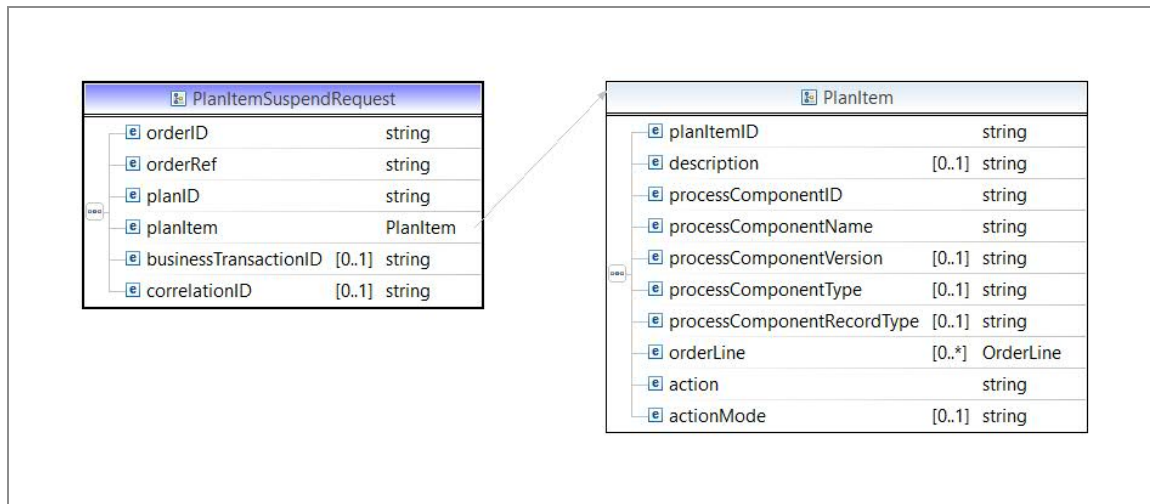
Orchestrator sends the below properties in the header according to their technology (HTTP header, JMS header).

Property	Type	Cardinality	Description
processComponentID	String	Required	A unique identifier for the Process Component to be executed.
processComponentName	String	Required	Name of the Process Component to be executed.

Property	Type	Cardinality	Description
			This is the name as configured in the Process Component Model for the specified processComponentID. If a model is not specified, then this field is null.
processComponentVersion	String	Required	Version of the Process Component to be executed. This is the version as configured in the Process Component Model for the specified processComponentID. If a model is not specified, then this field is null.
processComponentType	String	Required	Type of the Process Component to be executed. This is the type as configured in the Process Component Model for the specified processComponentID. If a model is not specified, then this field is null.
processComponentRecordType	String	Required	It is a class of processComponentType. This is the processComponentRecordType as configured in the Process Component Model. If a model is not specified, then this field is null.
JMSPriority	Integer	Required	It is the standard JMS message priority to be sent in the outbound message to support order priority.

The payload specification is as follows:

Plan Item Suspend Request



The following table lists the details of the elements.

Element	Type	Cardinality	Description
businessTransactionID	String	Optional	A unique identifier for tracing purposes across function calls.
correlationID	String	Optional	A unique identifier to correlate the request message with a response message.
orderID	String	Required	The internal unique identifier for the order associated with the plan containing the plan item to be suspended.
orderRef	String	Required	External unique identifier for the order associated with the plan containing the plan item to be suspended.
planID	String	Required	Internal unique identifier for the plan that contains the plan item to be

Element	Type	Cardinality	Description
			suspended.
planItem	Type	Required	Plan item type for the plan item to be suspended. See Schema References for the specification of this type.

Plan Item Suspend Response Event

Plan Item Suspend Response Event is sent by a Process Component as a response to a Plan Item Suspend Request Event. Orchestrator receives the result and interprets the result accordingly.

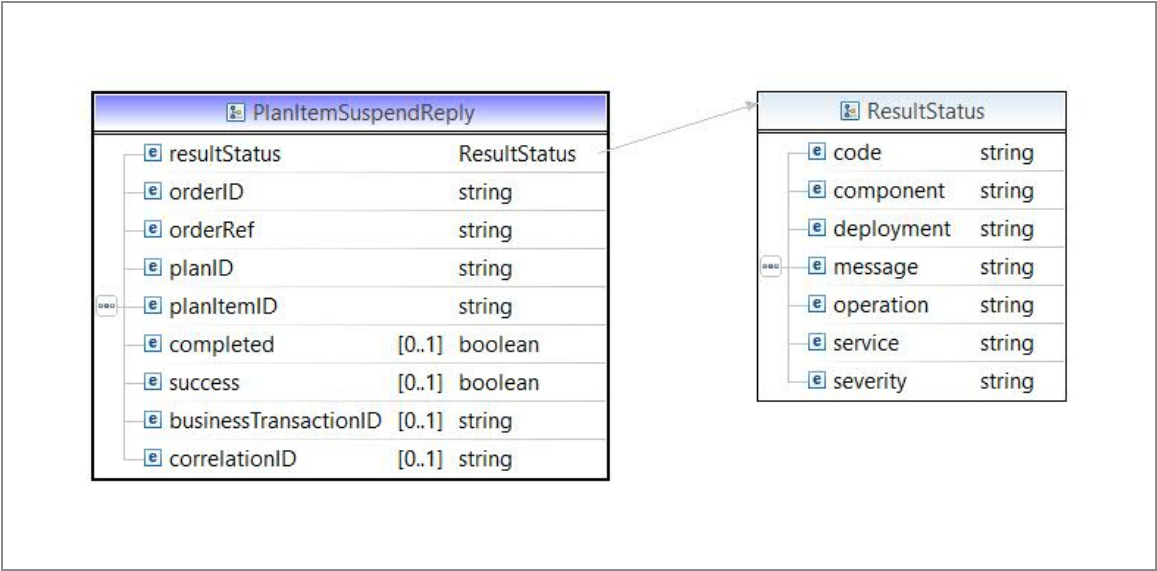
Event	Destination Type	Destination	Event Type
PlanItemSuspendResponse	POST (REST)	/v1/planitem/suspendreply	Asynchronous/Synchronous event
PlanItemSuspendResponse	JMS Queue	tibco.aff.orchestrator.planItem.suspend.reply	Asynchronous event

The event has the following property:

Property	Type	Cardinality	Description
originator	String	Optional	The value of the originator property in the PlanItemSuspendRequest message, received from the Orchestrator, which must be mapped and sent back in the response message.

The payload specification is as follows:

Plan Item Suspend Response



The following table lists the details of the elements.

Element	Type	Cardinality	Description
businessTransactionID	String	Optional	Unique identifier for tracing purposes across function calls.
correlationID	String	Optional	Unique identifier to correlate the request message with a response message.
resultStatus	Type	Required	Result status type. See Schema References for the specification of this type.
orderId	String	Required	Internal unique identifier for the order associated with the plan containing the plan item to suspend.
orderRef	String	Required	External unique identifier for the order associated with the plan containing the plan item to

Element	Type	Cardinality	Description
			suspend.
planID	String	Required	Internal unique identifier for the plan that contains the plan item to suspend.
planItemID	String	Required	Unique identifier for the plan item within the plan to be suspended.
completed	Boolean	Required	Flag indicating if the Process Component suspend completed processing.
success	Boolean	Required	Flag indicating if the Process Component suspend completed successfully.

If any of the flag `completed` or `success` is false in `PlanItemSuspendResponse`, then the process component suspension gets failed and you have to resubmit the `PlanItemSuspendResponse`.

Plan Item Activate Request Event

Plan Item Activate Request Event is sent by Orchestrator to a Process Component to request activation of a previously suspended plan item. It is received by the Process Component, which then resumes, cancels with roll back, or cancels without roll back. It is an asynchronous event to a JMS queue. There is no specific response to a Plan Item Activate Request Event, however the Process Component is expected to complete processing and return a Plan Item Execute Response Event as usual.

Event	Destination Type	Destination	Event Type
PlanItemActivate Request	POST (REST)	/v1/planitem/activaterequest	Asynchronous/Synchronous event

Event	Destination Type	Destination	Event Type
PlanItemActivate Request	JMS Queue	tibco.aff.orchestrator.planItem.activate.request	Asynchronous event



Note: The destination name `tibco.aff.orchestrator.planItem.activate.request` is valid only if owner value is "". Otherwise, the destination is as follows: *(If owner value is defined), the destination is* `tibco.aff.orchestrator.planItem.<ownertype>.activate.request`.

For example, if the owner value in the plan fragment model is BPM, the destination is `tibco.aff.orchestrator.planItem.BPM.activate.request`.

In the case of REST service, you can check the owner in the header property `processComponent` name.

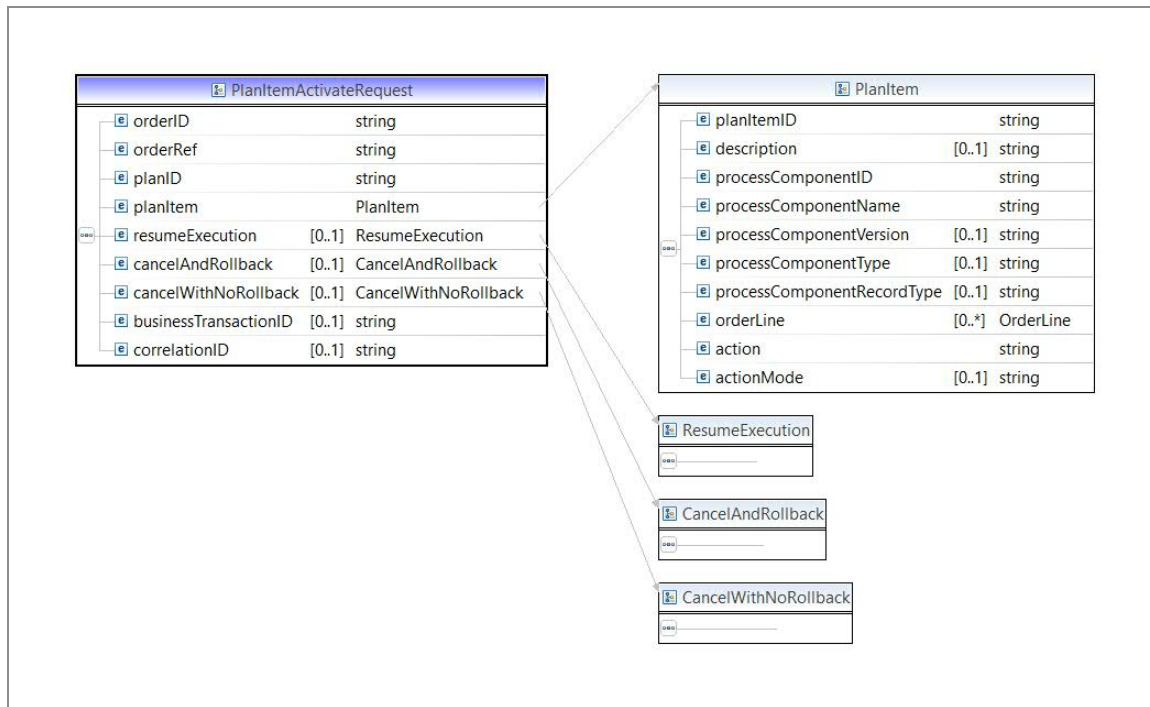
Orchestrator sends the below properties in header according to their technology (HTTP header, JMS Header).

Property	Type	Cardinality	Description
processComponentID	String	Required	Unique identifier for the Process Component to be executed.
processComponentName	String	Required	Name of the Process Component to be executed. This is the name as configured in the Process Component Model for the specified processComponentID. If a model is not specified then this field is null.
processComponentVersion	String	Required	Version of the Process Component to be executed.

Property	Type	Cardinality	Description
			This is the version as configured in the Process Component Model for the specified processComponentID. If a model is not specified then this field is null.
processComponentType	String	Required	Type of the Process Component to be executed. This is the type as configured in the Process Component Model for the specified processComponentID. If a model is not specified then this field is null.
processComponentRecordType	String	Required	It is a class of processComponentType. This is the processComponentRecordType as configured in the Process Component Model. If a model is not specified then this field is null.
JMSPriority	Integer	Required	It is the standard JMS message priority to be sent in the outbound message to support order priority.

The payload specification is as follows:

Plan Item Activate Request



The following table lists the details of the elements.

Element	Type	Cardinality	Description
businessTransactionID	String	Optional	Unique identifier for tracing purposes across function calls.
correlationID	String	Optional	Unique identifier to correlate the request message with a response message.
orderID	String	Required	Internal unique identifier for the order associated with the plan containing the plan item to activate.
orderRef	String	Required	External unique identifier for the order associated with the plan containing the plan item to activate.
planID	String	Required	Internal unique identifier for the plan

Element	Type	Cardinality	Description
			that contains the plan item to activate.
planItem	Type	Required	Plan item type for the plan item to activate. See Schema References for the specification of this type.
resumeExecution	Type	Required, Choice	Flag indicating that the Process Component must resume execution from the point where it was previously suspended.
cancelAndRollback	Type	Required, Choice	Flag indicating that the Process Component must cancel execution and roll back previously completed tasks.
cancelWithNoRollback	Type	Required, Choice	Flag indicating that the Process Component must cancel execution and not roll back previously completed tasks.

Pre-qualification Failed Handlers

Overview

A Pre-Qualification Failed Handler is a customer-implemented component used to manage failed feasibility and plan development steps in an orchestrator. During the fulfillment process, the orchestrator optionally calls out to a Feasibility Provider and always call out to a Plan Development Provider.

The Feasibility Provider analyzes the order to determine if it can be fulfilled. If it indicates that the order cannot be fulfilled, then the fulfillment process cannot proceed. The order might be referred to the Pre-Qualification Failed Handler for manual intervention if feasibility error handling is enabled.

Pre-Qualification Failed handler is an optional component in the architecture. Customers might choose to implement full error handling within the Feasibility Provider. In that case it is not valid to return anything back to the orchestrator other than **passed** in the case of feasibility. If anything else is returned to the orchestrator, there is no handler to process the result, and the plan remains either in a feasibility state.

Specification

Pre-Qualification Failed handlers must conform to the following requirements to be a valid implementation.

1. Receive event messages on a JMS or REST queue.
2. Interpret the Pre-Qualification Failed Request event and determine how best to route the failed order for further processing.
3. Create and send a response event on a JMS or REST queue.
4. In the response event, specify one of three possible actions that the orchestrator is to take in response to the error: resubmit, retry, or withdraw.

The three actions that can be specified are as follows:

RetryOPD	The Pre-Qualification Handler confirms that the order can be processed further. In this case, the Orchestrator sends an order plan generation request to Aopd and moves the order to OPD state.
RetryFeasibility	The orchestrator resubmits the order to the Feasibility Provider as it was originally submitted. If this new call fails, the order is sent back to the Pre-Qualification Failed Handler again.
Withdraw	The order is withdrawn from the Orchestrator and not fulfilled. It is deleted from the engine and Transient Data Store.

The details of the Pre-Qualification Failed Handler are left as a customer-specific implementation. An example of a handler can be the following:

1. Receive a Pre-Qualification Failed Handler Request event messages on a JMS or REST queue by a BusinessWorks process.
2. BusinessWorks starts a process instance in iProcess.
3. The iProcess process model creates a manual task and displays a form in a work

queue for operations support. The form displays the order and the details of the failure.

4. Operations support reviews the task and chooses one of the following options:
 - a. Change the order that allows the order to pass feasibility and the order is processed further.
 - b. Update configurations in back-end systems or product catalog that allows the order to pass feasibility. The order feasibility would then retry.
 - c. Determine the order is invalid and withdraw the order.
5. The task is completed.
6. iProcess then invokes a BusinessWorks service that creates the Pre-Qualification Failed Handler Response event, and populates the event with the appropriate information and flags the response to resubmit, retry, or withdraw as appropriate. This event is then sent on a JMS or REST queue to the orchestrator. The iProcess procedure then terminates.
7. The orchestrator receives the Pre-Qualification Failed Handler Response and processes it accordingly.

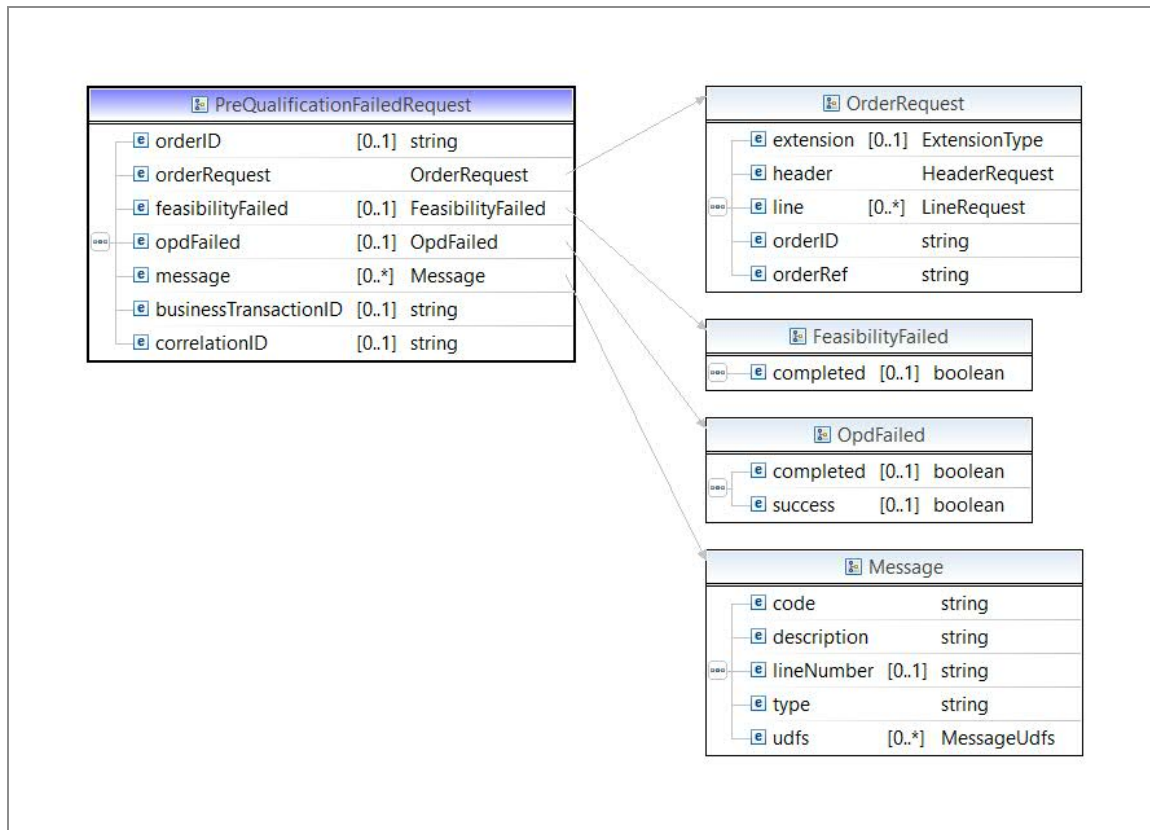
Pre-Qualification Failed Request Event

Pre-Qualification Failed Request Event is sent by Orchestrator in response to a failed feasibility or plan development call. It is received by the Pre-Qualification Failed Handler for manual processing. It can be a JMS or REST queue.

Event	Destination Type	Destination	Event Type
PreQualificationFailedRequest	POST (REST)	/v1/pqf	Asynchronous/Synchronous event
PreQualificationFailedRequest	JMS Queue	tibco.aff.orchestrator.provider.order.prequal.failed.request	Asynchronous event

The event has the following payload:

Pre-Qualification Failed Request



The following table lists the details of the elements.

Element	Type	Cardinality	Description
businessTransactionID	String	Optional	Unique identifier for tracing purposes across function calls.
correlationID	String	Optional	Unique identifier to correlate the request message with a response message.
orderId	String	Required	Order ID for the order that failed feasibility or plan development.
orderRequest	Type	Required	Order Request type for the order that failed feasibility or plan development. See Schema References for the

Element	Type	Cardinality	Description
			specification of this type.
feasibilityFailed	Type	Required, Choice	Flag indicating that this failure was due to a feasibility failure.
opdFailed	Type	Required, Choice	Flag indicating that this failure was due to a plan development failure.
message	Type	0-M	Message type. See Schema References for the specification of this type. This is any list of messages passed back from the Feasibility Provider or the Plan Development Provider.

Pre-qualification Failed Response Event

Pre-Qualification Failed Response Event is sent by Pre-Qualification Failed Handler as a response to the failed feasibility or plan development step. Orchestrator receives the result and interprets the result accordingly.

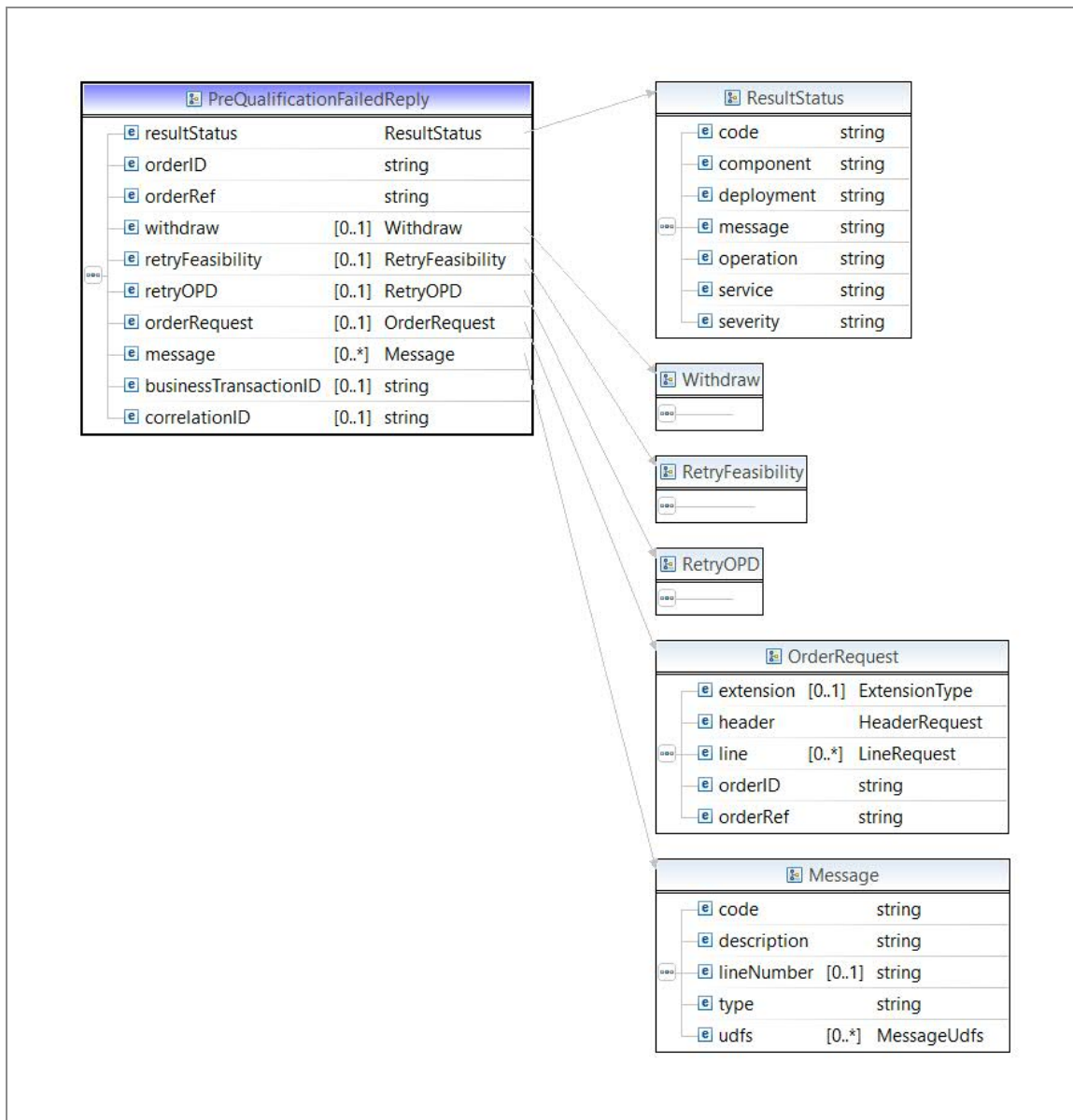
Event	Destination Type	Destination	Event Type
PreQualificationFailedResponse	POST (REST)	/v1/preQualificationFailedReply	Asynchronous/Synchronous event
PreQualificationFailedResponse	JMS Queue	tibco.aff.orchestrator.provider.order.prequal.failed.reply	Asynchronous event

The event has the following property:

Property	Type	Cardinality	Description
originator	String	Optional	The value of the originator property in the PreQualificationFailedRequest message, received from the Orchestrator, which must be mapped and sent back in the response message.

The event has the following payload:

Pre-qualification Failed Response



The following table lists the details of the elements.

Element	Type	Cardinality	Description
businessTransactionID	String	Optional	Unique identifier for tracing purposes across function calls.
correlationID	String	Required	Unique identifier to correlate the request message with a response message. Even though this field is marked as optional in the response schema, it is required for Orchestrator can correlate the response with the correct version of the submitted order. Populate this field the same as correlationID in the request message.
orderID	String	Required	Internal unique identifier for the order that failed feasibility or plan development.
orderRef	String	Required	External unique identifier for the order that failed feasibility or plan development.
withdraw	Type	Required, Choice	Flag indicating that the order must be withdrawn.
retryFeasibility	Type	Required, Choice	Flag indicating that the order must retry feasibility without any changes. This response might be made in cases where the request was for either feasibility or plan development failure.
retryOPD	Type	Required, Choice	Flag indicating that the order must retry plan development without any changes. This response might be made in cases where the request was for plan development failure.

Element	Type	Cardinality	Description
			Technically there is no restriction on doing so in the case of a feasibility failure, but functionally it does not make sense to do so.
orderRequest	Type	Required, Choice	Order request type. See Schema References for the specification of this type. This is provided in the case of an order resubmit.
message	Type	0-M	Message type. See Schema References for the specification of this type. If populated, this list of messages are returned in any Submit Order Response message occurring after the call to the PreQualification Failed Handler occurred.

Plan Item External Error Handlers

Plan Item External Error Handler is a customer-implemented component used to manage failed plan items. During fulfillment, the Orchestrator requests plan item execution from a process component. When execution completes, the Process Component might specify one of three result conditions:

- Execution completed successfully
- Execution completed, but with error
- Execution did not complete

Successful execution results in the plan item being flagged as complete and execution continuing with the next items in the plan.

Error or incomplete execution means that the plan item has not been successfully completed and therefore the next items in the plan must not begin execution until the conditions that caused the failure are rectified.

The orchestrator does not distinguish between an incomplete execution and an error response when determining how to handle the failed plan item. Both are handled in the same way by invoking the Plan Item Error Handler and indicating which failure mode returned. The semantics of how to determine whether a process component is incomplete or in error is left to a customer-specific interpretation and implementation. The implementation between process components and the Plan Item Error Handler must be consistent.

Plan Item Error Handler is an optional component in the architecture. Customers might choose to implement full error handling within the Process Component. In that case it is not valid to return anything back to the orchestrator other than **success**. If anything else is returned to the Orchestrator, there is no handler to process the result and the plan remains in an execution state without progressing beyond the failed plan item.

Specification

Plan Item Error Handlers must conform to the following requirements to be a valid implementation.

1. Receive event messages on a JMS or REST queue.
2. Interpret the Plan Item Failed Request event and determine how best to route the failed plan item for further processing.
3. If necessary, distinguish between incomplete execution and error response execution and interpret the Error Handler field in the Plan Item Failed Request and direct the plan item to the correct handling component.
4. Create and send a response event on a JMS or REST queue.
5. In the response event, specify one of three possible actions that the orchestrator is to take in response to the reply: retry, resume, or complete.

The three actions that can be specified are as follows:

Retry	The plan item is resubmitted to the Process Component to start over from the beginning. This occurs immediately on receipt of the Plan Item Failed Response event as all dependencies have been previously satisfied. The Process Component is notified to re-execute the plan item through a Plan Item Execute Request event. If Orchestrator has been automatically set up to retry failed process components, and this retry fails as well, the retry count is not reset to
-------	--

zero. In other words, if the retry from a Plan Item Failed Handler response fails, then that failure is immediately redirected back to the Plan Item Failed Handler for processing again as a new failed plan item, and not automatically retried further.

Resume	The plan item is resumed in the Process Component from the point of failure. The implementation details of this are left for Process Component design. However, it is conceivable that the Process Component might choose to handle a resume the same as a full retry if it is not functionally possible to resume execution from the point of failure. The Process Component is notified to resume the plan item through a plan item Activate event.
--------	---

Complete	The plan item is considered to be completed and not resubmitted to the Process Component. The orchestrator marks the plan item as Complete and processing continues. Any dependencies on the newly Completed plan item are evaluated and further plan items triggered as in the normal execution.
----------	---

The details of the Plan Item Failed Handler are left as a customer-specific implementation. An example of a handler can be the following:

1. Receive a Plan Item Failed Handler Request event message on a JMS or REST queue by a BusinessWorks process.
 2. BusinessWorks starts a process instance in iProcess.
 3. The iProcess process model makes a service call out to the Transient Data Store to retrieve the order.
 4. The iProcess process model creates a manual task and displays a form in a work queue for operations support. The form displays the order and the details of the failure.
 5. Operations support reviews the task and chooses one of the following options:
 - a. Change the order, which lets the order to process successfully in the Process Component. The changed order is then saved in the Transient Data Store and the plan item might be retried or resumed.
 - b. Change back-end systems that lets the order to process successfully in the Process Component. The plan item might be retried or resumed.
 - c. Implement the required functionality for the plan item manually in the back-end systems. The plan item might be completed.
-

6. iProcess then invokes a BusinessWorks service that creates the Plan Item Failed Handler Response event, populates the event with the appropriate information, and flags the plan item to complete, retry, or resume. This event is then sent on a JMS or REST queue to the orchestrator.
7. The orchestrator receives the Plan Item Failed Handler Response and processes it accordingly.

ShouldFailedPlanItemSuspend Flag

The default value of the ShouldFailedPlanItemSuspend flag is false. This flag is considered only in cases when the ExternalErrorHandler is configured.

Use-cases

When the Orchestrator sends execute request to the process-component and based on certain logic, the process-component decides to reply with an error. In such cases, the request is forwarded to the configured ExternalErrorHandler which is a custom component.

Similarly, when a suspend request is sent by the Orchestrator to the process-component and the process-component replies with an error,

- If the ShouldFailedPlanItemSuspend flag is set to false, on receiving an error in a suspend reply from the process-component, the plan item is sent to ExternalErrorHandler.
- If the ShouldFailedPlanItemSuspend flag is set to true, on receiving an error in a suspend reply from the process-component, the plan item is suspended.

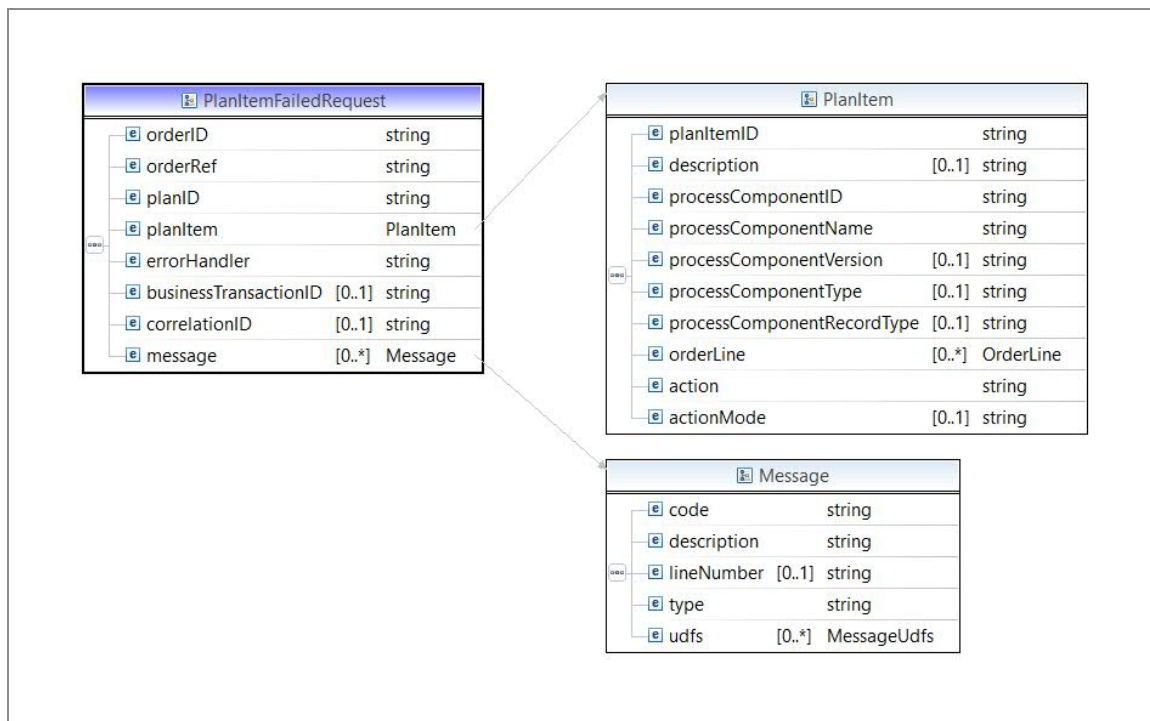
Plan Item Failed Request Event

Plan Item Failed Request Event is sent by Orchestrator in response to a failed plan item. It is received by the Plan Item Failed Handler for manual processing. It is an asynchronous event to a JMS or REST queue.

Event	Destination Type	Destination	Event Type
PlanItemFailed Request	POST (REST)	/v1/planitem/errorhandlerrequest	Asynchronous/Synchronous event
PlanItemFailed Request	JMS Queue	tibco.aff.orchestrator.provider.planItem.failed.request	Asynchronous event

The event has the following payload:

Plan Item Failed Request



The following table lists the details of the elements.

Element	Type	Cardinality	Description
businessTransactionID	String	Optional	Unique identifier for tracing purposes across function calls.

Element	Type	Cardinality	Description
correlationID	String	Optional	Unique identifier to correlate the request message with a response message.
orderId	String	Required	Internal unique identifier for the order associated with the plan containing the failed plan item.
orderRef	String	Required	External unique identifier for the order associated with the plan containing the failed plan item.
planID	String	Required	Internal unique identifier for the plan that contains the failed plan item.
planItem	Type	Required	Plan item type for the plan item that failed. See Schema References for the specification of this type.
errorHandler	String	Required	Name of the error handler to invoke for this failed plan item. This value is either populated from the Process Component Model for the Process Component if it exists or with the default error handler from the Orchestrator configuration.
message	Type	0-M	Message type. See Schema References for the specification of this type. This is the list of messages as returned in the Plan Item Execute Response Event.

Plan Item Failed Response Event

Plan Item Failed Response Event is sent by Plan Item Failed Handler as a response to the failed plan item. Orchestrator receives the result and interprets the result accordingly.

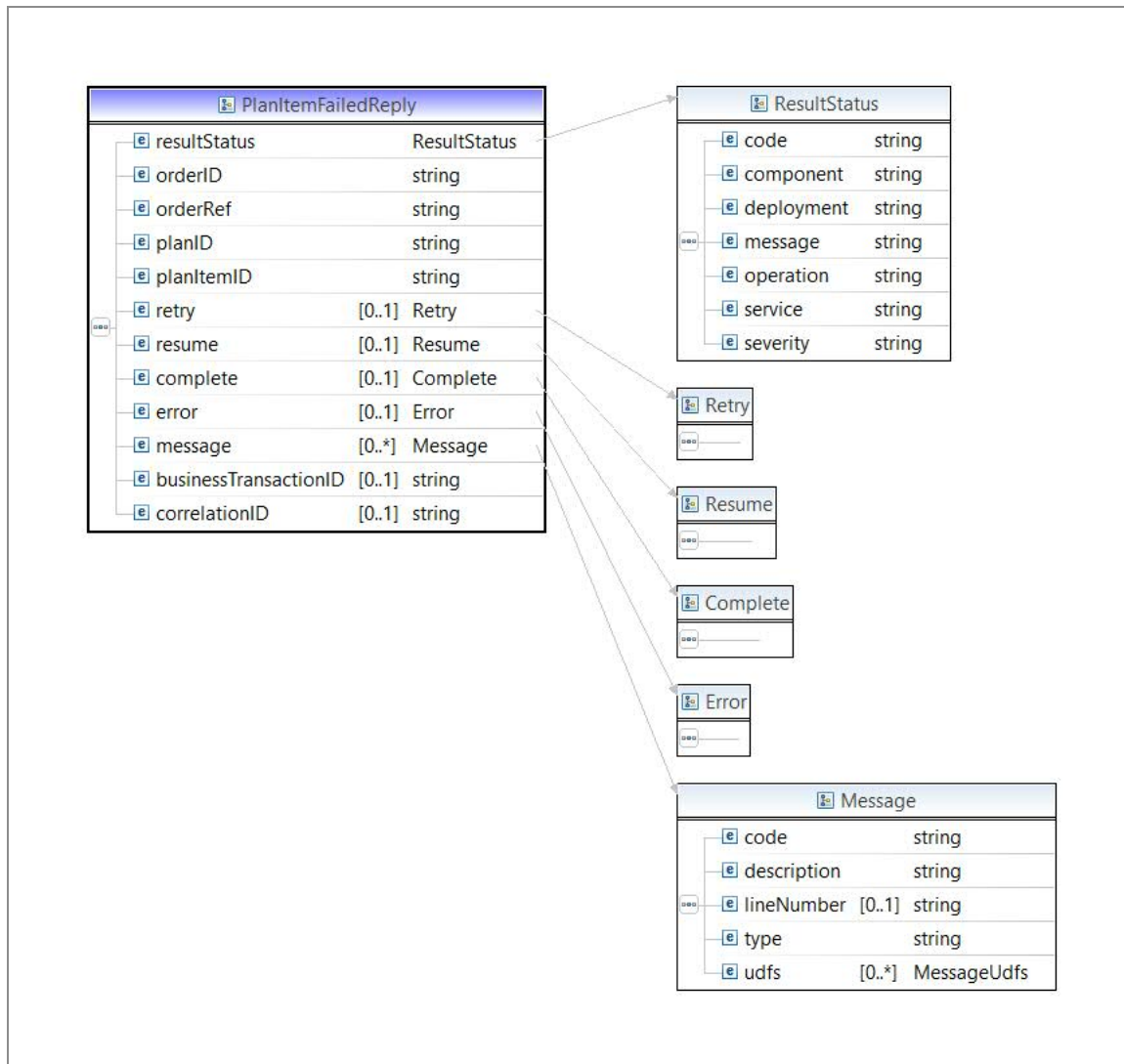
Event	Destination Type	Destination	Event Type
PlanItemFailedResponse	POST (REST)	/v1/planitem/errorHandlerreply	Asynchronous/Synchronous event
PlanItemFailedResponse	JMS Queue	tibco.aff.orchestrator.provider.planItem.failed.reply	Asynchronous event

The event has the following property:

Property	Type	Cardinality	Description
originator	String	Optional	The value of the originator property in the PlanItemFailedRequest message, received from the Orchestrator, which must be mapped and sent back in the response message.

The event has the following payload:

Plan Item Failed Response



The following table lists the details of the elements.

Element	Type	Cardinality	Description
businessTransactionID	String	Optional	Unique identifier for tracing purposes across function calls.
correlationID	String	Required	Unique identifier to correlate the request message with a response message. Even though this field is

Element	Type	Cardinality	Description
			marked as optional in the response schema, it is required for Orchestrator can correlate the response with the correct version of the submitted order. Populate this field the same as correlationID in the request message.
orderId	String	Required	Internal unique identifier for the order associated with the plan containing the failed plan item.
orderRef	String	Required	External unique identifier for the order associated with the plan containing the failed plan item.
planID	String	Required	Internal unique identifier for the plan that contains the failed plan item.
planItemID	String	Required	Unique identifier for the plan item within the plan that failed.
retry	Type	Required, Choice	Flag indicating that the plan item must be retried.
resume	Type	Required, Choice	Flag indicating that the plan item must be resumed from the point of failure.
complete	Type	Required, Choice	Flag indicating that the plan item must be marked as Complete and the execution continues.
message	Type	0-M	Message type. See Schema References for the specification of this type. If populated, this list of messages are returned in any Submit Order Response message occurring

Element	Type	Cardinality	Description
			after the call to the Plan Item Failed Handler occurred.

Broker Service

The Broker service enhances the reliability and availability of Orchestrator instances. In the event of an instance failure, the Broker service redirects orders from the failed instance to all other active instances, ensuring seamless operation and minimal disruption to users.

Each Orchestrator instance registers itself with the Broker service when it starts up. Subsequently, instances regularly send status updates to the Broker service. The Broker service monitors the health of every instance through this ping mechanism.

Feature Descriptions

- [Dedicated REST Endpoint Management for Orchestrator Instances](#)
- [Instance Status Management](#)
- [Instance Failure Mechanism](#)
- [Listener for Southbound Reply Queues \(Without Originator\)](#)

Dedicated REST Endpoint Management for Orchestrator Instances

- Registration of a new instance with the Broker Service
- Deregistration of an instance from the Broker Service
- Instance cleanup operation
- Retrieval of Instance details by Instance ID
- Retrieval of Instance details by Service Name
- Retrieval of all Instance details
- Retrieval of all available services

- [Instance Ping Operation](#)

Instance Status Management

Job schedulers are implemented to periodically monitor the health of every instance. Throughout the instance registration and deregistration processes, the status of instances is passed between different states. The job schedulers ensure that the status of instances is accurately tracked and managed throughout their life-cycle within the Broker service.

To effectively handle these status changes, the following job schedulers are implemented:

`InactiveInstanceMonitor`: Monitors the status of inactive instances.

`ActiveInstanceMonitor`: Monitors the status of active instances.

Instance Failure Mechanism

This mechanism handles instance failures, ensuring the distribution of orders from a failed instance to other active instances.

Listener for Southbound Reply Queues (Without Originator)

A listener is designed to monitor all southbound reply queues without selectors. This is intended to handle situations when southbound systems send replies without headers.

Design and Implementation

This section describes the design of the Broker service and its implementation.

- [Instance Status Descriptions](#)
- [Database Details](#)
- [Configuration Value Details](#)
- [Exposed REST Endpoints](#)
- [Instance Status Management](#)
- [Instance Failure Mechanism](#)
- [Re-assigning Order to an Instance](#)

- [Listener for Southbound Reply With no Originator](#)
- [Recovery Notification Processing](#)
- [Time Dependency Monitor](#)

Instance Status Descriptions

During the instance registration and deregistration processes, instances undergo the following statuses:

- **Pending:** Indicates that an instance is registered but verification is pending to confirm whether the instance has started successfully without errors.
- **Active:** Signifies that an instance is registered and currently active.
- **Inactive:** Indicates that an instance is not currently active. An instance is marked as inactive by the `ActiveInstanceMonitor` flag when it has not communicated or pinged back for an extended period.

Database Details

The following table shows the database details for Broker service.

Instance Ledger

Name	Description	Data Type	
		Oracle DB	PostgreSQL DB
instance_id	Unique identifier for the registered instance	VARCHAR2 (255)	character varying(255)
ip_address	IP address associated with a specific instance	VARCHAR2 (250)	character varying(250)
port	Port number associated with a particular service or endpoint, specifies the network port used for communication.	VARCHAR2 (20)	character varying(20)

Name	Description	Data Type	
		Oracle DB	PostgreSQL DB
status	Current operational status or state of a system component or service, aiding in monitoring and management by indicating whether it is active, inactive	NUMBER(10, 0)	numeric(10, 0)
service_name	The name or identifier of a specific service, aiding in the categorization and organization of services within the system.	VARCHAR2 (50)	character varying(50)
last_updated_timestamp	Last timestamp for instance status updated	NUMBER(20, 0)	numeric(20, 0)

The database scripts are present at the following locations:

PostgreSQL database: <OM_HOME>/db/dbscripts/postgreSQL/broker

Oracle database: <OM_HOME>/db/dbscripts/oracle/broker

Configuration Value Details

Properties File Update

Before the server startup, the Broker service uses the configurator component to retrieve essential configuration properties from the application.properties file, which ensures a seamless startup process.

Property Name	Property Value	Description
server.port	9105	Port at which the broker service starts.

Property Name	Property Value	Description
configuratorServiceUrl	http://localhost:9090	Configurator base URL to connect and download configuration properties and configuration files.
configuratorServiceRetryCount	5	Number of retries to perform in case of request failure to configurator.
configuratorServiceRetryDuration	5	Delay between each retry, in seconds.
configuratorTrustStoreAbsolutePath	C:/Users/cacert	Trust store file path if configurator is on HTTPS.
configuratorTrustStorePassword	tibco123	Trust store password if configurator is on HTTPS.
configuratorTrustStoreType	jks	Trust store type if configurator is on HTTPS.
security.key	ENC (nSa0k6lmjPPN8ZA5SO6BpQ==)	Security key to securely

Property Name	Property Value	Description
		connect to Configurator. The default value is encrypted of 1t1s@asy.

Update Broker Service configuration values

The configurator must have certain configuration properties available. Before the server starts, you must upload these configuration properties to the configurator. These properties are available in the \$OM_HOME/seed-data/app-properties/ConfigValues_BrokerService.json file.

Catalog Client Configuration

When the Broker service triggers pending southbound notifications, it populates the plan item details in some of those notifications. To achieve this, the Broker service connects to the Catalog Service and fetches the latest plan fragment details associated with the plan item, then includes these details in the southbound notifications.

Conversely, the Orchestrator fetches these notifications by directly connecting to the catalog database and caching the details in its memory. Unlike the Orchestrator, the Broker service retrieves data through REST calls to the Catalog service and does not cache the responses.

Property Name	Property Value	Description
catalogServiceBaseUrl	http://localhost:9092	Catalog Service base URL.
catalogTrustStorePassword	tibco123	Trust store password if Catalog is on HTTPS.
catalogTrustStoreType	jks	Trust Store

Property Name	Property Value	Description
		type if Catalog is on HTTPS.
catalogTrustStoreFileName	cacert	Trust Store certificate file name. This is required if the catalog is running on HTTPS. The Broker Service downloads this file from the configurator.

Broker Database Configuration

The Broker Service manages instance life-cycles by storing instance information in a database. For more information, see the [Instance Management APIs](#).

Property Name	Property Value	Description
brokerDsUrl	jdbc:postgresql://localhost:5432/broker dbll? currentSchema=brokerschemall	JDBC URL of the broker database.
brokerDsUsername	brokeruserll	User name of the broker database.
brokerDsPassword	brokeruserll	Password of the broker database.
brokerDsInitializeSize	2	Number of connections

Property Name	Property Value	Description
		that are to be established when the connection pool is started.
brokerDsMaxIdle	11	Maximum number of connections that must be kept in the idle pool.
brokerDsMaxActive	12	Maximum number of active connections that can be allocated from this pool at the same time.
brokerDsMaxWait	1000	Maximum number of milliseconds that the pool waits when there are no available connections.
brokerDsTestOnBorrow	false	Enable connection validation

Property Name	Property Value	Description
		before being borrowed from the pool.
brokerDsValidationInterval	5000	Data source validation interval in milliseconds.
brokerDsTestWhileIdle	true	Enable connection validation while idle in connection pool.
brokerDsTimeBetweenEvictionRuns Millis	5000	Minimum amount of time in milliseconds an object must sit idle in the pool before it is eligible for eviction.
brokerDsMinEvictableIdleTimeMillis	5000	Minimum amount of time in milliseconds an object might sit idle in the pool.
brokerDsNumTestsPerEvictionRun	5	Maximum

Property Name	Property Value	Description
		number of connections to examine during each evictor run.
brokerDsDefaultAutoCommit	false	Enable auto commit after each transaction.
brokerDsRollbackOnReturn	false	Enable transaction rollback when connection is returned back to the pool.
brokerDsCommitOnReturn	false	Enable transaction commit when connection is returned back to the pool.
brokerDsCustomProperty		Pooled Database Custom Property.

Instance Management Configuration

The Broker service includes job schedulers that monitor instance life-cycles. The following properties are required for these jobs. For more information, see [Instance Status Management](#) and [Instance Failure Mechanism](#).

Property Name	Property Value	Description
activeInstanceMonitorInterval	20	Time interval, measured in minutes, at which ActiveInstanceMonitor operates or runs.
activeInstanceMonitorThreshold	5	The duration parameter used by ActiveInstanceMonitor to identify and declare inactive instances based on the timestamp comparison (LastUpdatedTimestamp) against the defined threshold.
inactiveInstanceMonitorCronExpression	0 */20 * * * ?	A Cron expression that determines the execution schedule of InactiveInstanceMonitor.
orderEventTransferTaskQueue	com.tibco.broker.order.event.task.queue	Queue on OrderEventTransfer task would be published internally by Broker Service.
orderEventTransferTaskListenerCount	10	Number of concurrent listener on the orderEventTransferTask Queue.
orderEventTransferTaskRetryCount	5	Retry count for orderEventTransferTask.
orderEventTransferTaskRetryInterval	1	Retry interval between

Property Name	Property Value	Description
interval		orderEventTransferTask .
timeDependencyMonitorInterval	1	Fixed interval in minutes to monitor time dependency with null instance id.

No Originator Listener Configuration

The Broker maintains a connection to the EMS server to listen for southbound replies. This includes listening for replies from currently inactive instances and those lacking the required originator header. Additionally, to ensure the Broker service can recover requests sent to the southbound system, it must be configured with the appropriate queues.

For more information, see the following topics:

- [Instance Status Management](#)
- [Instance Failure Mechanism](#)
- [Re-assigning Order to an Instance](#)
- [Listener for Southbound Reply With no Originator](#)

Property Name	Property Value	Description
orchestratorInboundNoOriginatorQueue	tibco.aff.orchestrator.inbound.no.originator.queue	Queue on which Orchestrator listens for incoming replies from the southbound system.

Property Name	Property Value	Description
orchestratorInboundNoOriginatorDeadQueue	tibco.aff.orchestrator.inbound.no.originator.dead.queue	Dead Queue on which Orchestrator sends the unprocessable messages.
orchestratorInboundNoOriginatorReceiverCount	5	Number of concurrent consumers on each southbound response queue.
orchestratorInboundNoOriginatorRetryCount	5	Number of retries in case of any exception.
orchestratorInboundNoOriginatorRetryInterval	5000	Delay in Milliseconds between consecutive retries.

Recovery Notification Configuration

Broker service requires the following properties to process recovery notifications. For more information, see [Recovery Notification Processing](#).

Property Name	Property Value	Description
recoveryNotificationQueue	tibco.aff.orchestrator.order.event.notification.queue	The queue to listen to the recovery notification sent by Orchestrator.
recoveryNotificationDeadQueue	tibco.aff.orchestrator.order.event.notification.queue.dead	Dead Queue to send the unprocessable recovery notification.
recoveryNotificationReceiverCount	5	Number of listeners to listen to the recovery notification.
recoveryNotificationRetryCount	5	Number of retries in case of failure.
recoveryNotificationRetryInterval	5000	Interval in milliseconds between each retry.

Update Orchestrator Service Configuration Values

You need to update the Orchestrator service configuration values. The properties are available in the `$OM_HOME/seed-data/app-properties/ConfigValues_OrchService.json` file.

Order Database Configuration

Broker service interacts with the order database to carry out the following functions:

- Re-trigger pending notifications.
- Determine a new instance ID for an order in situations where,
 - The owner instance becomes inactive.
 - Southbound replies do not include the required originator header.

For more information, see the following topics:

- [Recovery APIs](#)
- [Instance Status Management](#)
- [Instance Failure Mechanism](#)
- [Re-assigning Order to an Instance](#)
- [Design and Implementation](#)

Property Name	Property Value	Description
orderDsUrl	jdbc:postgresql://localhost:5432/ orderdbllhf7?currentSchema=ordersche mallhf7	JDBC URL of the broker database.
orderDsUsername	orderuserll	User name of the broker database.
orderDsPassword	orderuserll	Password of the broker database.
orderDsInitializeSize	2	Number of connections that are to be established when the connection pool is started.

Property Name	Property Value	Description
orderDsMaxIdle	11	Maximum number of connections that must be kept in the idle pool.
orderDsMaxActive	12	Maximum number of active connections that can be allocated from this pool at the same time.
orderDsMaxWait	1000	Maximum number of milliseconds that the pool waits when there are no available connections.
orderDsTestOnBorrow	false	Enable connection validation before being borrowed from the pool.
orderDsValidationInterval	5000	Data source

Property Name	Property Value	Description
		validation interval in milliseconds.
orderDsTestWhileIdle	true	Enable connection validation while idle in connection pool.
orderDsTimeBetweenEvictionRuns Millis	5000	Minimum amount of time in milliseconds an object must sit idle in the pool before it is eligible for eviction.
orderDsMinEvictableIdleTimeMillis	5000	Minimum amount of time in milliseconds an object might sit idle in the pool.
orderDsNumTestsPerEvictionRun	5	Maximum number of connections to examine during each

Property Name	Property Value	Description
		evictor run.
orderDsDefaultAutoCommit	false	Enable auto commit after each transaction.

Common Database Configuration

Property Name	Property Value	Description
datasourceDriverClassName	org.postgresql.Driver	Data Source Driver Class Name.
datasourceValidationQuery	SELECT 1	SQL query that is used to validate connections.
hibernateDialect	org.hibernate.dialect.PostgreSQLDialect	Hibernate Dialect.
hibernateShowSql	false	Hibernate Show SQL.

EMS Configuration

The Broker stays connected to the EMS Server to listen for southbound replies, especially for instances that are inactive now and for replies lacking the required originator header.

Property Name	Property Value	Description
emsServerURL	tcp://localhost:7222	EMS Server URL

Property Name	Property Value	Description
emsServerUsername	admin	EMS Server User Name
emsServerPassword	admin	EMS Server Password
jndiConnectionFactory	GenericConnectionFactory	JNDI Connection Factory Name
tibjmsNamingSecurityProtocol		SSL configuration value, to be set to 'ssl'.
tibjmsNamingSslEnableVerifyHost		Set to 'false' to disable host verification in SSL.
initialContextFactory	com.tibco.tibjms.naming.TibjmsInitialContextFactory	Name of the Initial Context Factory.
jmsSessionTransacted	true	Determines if JMS sessions are transacted. Set to 'true'.
timeoutReceiveCalls	10000	Specifies the timeout for acknowledging EMS messages, in milliseconds.

Messaging Queues

The Broker stays connected to the EMS Server to listen for Southbound replies, especially for instances that are inactive now and for replies lacking the required originator header. Additionally, to recover requests sent to the Southbound system, the Broker requires proper queues configuration.

For more information, see the following topics:

- [Instance Status Management](#)

- [Instance Failure Mechanism](#)
- [Re-assigning Order to an Instance](#)
- [Listener for Southbound Reply With no Originator](#)

Property Name	Property Value	Description
orchestratorInboundQueue	tibco.aff.orchestrator.inbound.queue	The Broker service monitors the Orchestrator inbound queue as part of the Instance Failure Mechanism to manage messages in the event of an instance failure.

Exposed REST Endpoints

Instance Management APIs

The instance management APIs are as follows:

- [Registration of a new instance](#)
- [Unregistration of an existing instance](#)
- [Instance Ping Operation](#)
- [Instance clean up](#)
- [Get Instance Details By Instance Id](#)
- [Get Instance Details By Service Name](#)
- [Get Instance Details of all instances](#)

Registration of a new instance

URI: `v1/instance/register`

Request Body:


```
{
  "instanceId": "string",
  "serviceName": "string",
  "ipAddress": "string",
  "port": 0
}
```

Request Schema

Parameter	Description
serviceName	Service name of the instance. Currently only Orchestrator is supported.
ipAddress	IP address of the instance.
port	Port of the instance.
instanceId	Id of the instance.

Description

This API handles instance registration requests. After receiving a registration request, the Broker service uses a database sequence to create an `instanceId`. Subsequently, it performs validations before inserting a new record in the `instance_ledger` table.

Importantly, during registration, the Broker service does not immediately verify the instance's health status.

The initial status of the selected instance remains `Pending`. The instance status is updated to `Active` or `Inactive` based on the health assessment during the ping operation.

On successful registration, an Instance Status Change Notification is dispatched.

If a registration request is received for an already registered IP address and port, the Broker service returns the existing instance ID. This ensures that all pending messages on the EMS can be processed by the same instance that was restarted.

Request Validations and corresponding error responses

- IP Address cannot be null or empty string.

```
{
  "status": "BAD_REQUEST",
  "message": "Ip Address cannot be null or empty"
}
```

- Port cannot be less than or equal to zero

```
{
  "status": "BAD_REQUEST",
  "message": "Port cannot be less than or equals to Zero"
}
```

- Service Name cannot be null or empty string

```
{
  "status": "BAD_REQUEST",
  "message": "Service name cannot be null or empty"
}
```

- Service Name cannot be anything other than Orchestrator

```
{
  "status": "BAD_REQUEST",
  "message": "Invalid service name {serviceName}"
}
```

Response Body

```
{
  "instanceId": "string",
  "serviceName": "string",
  "ipAddress": "string",
  "port": int,
  "status": "string"
}
```

Response Schema

Parameter	Description
instanceId	A unique identification of the instance about to be registered.

Parameter	Description
serviceName	Service name of the instance. Currently only Orchestrator is supported.
ipAddress	IP address of the instance.
port	Port of the instance.
status	<ul style="list-style-type: none"> • Pending • Active • Pending Purge • Inactive

Responses

HTTP Status	Description
201	New Instance is registered successfully.
200	Instance with ipAddress and port already existed with inactive status. Updated the status to Pending.
202	Instance with ipAddress and port already existed with active status. No operation is performed by Broker.
401	Invalid token
400	<ul style="list-style-type: none"> • IPAddress cannot be null or empty • Port cannot be less than or equals to Zero • Service name cannot be null or empty • Invalid <<serviceName>> <p>In place of <<serviceName>>, the actual name of the service is displayed.</p>

Unregistration of an existing instance

URI: `v1/instance/un-register/{instance_id}`

Path Variable:

`instance_id` (Instance Id): A unique identification of the instance about to be unregistered. Accepted values can be any non null or non blank string.

Description

This asynchronous operation accepts the `instance_id` as a path variable and deregisters an instance from the Broker service. The deregistration process does not immediately remove the instance. The status of the instance is updated to `Inactive`.

Request Validations and corresponding error responses

- Instance Id cannot be null or empty string

```
{
  "status": "BAD_REQUEST",
  "message": "Instance Id cannot null or empty"
}
```

- Instance with id `{{instance_id}}` is not present

```
{
  "status": "BAD_REQUEST",
  "message": "Instance with id {{instance_id}} is not present"
}
```

Responses

HTTP Status	Description
202	Request accepted
401	Invalid token
400	Instance Id cannot null or empty
404	Instance with id <code>{{instance_id}}</code> is not present

Instance Ping Operation

URI: `v1/instance/ping`

Request Body

```
{
  "instanceId": "string",
  "status": "string",
  "inactiveComponentList": [
    "string"
  ]
}
```

Request Schema

Parameter	Description
instanceId	A unique identification of the instance
status	Health status of the instance. The following values are Allowed: <ul style="list-style-type: none"> • UP • DOWN • UNKNOWN • OUT_OF_SERVICE
inactiveComponentList	List of components within the instance that do not currently exhibit an active health status.

Description

This API allows instances to regularly update their health status by communicating with the Broker Service. Instances compute their health status periodically and send this data to the Broker Service using the REST API.

After receiving these updates, the Broker Service processes and updates the instance's status in the database. If an instance reports an Up status, it is labeled as Active; otherwise, it is marked as Inactive.

In cases where an instance is Inactive, the system triggers an Order Event Transfer task to EMS for further handling.

Request Validations and corresponding error responses

- Instance Id cannot be null or empty string

```
{
  "status": "BAD_REQUEST",
  "message": "Instance Id cannot null or empty"
}
```

- Instance with id {{instance_id}} is not present

```
{
  "status": "NOT_FOUND",
  "message": "Instance with id {{instance_id}} is not present"
}
```

- Status cannot be null or empty

```
{
  "status": "BAD_REQUEST",
  "message": "Status cannot be null or empty"
}
```

- Invalid status

```
{
  "status": "BAD_REQUEST",
  "message": "Invalid status. Only Up, Down, OUT_OF_SERVICE and UNKNOWN is allowed"
}
```

Responses

HTTP Status	Description
200	Request processed
401	Invalid token
400	<ul style="list-style-type: none"> • Instance Id cannot null or empty. • Status cannot be null or empty. • Invalid status. Only Up, Down, OUT_OF_SERVICE and

HTTP Status	Description
	UNKNOWN is allowed.
404	Instance with id {{instance_id}} is not present.

Instance clean up

URI: v1/instance/cleanup

Request Body: ["string"]

Request Schema: List of instance ids to purge from the database.

Description

In some scenarios, an instance might become inactive without being removed from the Broker database. You can use this API to manually specify a list of instance IDs that must be purged from the database.

If any instance in the provided list is not in an Inactive status, it is not purged. However, any other instances listed is removed from the database.

Request Validations and corresponding error responses

- Instance Id list cannot be empty

```
{
  "status": "BAD_REQUEST",
  "message": "Instance Id list cannot be empty"
}
```

- Are any of the given instances available with broker service?

```
{
  "status": "NOT_FOUND",
  "message": "None of requested instances are present"
}
```

- Are all the given instances still active?

```
{
  "status": "BAD_REQUEST",
```

```
"message": "All of the requested instances are still active.
Unregister them first."
}
```

Responses

HTTP Status	Description
200	All requested instances are purged from the database.
206	Only some of the requested instances are purged from the database. This would usually occur when some instances are still active.
401	Invalid token
400	<ul style="list-style-type: none"> Instance Id list cannot be empty All of the requested instances are still active. Unregister them first
404	None of request instances are present

Get Instance Details By Instance Id

URI: v1/instance/{instanceId}

Path Variable:

instance_id (Instance Id): An exclusive identifier associated with the instance for which details are being sought. The accepted values can be any non null or non blank string.

Description

This API accepts an instanceId as a path variable and returns the details of the instance that is registered with the broker service.

Request Validations and corresponding error responses

- Instance Id cannot be null or empty string


```
{
  "status": "BAD_REQUEST",
  "message": "Instance Id cannot be null or empty string"
}
```

- Instance with id <<instanceId>> is not present

```
{
  "status": "NOT_FOUND",
  "message": "Instance with id <<instanceId>> is not present"
}
```

Response Body

```
{
  "instanceId": "abc",
  "serviceName": "Orchestrator",
  "ipAddress": "0.0.0.0",
  "port": 9093,
  "status": "UP"
}
```

Response Schema

Parameter	Description
instanceId	A unique identification of the instance about to be registered.
serviceName	Service name of the instance. Currently only Orchestrator is supported.
ipAddress	IP address of the instance.
port	Port of the instance.
status	<ul style="list-style-type: none"> • Pending • Active • Pending Purge • Inactive

Responses

HTTP Status	Description
200	Returned the instance details of the requested instance.
401	Invalid token
400	Instance Id cannot be null or empty string.
404	Instance with id <<instanceId>> is not present.

Get Instance Details By Service Name

URI: /v1/instance?serviceName={serviceName}&pageNo={pageNo}&instancesPerPage={instancesPerPage}

Parameters	Description
serviceName (Service Name)	The service name for which details of every available instance are being requested. The accepted values can be any non-null or non-blank string.
pageNo (Page Number)	This parameter specifies the page number of the paginated data. The

Parameters	Description
	accepted values can be positive integers starting from 1, where 1 represents the first page.
instancesPerPage (Instances Per Page)	This parameter determines the number of instances displayed on a single page of paginated data. The accepted values can be positive integers indicating the desired number of instances per page.

Description

You can use this API endpoint to retrieve a paginated list of records from the system for a given `serviceName`. You can specify the service name, page number, and the number of records to display per page using query parameters.

Request Validations and corresponding error responses

- Service name cannot be null or empty string

```
{
  "status": "BAD_REQUEST",
  "message": "Service name cannot be null or empty"
}
```

- No instance is available for the given service name

```
{
  "status": "NOT_FOUND",
  "message": "No instances found for service name {serviceName}"
}
```

- Invalid Page number. Page number cannot be less than 1.

```
{
  "status": "BAD_REQUEST",
  "message": "PageNo cannot be less than 1"
}
```

- Invalid instances per page. InstancesPerPage must be between 1 and 20.

```
{
  "status": "BAD_REQUEST",
  "message": "InstancesPerPage should be between 1 and 20."
}
```

Response Body

```
{
  "instanceDetailsList": [
    {
      "instanceId": "abc",
      "serviceName": "Orchestrator",
      "ipAddress": "0.0.0.0",
      "port": 9093,
      "status": "UP"
    }
  ],
  "totalPages": 1,
  "currentPage": 1,
  "instancesPerPage": 10
}
```

Response Schema

Parameter	Description
instanceDetailsList	List of the instance details
totalPages	Total number of pages available as per instancesPerPage
currentPage	Current Page number
instancesPerPage	Instances per page requested by user

Instance Details

Parameter	Description
instanceId	A unique identification of the instance about to be registered.
serviceName	Service name of the instance. Currently only Orchestrator is supported.
ipAddress	IP address of the instance
port	Port of the instance
status	<ul style="list-style-type: none"> • Pending • Active • Pending Purge • Inactive

Responses

HTTP Status	Description
200	Returns all instance details for all the instances registered with requested service name.
401	Invalid token

HTTP Status	Description
400	<ul style="list-style-type: none"> Service name cannot be null or empty PageNo cannot be less than 1 InstancesPerPage must be between 1 and 20
404	No instances found for service name {serviceName}

Get Instance Details of all instances

URI: v1/instance/all?pageNo={pageNo}&instancesPerPage={instancesPerPage}

Parameters	Description
pageNo (Page Number)	This parameter specifies the page number of the paginated data. The accepted values can be positive integers starting from 1, where 1 represents the first page.
instancesPerPage (Instances Per Page)	This parameter determines the number of instances displayed on a single page of

Parameters	Description
	paginated data. The accepted values can be positive integers indicating the desired number of instances per page.

Description

You can use this API endpoint to retrieve a paginated list of records from the system. Also, you can specify the page number and the number of records to display per page using query parameters.

Request Validations and corresponding error responses

- Invalid Page number. Page number cannot be less than 1

```
{
  "status": "BAD_REQUEST",
  "message": "PageNo cannot be less than 1"
}
```

- Invalid instances per page. InstancesPerPage must be between 1 and 20.

```
{
  "status": "BAD_REQUEST",
  "message": "InstancesPerPage should be between 1 and 20."
}
```

Response Body

```
{
  "instanceDetailsList": [
    {
      "instanceId": "abc",
```

```

        "serviceName": "Orchestrator",
        "ipAddress": "0.0.0.0",
        "port": 9093,
        "status": "UP"
    },
    "totalPages": 1,
    "currentPage": 1,
    "instancesPerPage": 10
}

```

Response Schema

Parameter	Description
instanceDetailsList	List of the instance details
totalPages	Total number of pages available as per instancesPerPage.
currentPage	Current Page number
instancesPerPage	Instances per page requested by user

Instance Details

Parameter	Description
instanceId	A unique identification of the instance about to be registered.
serviceName	Service name of the instance. Currently only Orchestrator is supported.
ipAddress	IP address of the instance
port	Port of the instance
status	<ul style="list-style-type: none"> • Pending • Active • Pending Purge • Inactive

Responses

HTTP Status	Description
200	Returned all instance details for all the instances registered with requested service name.
401	Invalid token
400	<ul style="list-style-type: none"> • pageNo cannot be less than 1 • InstancesPerPage must be between 1 and 20
404	No instances available

Get all services

URI: /v1/instance/services

Description

This API provides a distinct list of service names for which instances are currently available within the Broker service.

Request Validations and corresponding error responses

- No instance is available

```
{
  "status": "NOT_FOUND",
  "message": "No instance exist."
}
```

Response Body: ["Orchestrator"]

Response Schema: List of distinct service names.

Responses

HTTP Status	Description
200	Returned all distinct service names.

HTTP Status	Description
401	Invalid token
404	No instance is currently available with broker service.

Recovery APIs

The following recovery APIs are designed to re-trigger requests for pending tasks.

- /v1/notifications/pending
- /v1/notifications/re-send
- /v1/notifications/re-send/all

For more information about recovery APIs, see the *TIBCO® Order Management Web Services Guide*.

Alternatively, re-trigger functions can be performed through the OMS UI. For more information, see [Pending Tasks](#).

Instance Status Management

ActiveInstanceMonitor

In scenarios where an active instance experiences an unexpected shutdown before updating its health status with the Broker service, the database might retain the instance's status as "Active."

The primary goal of the ActiveInstanceMonitor is to detect such instances that might be inactive due to an unforeseen disruption. This monitor initiates the transfer of any pending southbound replies from the affected instance to other operational instances, ensuring the continuity of service and preventing interruptions in order processing.

Configuration Used

Property Name	Purpose
activeInstanceMonitorInterval	Time interval, measured in minutes, at which ActiveInstanceMonitor operates or runs.

Property Name	Purpose
<code>activeInstanceMonitorThreshold</code>	The duration parameter used by <code>ActiveInstanceMonitor</code> to identify and declare inactive instances based on the timestamp comparison (<code>LastUpdatedTimestamp</code>) against the defined threshold.

The values of `activeInstanceMonitorInterval` and `activeInstanceMonitorThreshold` are set in consideration of the `brokerPingInterval` (available in `ConfigValues_Orchestrator`, used by Orchestrator). These values play a role in determining how often the `ActiveInstanceMonitor` evaluates the last updated timestamp and subsequently marks instances as inactive.

`activeInstanceMonitorInterval` is chosen to ensure that the `ActiveInstanceMonitor` runs frequently enough to promptly identify instances that might have missed their health status updates. It must be set at a frequency that allows for timely checks without being overly burdensome on the system.

`activeInstanceMonitorThreshold` must be configured in relation to the `brokerPingInterval`. The threshold value defines the maximum time allowed before considering an instance as inactive. It's essential to set it considering the expected frequency of instance health status pings (`brokerPingInterval`). The threshold must ideally be longer than the `brokerPingInterval` to allow for a margin of error due to delays or missed pings.

For example, if `brokerPingInterval` is set at 5 minutes, it might be reasonable to configure `activeInstanceMonitorInterval` slightly longer than that, such as 7-10 minutes, and then set the `activeInstanceMonitorThreshold` to a value greater than the `brokerPingInterval`, for instance, at 15-20 minutes. This setup allows the `ActiveInstanceMonitor` to run more frequently than the pings to ensure quick detection of inactive instances without marking instances inactive due to minor communication delays.

Behavior

- The `ActiveInstanceMonitor` operates as a time-based scheduler, executing at intervals set by `activeInstanceMonitorInterval`.
- It calculates the ``lastUpdatedTimestampThreshold`` using the formula `System.currentTimeMillis() - (long) activeInstanceMonitorThreshold`.
- With this threshold, the monitor identifies instances that haven't been updated

within the specified duration—specifically, instances whose `lastUpdatedTimestamp` is less than or equal to the `lastUpdatedTimestampThreshold`. Instances that meet this criterion are marked as inactive, indicating potential disruptions or a failure to report their health status.

- For each instance that meets this criterion, the monitor sets the instance status to `Inactive` in the database and triggers the dispatch of an `OrderEventTransferTask` to EMS.

InactiveInstanceMonitor

When an instance becomes inactive, there might be situations where southbound systems send replies for orders previously processed by that instance. Since the inactive instance is no longer operational, these messages accumulate on EMS queues, affecting order processing.

The primary goal of the `InactiveInstanceMonitor` is to relocate all such pending messages to an active instance. This action ensures that order processing continues uninterrupted, despite the inactivity of some instances.

Configuration Used

Property Name	Purpose
<code>inactiveInstanceMonitorCronExpression</code>	A Cron expression that determines the execution schedule of <code>InactiveInstanceMonitor</code> .

Behavior

- This monitor fetches instances that are in inactive status.
- For all such instances, `OrderEventTransferTask` is dispatched to EMS.

Instance Failure Mechanism

OrderEventTransferMonitor

The `OrderEventTransferMonitor` is designed to transfer all pending events in the Event Management System (EMS) queues associated with instances that are not in an active status. This is achieved by creating a separate thread for each queue. Additionally, it updates the originator (specified in the Java Message Service (JMS) header) for all messages present at that specific time.

The following queues are affected:

- tibco.aff.orchestrator.planItem.execute.reply
- tibco.aff.orchestrator.planItem.suspend.reply
- tibco.aff.orchestrator.provider.planItem.failed.reply
- tibco.aff.orchestrator.provider.order.opd.reply
- tibco.aff.orchestrator.planItem.milestone.notify.request
- tibco.aff.orchestrator.provider.order.prequal.failed.reply
- tibco.aff.orchestrator.provider.order.feasibility.reply

Input

```
{
  "instanceId": "",
  "ipAddress": "",
  "port": ""
}
```

Input Schema

Parameter	Description
instanceId	An exclusive identifier associated with the instance
ipAddress	IP address of the instance
port	Port of the instance

Configuration Used

Property Name	Purpose
emsServerUrl	EMS Server URL
emsServerUsername	EMS Server User Name
emsServerPassword	EMS Server Password
timeoutReceiveCalls	Ems message receive timeout in milliseconds

Property Name	Purpose
orderEventTransferTaskQueue	Queue on OrderEventTransfer task would be published internally by Broker Service
orderEventTransferTaskListenerCount	Number of concurrent listener on orderEventTransferTaskQueue

Behavior

- Order event initiation occurs under the following circumstances:
 - An active node has not updated its status within a specified duration.
 - An instance remains in an inactive status.
- Verification of instance availability is performed with the Broker for a given `instanceId`. This task is skipped if the instance is not found.
- If the instance status is not Active, the order event transfer process is initiated for all relevant queues.
- Individual threads are created for each relevant queue.
- These threads establish connections with the EMS Server and begin monitoring the designated queue.
- After identifying a message, the thread reassigns the order to a new instance. The Broker updates the existing JMS Header value named `Originator` with the new value = new `InstanceId`. For more information, see the [Re-assigning Order to an Instance](#) section.
- The Broker updates the `instanceId` associated with the order in the `Order_data` table.
- The message, now carrying the updated `Originator` header, is dispatched back to the same queue for further processing.

Re-assigning Order to an Instance

The primary objective of this task is to reallocate orders previously processed by an inactive instance to an instance currently marked as Active.

Input

Parameter	Description
orderId	An exclusive identifier associated with the order.
currentInstanceId	An exclusive identifier associated with the instance that was processing this order and is currently not in Active state.

Behavior

- Fetch the `instanceId` assigned to a given `orderId`.
- If the `instanceId` matches the `currentInstanceId`, reassign the order to a new instance.
- If the `instanceId` does not match the `currentInstanceId`, it implies that the order might already be assigned to another new instance. Subsequently, the task checks the status of this new instance. If the new instance is found to be Inactive, the broker reassigns the order to an alternate new instance.

Logic to compute new InstanceId

- Retrieve all active instances available within the Broker Service.
- It assesses the workload managed by each instance using the following query.

```
select node_id, count(*)
from plan_item_data pid
      inner join order_data od on pid.orderid = od.orderid
where od.node_id is not null
      and od.node_id in :NODE_ID_LIST
      and od.status NOT in ('COMPLETE', 'CANCELLED', 'WITHDRAWN')
      and pid.status not in ('COMPLETE', 'CANCELLED')
group by node_id
```

- The order is assigned to the instance handling the least workload among the available active instances.

Message routing On EMS

Configuration Used

Property Name	Description	Default Value
orchestratorInboundQueue	Queue on which the Orchestrator listens for all southbound replies that contain an originator header.	tibco.aff.orchestrator.inbound.queue
orchestratorInboundNoOriginator Queue	Queue on which the Broker Service listens for all southbound replies that do not contain an originator header.	tibco.aff.orchestrator.inbound.no.originator.queue

Bridges at the EMS level are introduced to optimize message handling between the southbound service and the Orchestrator. These bridges facilitate the routing of replies based on the presence of an `originator` header in the message.

Southbound Message Routing

When the southbound service sends a reply to the reply queue, the message is bridged from the source queue to a specific target queue based on a selector.

- **With Originator Header:** If the southbound reply contains the originator header, the message is redirected to the `orchestratorInboundQueue`.
- **Without Originator Header:** If the southbound reply does not contain the originator

header, the message is redirected to the `orchestratorInboundNoOriginatorQueue`.

When the southbound service sends a Plan Item Execute Reply to the `tibco.aff.orchestrator.planItem.execute.reply` queue, the EMS routes this message to either `orchestratorInboundQueue` or `orchestratorInboundNoOriginatorQueue` based on the presence of the originator header in the message.

System Adaptation and Customer Impact

This bridging mechanism ensures that the system adapts to these changes without requiring any modifications on the customer's end. The orchestration and broker services handle the messages as follows:

- Orchestrator Service: Listens for messages on `orchestratorInboundQueue`.
- Broker Service: Listens for messages on `orchestratorInboundNoOriginatorQueue`.

Advantages of the New Approach

This approach provides significant advantages over a model where microservices have a fixed number of listeners on each queue:

- Thread Utilization: Fixed listeners on multiple queues can lead to blocked threads when there are no messages, wasting resources.
- Load Balancing: With the new approach, a fixed number of threads on a single queue process all incoming messages, ensuring efficient load balancing and optimal resource utilization.

By consolidating the message processing to a single queue with intelligent routing based on message headers, the service maintains high efficiency and flexibility in handling southbound messages. This design minimizes thread blocking and enhances the system's ability to manage varying message loads effectively.

Listener for Southbound Reply With no Originator

When Orchestrator dispatches requests to Southbound systems, it includes its InstanceId in a header named 'Originator'.

This setup is designed for the Southbound systems to respond with a message containing the same 'Originator' header. Each Orchestrator instance is configured to listen solely to messages associated with its InstanceId, using a specific selector, 'Originator=<<InstanceId>>'.

However, there are scenarios where a Southbound system replies without this expected header. In such cases, the Broker service intervenes by monitoring messages lacking the

'Originator' header using a selector, 'Originator IS NULL'. The Broker service then directs these messages to the instance processing the order.

If this instance is inactive in the Broker database, the Broker service assigns a new instance for handling this order. For more information, see the [Re-assigning Order to an Instance](#) section.

Configuration Used

Property Name	Purpose
orchestratorInboundNoOriginatorQueue	Queue from which the broker service listens for messages that do not contain an originator header.
orchestratorInboundNoOriginatorReceiverCount	Number of concurrent consumers on No originator inbound queue
orchestratorInboundNoOriginatorRetryCount	Number of retries in case of any exception
orchestratorInboundNoOriginatorRetryInterval	Delay in Milliseconds between consecutive retries
orchestratorInboundNoOriginatorDeadQueue	Dead queue to redirect the failure messages
orchestratorInboundQueue	Queue from which the Orchestrator listens for Southbound messages. The broker service redirects messages to this queue for processing by the Orchestrator.

Behavior

- The Broker service operates by monitoring individual southbound response queues.
- After receiving a message, it extracts specific order identifiers such as orderId, orderRef, and planId (optionally), to search for the corresponding order in the order database.
- When the order is located in the database, the Broker service computes the instance

id for that particular order, following the process detailed in the "Re-assigning order to an instance" section.

- The Broker service forwards the message back onto the same queue, adding a header named 'Originator' with the value set as `instanceId`.

Error Handling

In the event of an exception during message processing, the system initiates a retry mechanism. The process is retried for a specified count of attempts, defined as `noOriginatorRetryCount`, with a delay between each attempt, set as `noOriginatorRetryInterval`.

Recovery Notification Processing

Configuration Used

Property Name	Purpose
<code>recoveryNotificationQueue</code>	The queue to listen to the recovery notification sent by Orchestrator.
<code>recoveryNotificationDeadQueue</code>	Dead Queue to send the unprocessable recovery notification.
<code>recoveryNotificationReceiverCount</code>	Number of listener to listen to the recovery notification.
<code>recoveryNotificationRetryCount</code>	Number of retries in case of failure.
<code>recoveryNotificationRetryInterval</code>	Interval in milliseconds between each retry.

The recovery notifications are managed by the Broker service as follows:

- **Notification Dispatch:** The Orchestrator dispatches a notification to the queue specified by the `recoveryNotificationQueue` property.
- **Broker Service Responsibility:** The Broker service listens to these notifications and save them in the notification table, tracking the status and ensuring proper handling of Southbound requests and replies.

Time Dependency Monitor

The purpose of the TimeDependencyMonitor class is to ensure the consistency and correctness of time dependencies associated with orders in the system. It performs periodic checks to address orphan time dependencies, which are those with null instance IDs, assigning them to active owners based on predefined rules.

Configuration Used

Property Name	Purpose
timeDependencyMonitorInterval	Fixed interval in minutes to monitor time dependency with null instance id.

Behavior

- This job fetches all the orphan time dependencies from the database and groups them by order ID.
- For each order ID, it checks if the order has an owner assigned to it.
 - If an order has a valid instance ID (for example, the instance is active), the scheduler updates the time dependency with that instance ID.
 - If an order's instance is inactive or does not exist, the scheduler computes a new instance ID and assigns it to the time dependency and order.

Automated Order Plan Development

This section describes the functions of the Automated Order Plan Development feature in TIBCO Order Management.

Overview

Basic *Automated Order Plan Development* is the capability to create custom order plans that fulfill an order, taking into account the specifications of the required products and the products currently provided to a customer.

A *Product Model* contains bundles and products services. A product model also contains concepts such as sequencing and dependencies.

When an order is received, the order lines are decomposed by using a product model. The product specification for each order line is extracted from a product catalog by the decomposition component.

The product specification is required to create execution plan fragments. These execution plan fragments define the services, products, and resources required. For example, an order line might contain a bundle, which might be composed of several products and services. Taking into consideration factors such as sequencing and dependencies, these execution plan fragments are then combined to create a single execution plan.

Model Deployment

A catalog service loads the product model and the product model is stored in the database. During plan generation, Automated Order Plan Development receives models from the catalog database. For more details about model loading, see the "Catalog Services API samples" section in *TIBCO® Order Management Web Services Guide*.

Product Models Purging

Method: HTTP DELETE method

Endpoint: `http://<host_address>:<port_address>/v1/productmodel/bulk`

Parameter content type: application/json

The screenshot shows a web-based API configuration tool for the endpoint `/v1/productmodel/bulk`. The title bar indicates the method is **DELETE**. The main heading is "Delete Bulk Product Model".

Under the **Parameters** tab, there is a table with two columns: "Name" and "Description". A single parameter is listed:

Name	Description
purgeAllModels <small>* required</small> boolean (query)	-- true false

A "Cancel" button is located to the right of the parameters table.

Below the parameters is the **Request body** section, marked as *required*. A dropdown menu shows `application/json`. The request body area contains a sample JSON:

```
[
  {
    "productID": "string",
    "productIDExt": "string"
  }
]
```

A large blue **Execute** button is positioned at the bottom of the configuration panel.

- When you select the **purgeAllModels** field value as true and click **Execute**, all the product models are purged from the database. Here the request body is ignored entirely.
- When you select the **purgeAllModels** field value as false and click **Execute**, the request body is considered in this case. You can enter the values of `productID` and/or `productIDExt` in the request body as required. For more details on Product Id and Product Id Ext, see [Product Id and Product Id Ext](#).

Configuration

Automated Order Plan Development configuration is stored in files in the directory `$OM_HOME/seed-data/app-properties`. These files can be either manually edited or accessed from Configurator.

Main Configuration

The main Automated Order Plan Development configuration is stored at \$OM_HOME/seed-data/app-properties.

Parameter Name	Description
disableParentItemsDependencyImpact	DisableParentItemsDependencyImpact flag is used to ignore modification rule application on child plan item In case of parent plan item modification, if the flag is set to true. (default: false)
affinityudfnamemerge	Controls the flag to merge user-defined field name in the affinity plan item.
characterisitcswithoutaffinitypostfix	To not merge certain User Defined Fields during Affinity Sequencing, those User Defined Fields must be added as CSV in the variable (default: "")
skipitemsequence	Within Automated Order Plan Development, if the sequence is -1, it skips the product and all its mandatory children in the Execution Plan (default: -1)
mergeaffinityitemdescription	Merges affinity item description (default: false)
hierarchysingleuse	Uses unconditional removal of child product (default: false)
enableaffinityudfparent	Enables affinity user-defined field parent (default: false)
udflist	List of internal User Defined Fields to be skipped for affinity merging (default: "EPMR_ACTION_PROVIDE, EPMR_ACTION_UPDATE, EPMR_ACTION_CEASE, EPMR_ACTION_WITHDRAW, COMPENSATE_PROVIDE, COMPENSATE_UPDATE, COMPENSATE_CEASE")

Parameter Name	Description
enablebidirectionallinkid	Enables extended behavior for PDO/MDO and LinkID mapping (default: false)
allowmultiplerequiredproducts	Multiple Required Products for the same link ID are available (default: false)
ignorepdofirstchilddependency	Ignores First child dependency for source product in ProductDependsOn relationship (default: false)
handleconflict	Configurable handling for ProductComprisedOf conflict (default: "Apply")
ABDProductOrderline	Include only the order line for attribute-based decomposition (default: false)
ABDIncludeCharacteristics	Include plan item User Defined Fields for evaluating attribute-based decomposition (default: false)
compensateRestartForNoEPMRChar	Enables COMPENSATE and RESTART behavior in case of the required Execution Plan Modification Rules characteristic that is not present in the product model.
dateshiftcompredo	Enables the backward compatibility for Date Shift amendment to generate comp redo tasks (default: false)
enablemodificationidentifyingattribute	Enables the backward compatibility for user defined field change amendment by using MODIFICATION_IDENTIFYING_ATTR (default: false)
noDependencyInCOMPPlanItems	Enables the backward compatibility of NO dependency in the COMPENSATE plan item on the existing plan item being canceled.
enableparentidudfcheck	The Parent_ID user-defined field check, which was

Parameter Name	Description
	not present in plans generated in earlier versions of TIBCO Order Management, caused issues in amendment for later versions (ones with a user-defined field). A flag <code>com.tibco.af.aopd.flags.enableparentidudfcheck</code> , is introduced, which toggles checking this user-defined field during amendments. By default, the value is <code>true</code> , which enables checking of this user-defined field. Set the flag to <code>false</code> to disable the check.
<code>handlepcocircularDependency</code>	Handles circular dependency during plan generation. In case of <code>Ignore</code> value, it ignores circular dependency during plan generation. (default: <code>error</code>)
<code>disableAffinityBrokeUDFImpact</code>	Disable UDF amendment when affinity breaks due to an order line cancellation. This flag works with <code>disableParentItemsDependencyImpact</code> flag.

Logs

The Automated Order Plan Development logs are at `OM_HOME/seed-data/config-files`.

Features

Autoprovision

Autoprovision is a condition that determines the relationship between a parent product and a child product. The relationship can either be *Static* or *Dynamic*. Autoprovision flag determines whether the product is mandatory or not. For instance, consider a product A having a child product B, which has `AutoProvision` set as `TRUE`.

Static: If Autoprovision is set to TRUE, then the product is considered to be a static bundle and the child is automatically assumed to be a part of the order. This indicates that the child product is implicitly assumed to be on the order no matter there is an order line with that product.

Example: Consider bundle B comprises of products P1 and P2.

- AUTOPROVISION flag is TRUE between B and P1.
- AUTOPROVISION flag is FALSE between B and P2.

When bundle B is ordered, P1 is provisioned automatically though it is not in the order line.

i Note: This process of order fulfillment is known as *Implicit Fulfillment*.

Dynamic: Auto Provision is set to FALSE. This indicates that the child product must be explicitly included in the order.

The [product diagram](#) shows the mandatory products with solid lines (Autoprovision TRUE, Static bundle) and are included in the order automatically. Otherwise, the products have to be ordered separately, which is a Dynamic bundle feature.

i Note: The instanceOptional field is not used during plan generation in Automated Order Plan Development.

Dynamic Bundles

Dynamic Bundles lets for a bundle to be modeled by using a product hierarchy in a product catalog and items are selected by the user and then submitted for order plan development. An example is where a bundle is modeled to have mandatory items and optional items and the customer needs to select the options.

The *optional* products are specified as specific order lines within the order. The bundle is also specified as an order line but the decomposition component recognizes the options belonging to the parent bundle.

The *mandatory* products are automatically added for order planning.

Any *required* products are validated as part of the validation to ensure the basket or the customer image has the required product before any decomposition occurs.

It is possible to reuse the common products having Autoprovision=false using LinkParentID and LinkedParentID User-Defined Fields in the order line. The LinkParentID-LinkedParentID and LinkID User-Defined Fields are used to define PCO-tree, although the LinkParentID-LinkedParentID user-defined field has the higher priority.

- Link child to parent based on LinkParentID-LinkedParentID if present. Else,
- Link child to parent based on LinkID if present. Else,
- Link child to parent randomly.

For example, consider the following product model:

```
T-COM Wireline --> (PCO) Additional Voice Service(autoprovision=false)
T-COM Wireline --> (PCO) Tariff1(autoprovision=false)
Additional Voice Service --> (PCO) Tariff1(autoprovision=false)
```

Therefore, the order can be:

Orderline Number	Product	LinkParentID	LinkedParentID
1	T-COM Wireline	T-COM Wireline	
2	Additional Voice Service	Additional Voice Service	T-Com Wireline
3	Tariff1	Tariff1	Additional Voice Service
4	Tariff1	Tariff1	T-Com Wireline

The LinkParentID-LinkedParentID User-Defined Fields are added for order line number 3 in the following format to add dependency between OrderLine 2 and OrderLine 3:

```
"udf" : [ {
  "name" : "LinkParentID",
  "value" : "Tariff1"
}, {
  "name" : "LinkedParentID",
  "value" : "Additional Voice Service"
} ]
```

i Note: The value of LinkParentID, LinkedParentID, or LinkID can be anything. But the values must be the same between the child and parent product, which you need to link.

i Note: This change is backward-compatible. To use the LinkID functionality, do not add the LinkParentID-LinkedParentID User-Defined Fields in the order line.

Static Bundles

Through Static Bundles, a bundle to be modeled by using a product hierarchy in the catalog, with the bundle only containing mandatory options. An example of this is a bundle with mandatory products and when a customer orders this bundle all the dependent products are provisioned without the customer needing to select any more products.

Time Dependency

The decomposition component has the ability to provision an execution plan for a given order based on the time constraints, if any, placed on the products within that order. Example: it determines when a particular product execution must be started. This time constraint can apply to an individual plan item within an execution plan or to the entire execution plan. Time dependency is added in each plan item if the requiredByDate specified in order is in the future.

Time dependency defines the absolute time when the particular plan fragment starts execution. It is calculated on the basis of the requiredByDate present in either the Order header or the OrderLine. The expected behavior for the required by date is as follows.

1. If requiredByDate is set on the order level, the start time dependency applies to all plan items with no leading dependencies
2. If requiredByDate is set on the order line level only, the start time dependency applies to plan items for that order line, which have no leading dependency
3. If requiredByDate is set on the order header level and on the order line level, the following behavior applies:
 - a. If requiredByDate in Order Header is later than requiredByDate in the line item,

then the start time used is the one at order level

- b. If requiredByDate in Order Header is earlier than requiredByDate in line item, then the start time used is the one at order line level.



Note: RequiredOnDate is no longer used or supported.

Product Specification Field Decomposition

Each product has a modeled set of characteristics within a product catalog. When a product is decomposed to a plan item, the default and the instance characteristics are copied over into the User-Defined Fields (UDFs) of every plan item. Through this, the information is reused later when the plan item is run.

For example, consider a product "Line Access 5MB" has characteristics modeled such as Speed=5, QOS=4, IPAccess=false. These are all modeled as instance variables. When an order is submitted for Line Access or is part of a bundle, the plan item uses the same instance characteristics copied as User-Defined Fields into the plan item. When the plan item is run, the User-Defined Fields can be passed to the service call.

When an order is made the characteristics are visible as user-defined fields for each order line. When you submit the order, the user-defined fields are converted into user-defined fields for the new plan items and if the order line is a bundle then those items can have user-defined fields as well, which are copied to the execution plan. All these user-defined fields can be used later through the service call.

Custom Action Based Product Decomposition

The custom action provides flexible way to define products and product fulfillment by allowing product decomposition and characteristic list inclusion. The ProductComprisedOf (henceforth, referred to as PCO) relationship enables you to model complex product hierarchies. This allows a product modeler to model specific product decomposition according to the specified action.



Note: Irrespective of an action, all the PCO or Characteristic relationships are valid.

The following table describes the custom action for the PCO and Characteristic relationships:

ProductComprisedOf (PCO)		Characteristic (C)	
If PCO.ActionID=null	The child product is always a part of the decomposition during decomposition	If C.ActionID=null	The characteristic is always included as planItem User Defined Fields
If PCO.ActionID=not null	The child product is only added if the following order action is specified during decomposition: order Action = the ActionID	If C.ActionID=not null	The characteristic is included if the following order action is specified during decomposition: order Action = ActionID

Scenario for the Custom Action Based Product Decomposition

The following table describes how a custom action impacts the product decomposition:

i Note: This scenario is applicable for characteristic list inclusion based on custom action.

Data Model Configuration	Order	Plan
Action repository has record with ID as HomeMove and recordtype as PROVIDE. Product B has PCO relationship with P1, P2, P3 with autoprovision=true P1.PCO.ActionID=null P2.PCO.ActionID=PROVIDE P3.PCO.ActionID=HomeMove	OL=B(PROVIDE)	There are three planItems: <ul style="list-style-type: none"> • B • P1 • P2 B depends

Data Model Configuration	Order	Plan
		on P1 & P2
	OL=B(UPDATE)	There are two planItems: <ul style="list-style-type: none"> • B • P1 B depends on P1
	OL=B (HomeMove)	There are two planItems: <ul style="list-style-type: none"> • B • P3 B depends on P3

Sequencing

The product catalog defines the sequencing requirements between the fulfillment steps for products in a product offering.

When the order plan is being developed, the information in the product catalog is used such that the instance sequence defined for each subproduct and products, which contain these subproducts translates to a dependency between *Plan Fragments* associated with each product/sub-product and the fulfillment happens in the correct sequence.

Sequencing is governed by certain rules on the Plan Fragments.

Order Management supports the action-based sequencing. This use case-based sequence of back-end systems is used in the decomposition to ensure back-end (process component) are called in the correct order. Based on the order line action, the following types of sequencing are used:

- PROVIDE
- CEASE
- UPDATE

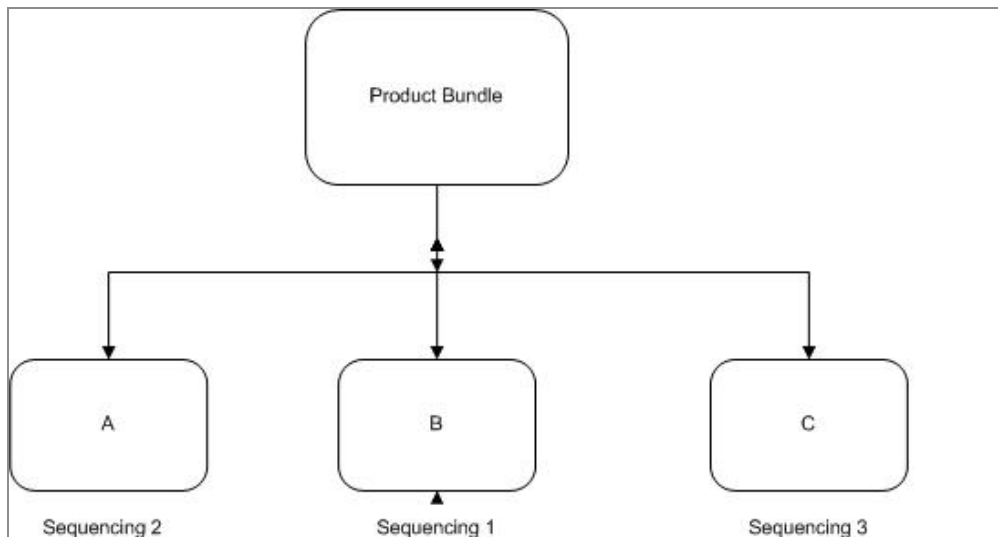
PROVIDE Sequencing: This scenario occurs when the order line action is PROVIDE and all the subproducts use the provided instance sequence number.

CEASE Sequencing: This scenario occurs when the order line action is CEASE and all the subproducts use the cease instance sequence number.

UPDATE Sequencing: This scenario occurs when the order line action is UPDATE and all the subproducts use the update instance sequence number.

The following figure shows the sequencing for the products A, B, and C.

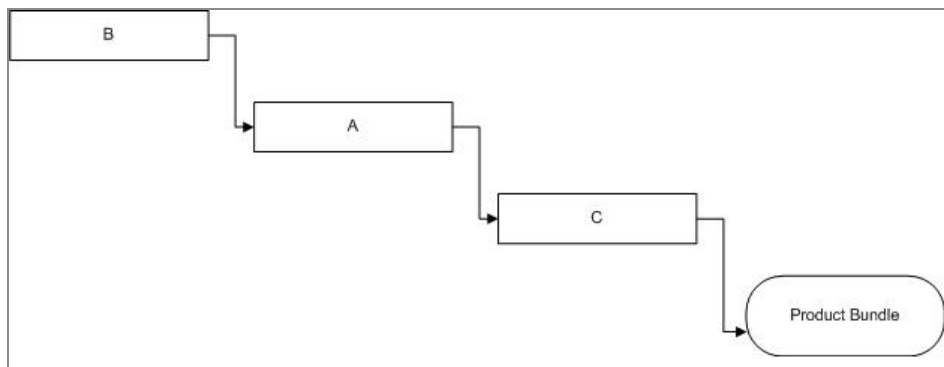
Sequencing for the products A, B, and C.



i Note: Sequence number is the relationship attribute value based on the actions PROVIDE, CEASE, and UPDATE.

For example, a bundle is composed of Product A, Product B, and Product C, with PROVIDE sequencing set to 2, 1 and 3 respectively. When an order plan is developed, Product B is run first, followed by Product A and then Product C.

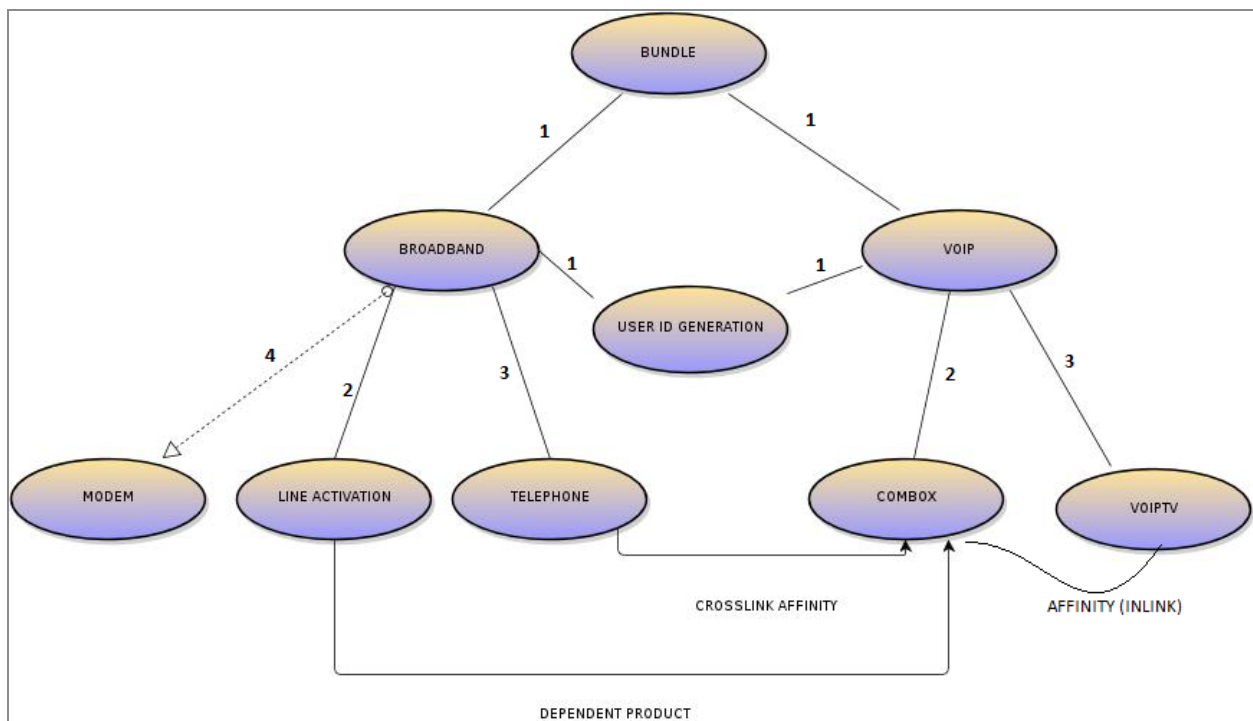
Order Plan Execution Sequence




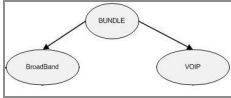
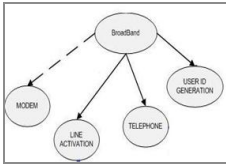
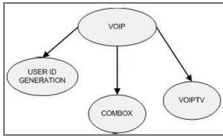
Similarly, a CEASE sequencing order can also be defined for the same Product Bundle with a sequencing of 3, 1, and 2 for products A, B, and C respectively. In this manner, the order might be fulfilled in the correct sequence taking into account what action needs to be performed.

The Order lines are converted into plan items during the plan development by using the information in the product catalog. The diagram explains the sample product model and its components (product offerings). This diagram is used to briefly explain the different Plan Development Concepts (for details, see [Automated Order Plan Development](#)).

Sample Product Model



The table describes the diagram elements of the Product Model Hierarchy.

Diagram Element	Description
	Product entity.
	<p>The arrows represent the <i>ProductComprisedOf</i> relationship in the product catalog between BUNDLE and a group of products. Thus, the diagrams state that:</p> <ul style="list-style-type: none"> • BUNDLE is composed of the product Broadband. • BUNDLE is composed of the product VoIP.
	<p>The Broadband product offering contains the following mandatory products:</p> <ul style="list-style-type: none"> • Telephone • UserID • Line Activation <p>The dotted line indicates that the Modem is an optional product.</p>
	<p>The VoIP product offering contains the following mandatory products:</p> <ul style="list-style-type: none"> • VOIPTV • COMBOX • UserID <p>The "UserID" product has two parents.</p>

Product Model Description:

The product BUNDLE is composed of the two product offerings:

- Broadband.
- VoIP.

The Broadband product offering contains the products as the Telephone, UserID, and Line Activation as the mandatory product. Modem is an optional product.

VoIP has the COMBOX, UserID, and VOIPTV as part of its technical products.

The product UserID here has two parents - Broadband and VoIP. The product UserID has the single use record attribute set to true with both its parents.

Using (relationship) sequencing, all the child products of the BUNDLE are fulfilled or processed in parallel, and all must complete before the entire BUNDLE can be fulfilled. Often, additional sequencing is required within elements at the same hierarchy level in the model. This can be accomplished by providing sequence numbers on the *ProductComprisedOf* relationship.

Product Model Description with the Sequencing Feature:

The correct fulfillment sequencing of the product plan execution as per the diagram is:

1. The UserID is created.
2. The order on the Telephone and COMBOX is processed. The Telephone and COMBOX are installed.
3. The Line Activation is completed.
4. Modem is installed.
5. VOIPTV is installed.
6. Broadband Product Order and VoIP Order are completed.
7. The entire BUNDLE order (Broadband and VoIP) is completed.

Taking the product as an example, the table shows the sequencing of the products:

Parent Product	Product Offering
BUNDLE	UserID
BUNDLE	Telephone/COMBOX
BUNDLE	Line Activation
BUNDLE	Modem Installation (optional)
BUNDLE	VoIP
BUNDLE	BUNDLE Order complete

Delta Provisioning

Delta Provisioning ensures that products, which have been defined for 'single use' are not provisioned more than once for a given order. The combination of the order line action of the products is used to determine how the products are provisioned.

Single Use

Single Use ensures that if the products have the same product ID and have been defined for single use with the order line actions as PROVIDE then those products are not provisioned more than once. It deletes one of the instances and ensures that the dependencies point to the single instance, which remains in the plan. This is done for products with the same parent only.

For example, only one shipment needs to be typically sent for a batch of phones.

Product Model description in relation to Single Use:

The [Product Model diagram](#) shows the Single Use feature. If the Order is a 'BUNDLE in a single Order line', the UserID is generated only once, although it has been ordered twice by the products Broadband and VoIP respectively.

If the product exists more than once on the order, then it is only included once in the final plan. If the product exists on the order and in the inventory, it is not included in the plan.

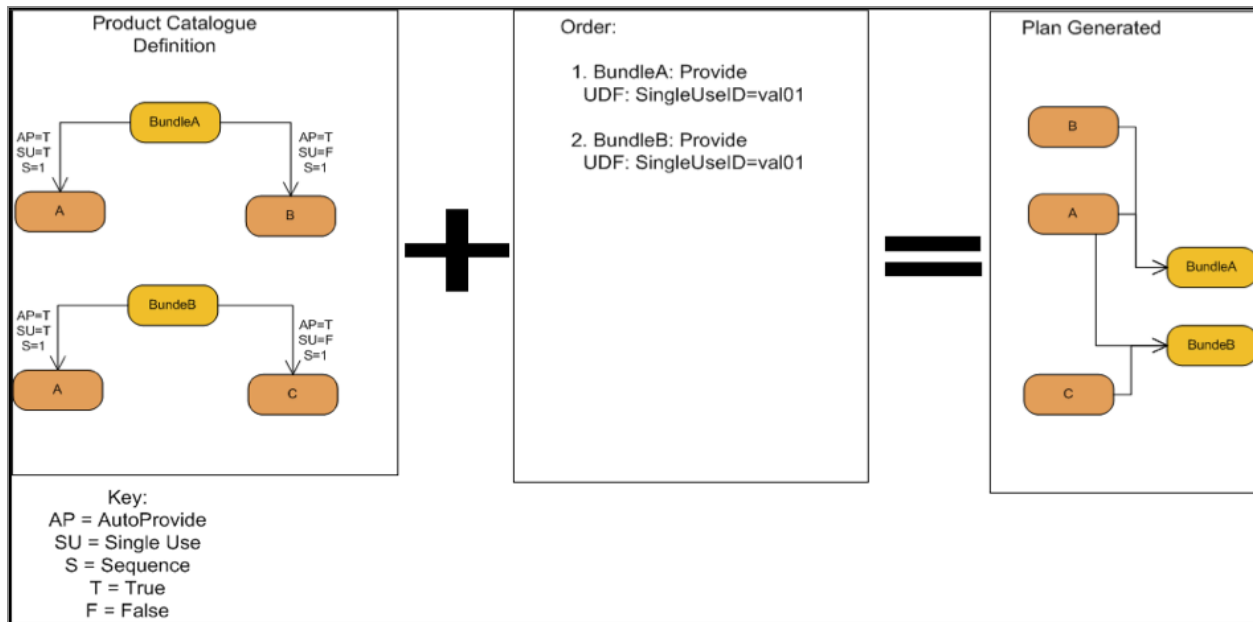
Provide Single Use

The product catalog contains 2 bundles BundleA and BundleB. BundleA contains 2 subproducts A and B. Both the subproducts have sequence set to "1" and auto provision set to "True". A has the attribute single use set to "True" when B has the attribute set to "False". BundleB contains two subproducts A and C. Both the subproducts have sequence set to "1" and auto provision set to "True". A has the attribute single use set to "True" when C has the attribute set to "False".

The order sent into AFF contains 2 order lines. Order line 1 contains BundleA with order line action Provide. Order line 2 contains BundleB with order line action Provide. Both the order lines contain a user-defined field with the name SingleUseID and the value is the same for both BundleA and BundleB.

The generated plan contains only one instance of subproduct A. BundleA, which contains a dependency to subproduct A and B. BundleB contains a dependency to subproduct A and C. The User-Defined Fields is not merged into the retained product.

Single use (Provide-Provide)

**Cease Single Use**

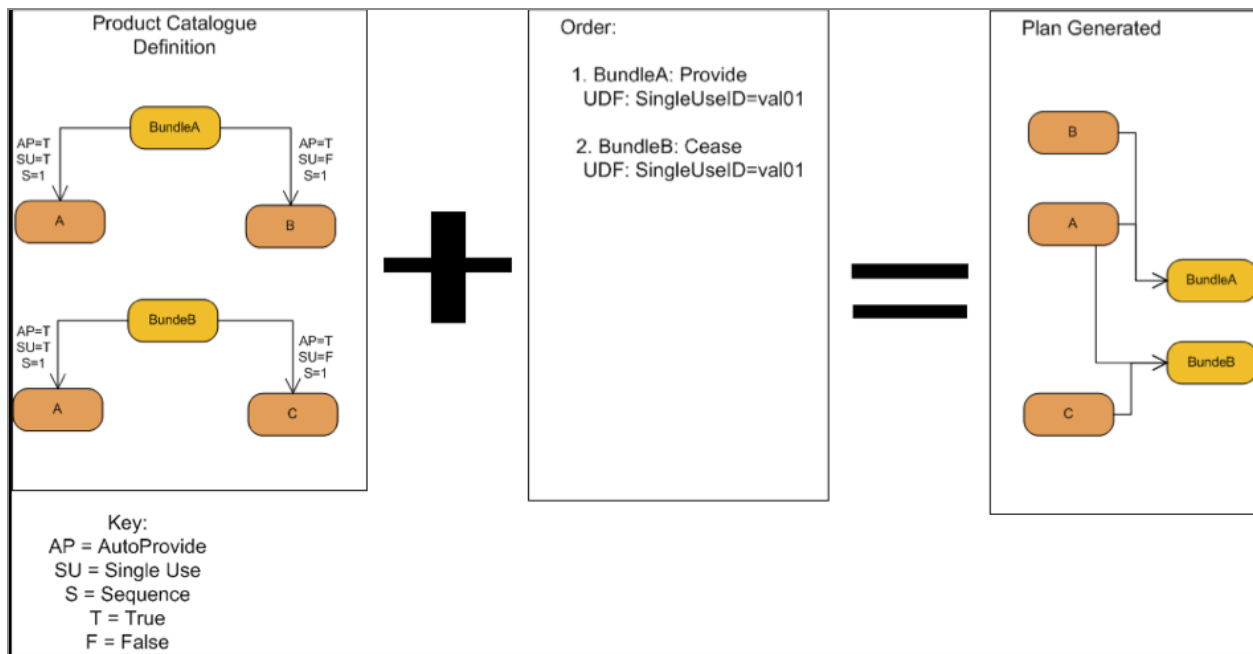
This functionality ensures that if the products have the same product ID and have been defined for single use with their order line actions as PROVIDE and CEASE then those products are not provisioned more than once. It deletes the instance, which has its order line action as CEASE, mark the action as UPDATE and also ensure that the dependencies point to the PROVIDE instance, which remains in the plan. This is done for products with the same parent only. Single Use is modeled in the product model by setting the record attribute single use as true.

In this scenario, the Cease instance of the product is removed from the plan. Bundle A has a dependency to both subproduct A and B. BundleB has a dependency to both subproduct A and C. The instance of A still left in the plan has a new line action of UPDATE. The User-Defined Fields is not merged into the retained product.

The plan fragment and the plan description are set to the Update fragments from the product information.

In cases where the subproduct A has dependent products all those dependent products is made dependent to the remaining instance of product A.

Single use (Provide-Cease)



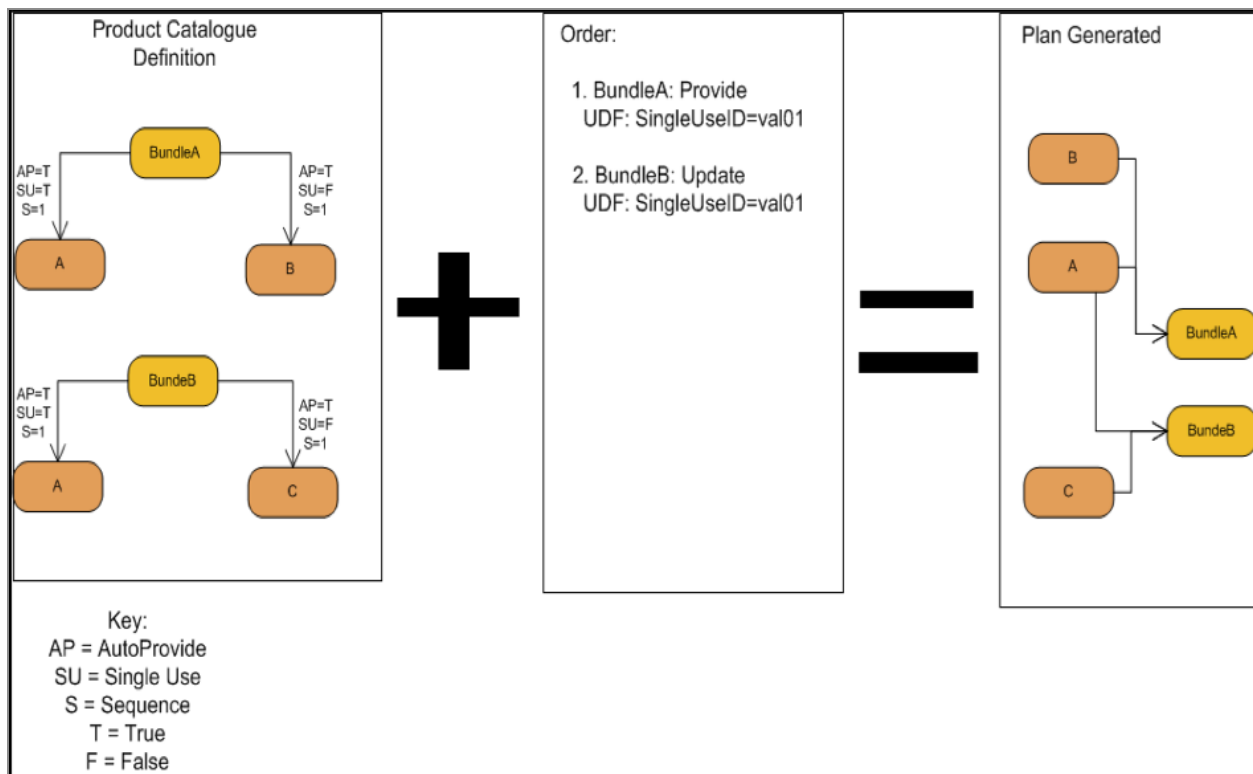
Update Single Use

This functionality ensures that if the products have the same product ID and have been defined for single use with their order line actions as PROVIDE and UPDATE then those products are not provisioned more than once. It deletes the instance, which has its order line action as PROVIDE and also ensure that the dependencies point to the UPDATE instance, which remains in the plan. This is done for products with the same parent only.

In the following scenario, the Update instance of subproduct A remains in the plan. Bundle A has a dependency to both subproduct A and B. BundleB has a dependency to both subproduct A and C. The instance of product A retains the line action of UPDATE. The User Defined Fields is not merged into the retained product.

The plan fragment and the plan description is set to the Update fragments from the product information

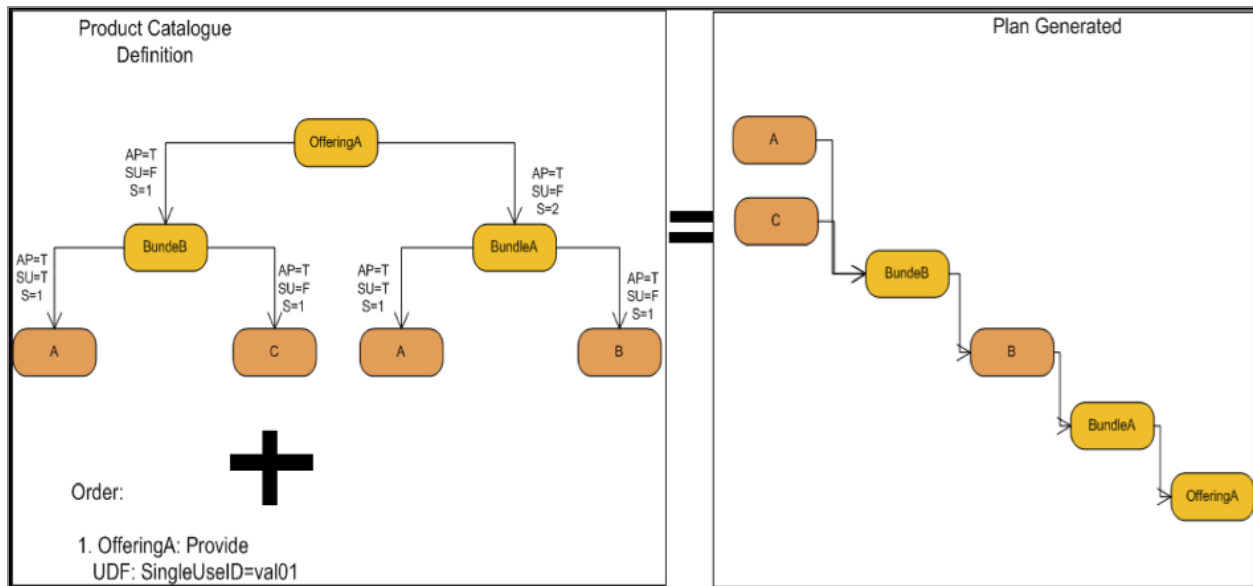
Single use (Provide-Update)

**Sequenced Single Use**

In this scenario OfferingA contains both BundleA and BundleB, which have been sequenced 2 and 1 respectively. Since both bundles contain subproduct A, this product is merged into a single instance. BundleB is the only product to contain a dependency to subproduct A since it is the 1st product to be provisioned in the plan. BundleA has the dependency to subproduct A deleted.

The User-Defined Fields are merged as mentioned in the above scenarios. The lineIDs and EOL attributes are merged as well with a comma as a separator.

Single use (Sequenced)



Product Affinity (Plan Item Level)

Through the Product Affinity between different products on the same order, the products to be grouped and fulfilled together through the execution of a single plan item occur. It can be termed as an order fulfillment optimization.

Generally, a plan item corresponding to an order line specifies a product to be fulfilled in the order. If an affinity is specified between the products that are either being fulfilled implicitly as mandatory children, or ordered explicitly as separate order lines, the individual plan items are grouped together into a single affinity plan item during plan optimization in Automated Order Plan Development. Thus, the corresponding products are fulfilled through the execution of this single plan item.

The product affinity is specified in the product catalog in one of the following two different ways:

- By specifying the affinity type and action-specific plan fragments attributes in the AffinityGroup tab in PRODUCT repository
- By assigning the plan fragments by using ProductHasXXPlanFragment relationships and specifying the affinity specific relationship attributes

The *XX* in relationship name refers to actions, such as PROVIDE, CEASE, UPDATE, and CANCEL.

Automated Order Plan Development recognizes the affinity and combines the plan items corresponding to the order lines depending on the following two main conditions:

- If the plan fragments defined in the product catalog for the ordered products are the same
- If the affinity type defined in the product catalog for the ordered products is the same (InLink or CrossLink)

user-defined field Data Handling

Affinity groups together plan items for different order lines into a single plan item. Automated Order Plan Development is also responsible for populating the User Defined Field (UDF) that are associated with these plan items. The potential exists for the same user-defined field to be present on different order lines, all values must be available in the plan and the relevant order lines identified.

Affinity between the product catalogs can be added at design time in the TIBCO® Product and Service Catalog.

At runtime, when generating the plan for a given order and the affinity is detected in multiple products, plan items related to those products (where affinity is detected) are merged into a single plan item. The UDF in both the plan items are merged.

The value of the GLOBAL_PRODUCT_NAME UDF name is not assigned randomly. The plan items in the affinity are sorted based on the plan item id. The product id of the first plan item is assigned to the value of GLOBAL_PRODUCT_NAME UDF.

The following data handling rules must be implemented:



Note: The following table lists the Sample Order Line Data representing the order lines being affinity grouped. The Sample Plan Item Data represents the output affinity grouped plan item for those order lines.

Sr No	Rule	Outcome	Sample Order Line Data	Sample Plan Item Data
1	user-defined field exists on only one of the	user-defined field name is the original user-defined field	Order Line = 1 user-defined field Name = ServiceID user-defined	user-defined field Name = ServiceID:1 user-

Sr No	Rule	Outcome	Sample Order Line Data	Sample Plan Item Data
	order lines being affinity grouped	name concatenated with the order line number. Value is the original user-defined field value.	field Value = 1234 Order Line = 2 <i>Does not contain ServiceID user-defined field</i>	defined field Value = 1234
2	user-defined field exists on more than one of the order lines being affinity grouped, but not all order lines. user-defined field value is the same on all order lines.	user-defined field name is the original user-defined field name concatenated with the order line number as a comma-separated list. Value is the original user-defined field value.	Order Line = 1 user-defined field Name = ServiceID user-defined field Value = 1234 Order Line = 2 user-defined field Name = ServiceID user-defined field Value = 1234 Order Line = 3 <i>Does not contain ServiceID user-defined field</i>	user-defined field Name = ServiceID:1,2 user-defined field Value = 1234
3	user-defined field exists on all order lines being affinity grouped. user-defined field value is the same on all order lines.	user-defined field name is the original user-defined field name. Value is the original user-defined field value.	Order Line = 1 user-defined field Name = ServiceID user-defined field Value = 1234 Order Line = 2 user-defined field Name = ServiceID user-defined field Value = 1234 Order Line = 3 user-defined field Name = ServiceID user-defined field Value = 1234	user-defined field Name = ServiceID:1,2,3 user-defined field Value = 1234
4	user-defined field exists on more than one order line	user-defined field is created for each unique user-defined field value, with the	Order Line = 1 user-defined field Name = ServiceID user-defined field Value = 1234 Order	user-defined field Name = ServiceID:1,2 user-defined

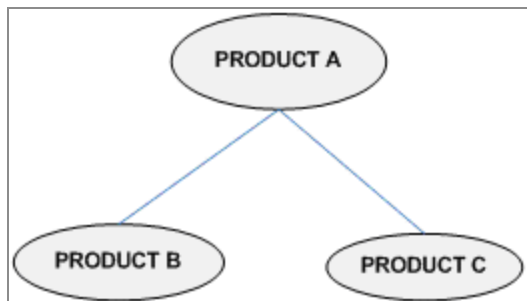
Sr No	Rule	Outcome	Sample Order Line Data	Sample Plan Item Data
	being affinity grouped. user-defined field value is different on different order lines.	corresponding name containing the original user-defined field name concatenated with the order line numbers as a comma-separated list.	Line = 2 user-defined field Name = ServiceID user-defined field Value = 1234 Order Line = 3 user-defined field Name = ServiceID user-defined field Value = 6789	field Value = 1234 user-defined field Name = ServiceID:3 user-defined field Value = 6789

TIBCO Order Management supports the following types of product affinities:

Inlink

The *Inlink Affinity* can be defined between the products at the same level in a bundle.

Inlink Affinity



As shown in the figure, the InLink affinity can be defined between the Product B, and Product C for the PROVIDE action by specifying the affinity type as **InLink**. The PROVIDE plan fragment is defined as **PC_PROVIDE_BC**.

For the InLink affinity, the **LinkID** user-defined field having the same value must be passed in the order lines.

In addition to the two conditions, Automated Order Plan Development also checks the following conditions for the InLink affinity:

1. If a value of the **LinkID** user-defined field in the plan items, which is propagated from

the order lines to be merged, is the same.

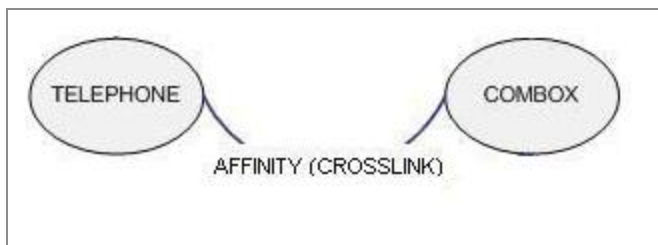
2. If the dependentOn (parent product) plan item for the plan items to be merged is the same.

If these conditions are fulfilled, the plan items are combined into a single *affinity plan item* containing the plan fragment from any of the merging plan items, since it is the same for all of them.

Crosslink

The Crosslink Affinity is defined between the products at any levels across the bundles.

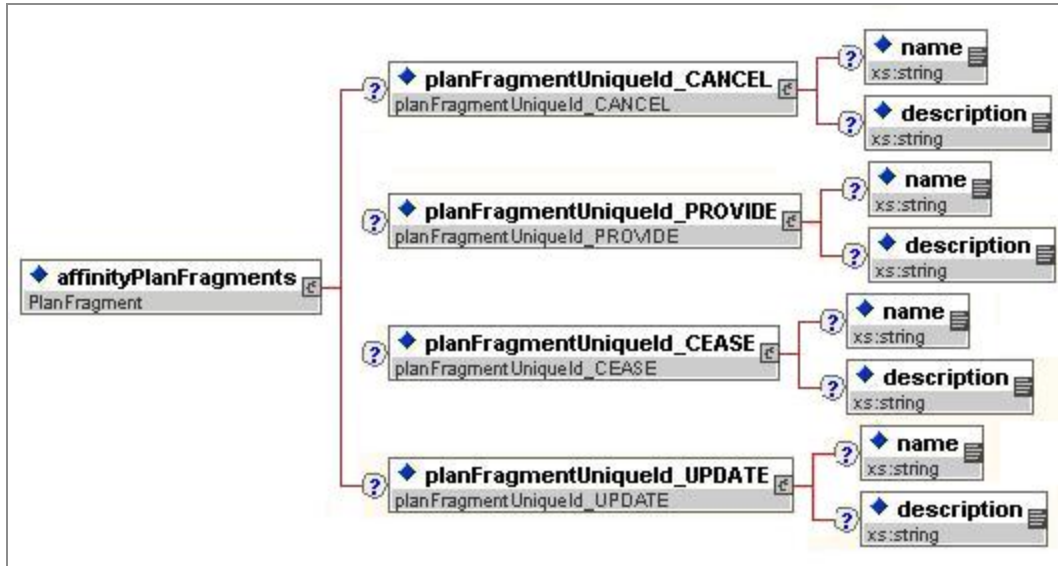
In the [Product Model diagram](#), the products **Telephone** and **COMBOX** can have CrossLink affinity between them. when the order fulfillment process, both these technical products are installed and configured in one go through the affinity grouped single plan item.



Automated Order Plan Development does not check any additional conditions for CrossLink affinity.

Affinity applies to the order plan development and this element is used to determine what plan fragments are run for the product when the affinity grouping is active. Affinity Plan Fragments XSD is illustrated as:

Affinity Plan Fragments XSD



The following table explains the Affinity Plan Fragments Data Model.

Element Name	Element Type	Description	Example
planFragmentUniqueId_CANCEL	String (Optional)	Plan fragment cancels type.	
planFragmentUniqueId_CANCEL/ name	String (Optional)	Name of the plan fragment to run when canceling this product.	EP_BUNDLE_CANCEL NO_RECIPROCAL_ACTION
planFragmentUniqueId_CANCEL/ description	String (Optional)	Description of the plan fragment to run when canceling this product.	Product 1 Cancel
planFragmentUniqueId_PROVIDE	String (Optional)	Plan fragment provides type.	
planFragmentUniqueId_PROVIDE / name	String (Optional)	Name of the plan fragment to run when providing this product.	EP_BUNDLE_PROVIDE

Element Name	Element Type	Description	Example
planFragmentUniqueld_ PROVIDE / description	String (Optional)	Description of the plan fragment to run when providing this product.	Product 1 Provide
planFragmentUniqueld_ CEASE	String (Optional)	Plan fragment ceases type.	
planFragmentUniqueld_ CEASE/ name	String (Optional)	Name of the plan fragment to run when ceasing this product.	EP_BUNDLE_ CEASE
planFragmentUniqueld_ CEASE / description	String (Optional)	Description of the plan fragment to run when ceasing this product.	Product 1 Cease
planFragmentUniqueld_ UPDATE	String (Optional)	Plan fragment update type.	
planFragmentUniqueld_ UPDATE/ name	String (Optional)	Name of the plan fragment to run when updating this product.	EP_BUNDLE_ UPDATE
planFragmentUniqueld_ UPDATE/ description	String (Optional)	Description of the plan fragment to run when updating this product.	Product 1 Update

Affinity Sequencing

Affinity Sequencing is used to support the scenario for maintaining sequencing during affinity grouping. Affinity Sequencing was introduced during affinity RulesEngine (BusinessEvents) selects plan items at random, which are then merged into a single plan item. Since items are selected at random during this process, sequencing is not maintained for plan items that must be in a sequence.

To make products available for affinity sequencing:

- Affinity TYPE value for all products where sequence must be respected must be set to

"SequencedAffinity" in the affinity type

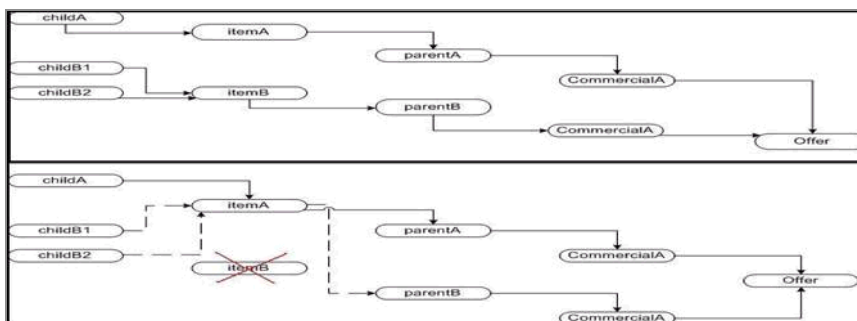
- All order lines where affinity components are known to exist must have a user-defined field defined as AffinitySequence and the value must be an integer

Depending on the AffinitySequence values being compared, the following actions are possible:

1. itemA.AffinitySequence = itemB.AffinitySequence

- If both items have dependent children the children from itemB is copied to itemA
- itemB parent is updated with the plan item ID from itemA, thus making itemA dependent to its own parent and the itemB parent
- user-defined field values from itemB is merged into itemA
- Any item, which has a reference to itemB have that reference replaced with a reference to itemA
- The instance of itemB is deleted from the plan

Parallel Scenario



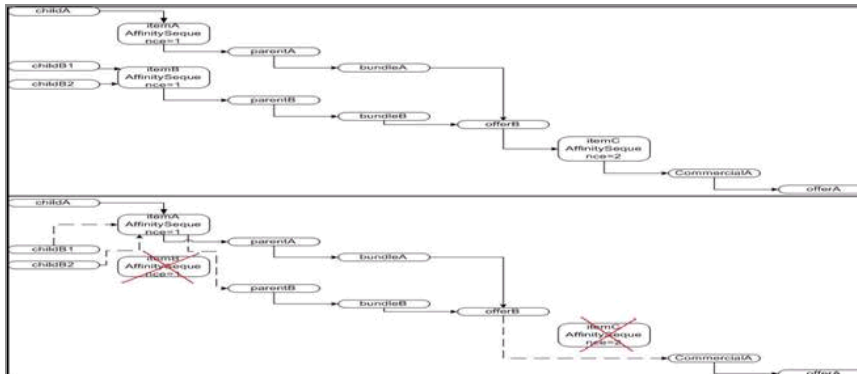
The figure depicts the scenario where the items to be affinity grouped are running in parallel. One of the items in this case itemB is deleted from the plan. The dependent items to itemB, which are childB1 and childB2 are dependent on itemA. Then itemA is made dependent to parentB, which is the parent to itemB.

2. itemA.AffinitySequence < itemB.AffinitySequence

- itemB is merged into itemA
- user-defined field values from itemB is merged into itemA
- Any item, which has a dependency to itemB have that reference removed
- all the children from itemB is made dependent to itemB parent(s)

- itemB is deleted from the plan

Sequenced Scenario

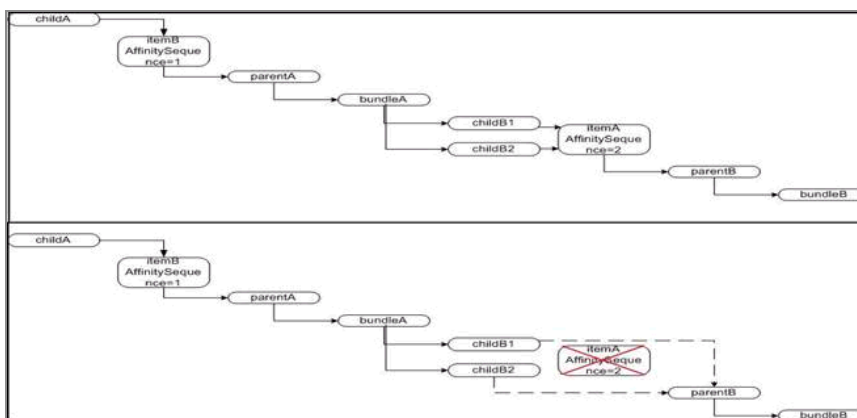


The figure depicts an offering, which has items that are in parallel and in sequence that have to be affinity grouped. For items that are in parallel they are handled similar to the figure 1. For the item that is in sequence itemC. It is dependent item offerB is made dependent to CommercialA, which is the parent to itemC.

3. itemA.AffinitySequence > itemB.AffinitySequence

- itemA is merged into itemB
- User Defined Field from itemA is merged into itemB
- if itemA has dependent children those children are copied into itemB and the parent item of itemA
- Any item, which has a reference to itemA have that reference erased
- itemA is deleted from the plan

Items in Sequence



For all the above actions the following occurs in all of them:

- EOL, Plan description and Line ID values are merged into comma-separated values from itemA and itemB
- The planID is updated with the affinity plan ID

i Note: When an order is submitted, the order lines for items, which have Affinity must have the AffinitySequence defined and updated.

i Note: To not merge certain User Defined Fields during Affinity Sequencing, those User Defined Fields must be added as a comma-separated values in the global variable CharacteristicsWithoutAffinityPostfix in the rules engine (Automated Order Plan Development).

Conditional Affinity

Conditional Affinity combines InLink and CrossLink affinities in a single affinity type and provides additional flexibility. Affinity grouping enables different plan items to be grouped together based on the evaluation of the XPATH expression defined at the product catalog. The two affinity grouping types are:

Inlik Affinity	Crosslink Affinity
Affinity groups plan items having the same parent product share a common LinkID user-defined field value and have the same affinity plan fragment name	Affinity groups plan items having the same affinity plan fragment name

The additional configuration fields and rules in conditional affinity are:

Field	Description
AffinityType	<p>Determines the type of affinity implemented.</p> <ul style="list-style-type: none"> • InLink • CrossLink • Sequenced Affinity

Field	Description
AffinityCondition	<ul style="list-style-type: none"> ConditionalAffinity <p>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:</p> <ul style="list-style-type: none"> If the expression is true, the product plan item is affinity-grouped If the expression is false, then the product plan item is not affinity-grouped If the field is blank, assume that the value is true If the XPATH expression evaluates to anything other than the true or false, Automated Order Plan Development fails and returns an exception <p>The XPATH expression evaluates against the following data fields on the order:</p> <ul style="list-style-type: none"> Order Header user-defined field Name and Value Order Line ProductID Order Line Action and ActionMode Order Line user-defined field Name and Value <p>The XPATH expression can also be defined against the following plan data fields:</p> <ul style="list-style-type: none"> planItem productID planItem user-defined field name value planItem Action
AffinityCorrelation	<p>Valid for Conditional type only. The XPATH is evaluated on the Plan data and the order data. A String field containing an xpath expression based on a data is in the following order:</p> <ul style="list-style-type: none"> All plan items that evaluate to the same AffinityCorrelation are grouped together

Field	Description
	<ul style="list-style-type: none"> • The field is functionally similar to the LinkID method of correlating plan items in the InLink affinity. However, it lets correlation based on complex conditions without a restriction on the user-defined field names • If the field is blank, a default LinkID value is shared by all other blank configurations • If the XPATH expression evaluates to an empty string, the XPATH expression is blank, or assume a default LinkID <p>The XPATH expression evaluates against the following order data fields:</p> <ul style="list-style-type: none"> • Order Header user-defined field Name and Value • Order Line ProductID • Order Line Action and ActionMode • Order Line user-defined field Name and Value <p>The XPATH expression can also be defined against the following plan data fields:</p> <ul style="list-style-type: none"> • planItem productID • planItem user-defined field name value • planItem Action
AffinityParentGroup	<p>Valid for Conditional type only. A Boolean field containing the value true or false:</p> <ul style="list-style-type: none"> • If set to true, the plan items with products sharing the same immediate parent product are grouped together • If set to false, the parent product is not considered for grouping
AffinityActionGroup	<p>Valid for Conditional type only. A Boolean field containing the value true or false:</p> <ul style="list-style-type: none"> • If set to true, then only plan items with products that share the same action are grouped together

Field	Description
	<ul style="list-style-type: none"> • If set to false, then the action is not considered for grouping
AffinityActionValue	<p>AffinityActionValue is considered for grouping when AffinityActionGroup is set to true. This is valid for Conditional type only. String field containing an XPATH expression that evaluates to a String based on data is in the following order: The XPATH expression must evaluate to one of the following:</p> <ul style="list-style-type: none"> • PROVIDE • UPDATE • CEASE • Empty String <p>If the XPATH expression evaluates to anything other than these actions, then Automated Order Plan Development fails and returns an exception.</p> <ul style="list-style-type: none"> • If the field is blank, or the return value from the XPATH expression is an empty string, the remaining action rules must be applied. <p>The XPATH expression can evaluate against the following data fields on the order:</p> <ul style="list-style-type: none"> • Order Header user-defined field Name and Value • Order Line Action and ActionMode • Order Line user-defined field Name and Value <p>The XPATH expression can also be defined against the following plan data fields:</p> <ul style="list-style-type: none"> • planItem productID • planItem user-defined field name value • planItem Action
AffinityProvide	<p>Provide plan fragment name for affinity grouped plan item. Only plan items with the Provide action and the same value in this field are grouped together</p>

Field	Description
AffinityUpdate	Update plan fragment name for affinity grouped plan item. Only plan items with the Update action and the same value in this field are grouped together
AffinityCease	Cease plan fragment name for affinity grouped plan item. Only plan items with the Cease action and the same value in this field are grouped together
AffinityCancel	Cancel plan fragment name for affinity grouped plan item. Only plan items with the Cancel action and the same value in this field are grouped together

In the case where plan items with different actions are grouped together due to affinity, the following logic is used to determine what action to apply to the plan item. The following rules apply:

1. If AffinityActionValue is specified, then the action of the plan item is the result of evaluating this xpath.
2. If AffinityActionValue is blank, or evaluates to an Empty String, then the remaining rules apply:
 - a. If all order lines have the same action, then the plan item action is the same as the order lines.
 - b. If order lines have different actions, then:
 - i. If at least one order line has PROVIDE action, then the plan item has PROVIDE action.
 - ii. Otherwise if at least one order line has CEASE action, then the plan item has CEASE action.
 - iii. Otherwise, the plan item has UPDATE action.

For details, see *TIBCO® Product and Service Catalog Product Catalog Guide*.

Important: 1) If XPath is defined against plan data, the format must be `<Actual XPath>` containing string `$var/PlanItem`. For example, if you want to define the XPath for user-defined field name-value pair `MSISDN=123`, the XPath can be `$var/PlanItem[productID='GSMLine']/udfs[name='MSISDN']/value/text()`.

XPath evaluates data from the `planItem`. Refer to the *sample planItem xml*.

2) If XPath is defined against the order data, the format must be `<Actual XPath>` containing string `$var/Order`. Refer to *Sample order XML*.

3) Default order data is considered for evaluation if XPATH does not contain `$var/PlanItem`.

Note: See *Sample XPATHs* for XPATH definitions.

When Automated Order Plan Development returns a plan it indicates the action of the plan item. Under normal circumstances, this maps directly to the action of the associated order line that caused the creation of the plan item. In the case where plan items with different actions are grouped together, the following logic is applicable to determine what action to apply to the plan item.

1. If `AffinityActionValue` is specified, then the action of the plan item is the result of evaluating this xpath.
 - If `AffinityActionValue` is blank, or evaluates to an Empty String, then the remaining rules apply
2. If all order lines have the same action, then the plan item action is the same as the order lines.
3. If order lines have different actions, then:
 - If at least one order line has PROVIDE action, then the plan item has PROVIDE action.
 - Otherwise if at least one order line has CEASE action, then the plan item has CEASE action.
 - Otherwise, the plan item has UPDATE action.

Conditional Affinity Sample

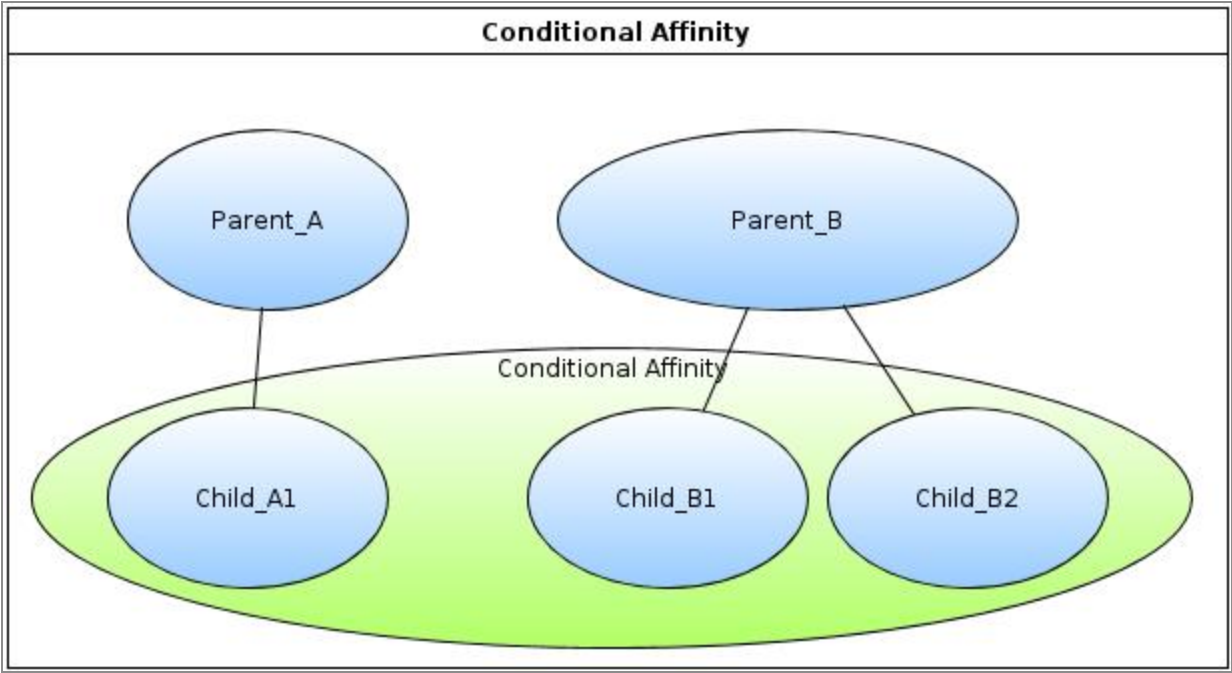
A product model is having two parents:

- Parent_A
- Parent_B

Consider a product model where conditional affinity is defined at all the child products:

- Child_A1
- Child_B1
- Child_B2

Conditional Affinity



The XPATH defined in the product model is evaluated against the submitted order schema. The following table describes the attribute-based conditional affinity scenarios:

Attribute	Sample XPATH Expressions	Order Payload
AffinityCondition	exists (\$var/Order/OrderHeader UDF[name=UDFNAME and value="UDFVALUE"])	<div><div><ord1:udf> <ord1:name>UDFNAME</ord1:name> <ord1:value>UDFVALUE</ord1:valu</div></div>

Attribute	Sample XPATH Expressions	Order Payload
		<pre>e> </ord1:udf></pre>
AffintyCorrelation	<pre>\$var/Order/orderlines [productID= 'Child_ A1']/ OrderlinesUDF [name= 'UDFNAME']/ value/text()</pre>	<pre><ord1:line> <ord1:lineNumber>1</ord1:lineNum ber> <ord1:productID>Child_ A1</ord1:productID> <ord1:quantity>1</ord1:quantity> <ord1:uom>1</ord1:uom> <ord1:action>PROVIDE</ord1:actio n> <ord1:actionMode>New</ord1:actio nMode> <ord1:udf> <ord1:name>UDFNAME</ord1:name> <ord1:value>UDFVALUE</ord1:valu e> </ord1:udf> </ord1:line></pre>
AffintyParentGroup	<p>Child_B1 and Child_B2 have immediate parent. The two is affinity grouped when: AffintyParentGroup=true</p>	
AffinityActionValue The affinityAction	<pre>\$var/Order/orderlines [productID= 'Child_ A1']/OrderlinesUDF [name=</pre>	<pre><ord1:line></pre>

Attribute	Sample XPATH Expressions	Order Payload
Group must be true	'UDFNAME']/value/text()	<pre> <ord1:lineNumber>1</ord1:lineNumber> <ord1:productID>Child_A1</ord1:productID> <ord1:quantity>1</ord1:quantity> <ord1:uom>1</ord1:uom> <ord1:action>PROVIDE</ord1:action> <ord1:actionMode>New</ord1:actionMode> <ord1:udf> <ord1:name>UDFNAME</ord1:name> <ord1:value>UDFVALUE</ord1:value> </ord1:udf> </ord1:line> </pre>

Configurable Handling of CrossLink + ProductComprisedOf Conflicts and Single Use + ProductComprisedOf Conflicts


Affinity can violate the product model ProductComprisedOf action-based sequencing when the provisioning of two or more products must be grouped through a single affinity plan item for execution but have different parents, which provisioning must be sequenced in a specific order. The affinity plan item is run for all parents irrespectively of the ProductComprisedOf action-based sequencing, which breaks the product model and can lead to circular dependencies and missing dependencies.

System config parameters must be added to trigger the following behavior:

- **Error:** If the Affinity and ProductComprisedOf conflict, stop and report an error.
- **Ignore:** If the Affinity and ProductComprisedOf conflict, ignore the Affinity rule and

apply only ProductComprisedOf.

- **Apply:** If the Affinity and ProductComprisedOf conflict, the process with both rules applied but add dependencies.

 **Note:** This applies to all other Affinity types.

SingleUseHandling: Error | Ignore | Apply

- **Error:** If single use and ProductComprisedOf conflict, stop and report an error.
- **Ignore:** If single use and ProductComprisedOf conflict, ignore the SingleUse rule and apply only ProductComprisedOf.
- **Apply:** If single use and ProductComprisedOf conflict, process with both rules applied but add dependencies.

The default is *Apply*.

Sort Plan

The sort plan functionality was implemented to sort the plan according to the subscriber for batch orders with multiple subscribers contained within the plan.

The sort plan function is defined to sort the plan according to an attribute defined within the order. This function makes sure that products that belong to similar grouping attributes follow each other in the GUI.

In scenarios where bulk orders for multiple subscribers are submitted into TIBCO Order Management, the subscriber ID is used as the grouping mechanism. All the order lines have a user-defined field defined with the name SubscriberID. Once the entire plan has been generated all the plan items are sorted according to the value for the subscriber ID user-defined field. This ensures that all products for an individual subscriber follow one after the other. This is followed by the next subscriber and so on.

Attribute-Based Decomposition

This functionality defines the decomposition rules along the relationship path for products. With the decomposition rule that is defined as XPATH logic, you can apply the defined logic along with the order.

This feature applies to the products having PCO, PDO, and PRF relationships.

The attribute-based decomposition can be applied if the following conditions are satisfied:

- The DECOMPOSITION attribute must exist in the product where the XPATH logic can be defined.
- The XPATH logic must exist (As the XPATH logic is evaluated to true or false).

Automated Order Plan Development supports two xpath evaluation patterns. If the xpath expression contains \$var//Product, attribute-based decomposition is evaluated based on the old evaluation technique. If the xpath expression contains \$var/Order, attribute-based decomposition is evaluated based on the new evaluation technique.

Evaluation expression with "Product"

The purpose of the logic can be to check for the user-defined fields or a particular product within the order.

Example:

```
exists($var//Product[udf[name='AccessType' and value='copper']])

exists($var//Product[name='PO_TV' and udf[name='AccessType' and
value='copper']])
```

xml representation for the same expression is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<product> <!-- names are productid from every orderline of order >
  <name>PO_TV</name>
  <name>CFS_TV</name>
  <name>PRODUCT_TV</name><!-- udfs included based on flags and order
input>
  <udf>
    <name>Name1</name>
    <value>Value1</value>
  </udf>
  <udf>
    <name>AccessType</name>
    <value>copper</value>
  </udf>
  <udf>
    <name>Name3</name>
```

```

        <value>Value3</value>
    </udf>
</product>

```

The XPath for attribute-based decomposition can be used against the user-defined fields. The user-defined field considerations can also depend on flags such as `includeproductmodelcharacteristics` and `includeonlyproductorderline`.

Description of these flags are listed as follows:

includeproductmodelcharacteristics: By default this flag is false. The user-defined fields can come from orderline or the product model characteristics. Automated Order Plan Development configuration flag `includeproductmodelcharacteristics` is used to include product model characteristics for xpath execution. By default, product model characteristics are not considered.

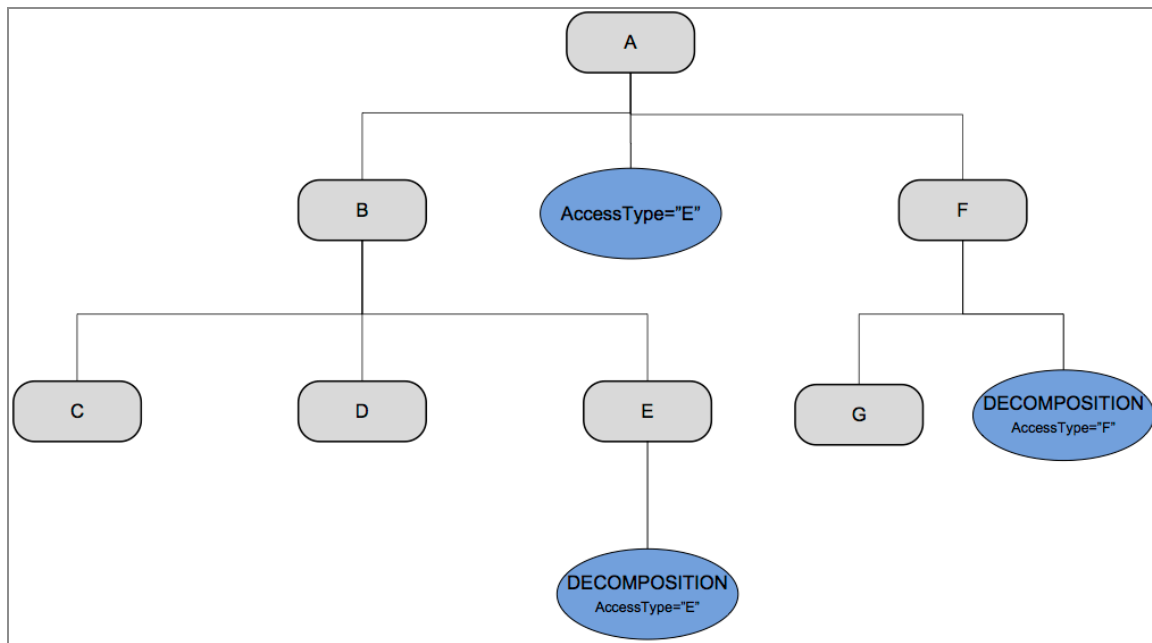
includeonlyproductorderline: By default, this flag is false and it considers those orderline's user-defined fields from the order to check against whose Xpath must be executed that satisfies the following conditions:

- Have the same linkid with orderline bundle which is responsible for including the product instance in the plan.
- Product in orderline is a child of the evaluating product.

Alternatively it can also be made to execute only against the orderline from which the product is being decomposed by setting this flag to `true`, by default, which is `false`.

Example: You can define that Copper Access or Fiber Access process component must only be in the plan if the Access Type in the order is Copper or Fiber.

A simple scenario for attribute-based decomposition is described as follows:



Attribute-based decomposition of the product generated. The DECOMPOSITION characteristic for F and E contains a relationship value with AccessType set to either F or E. The Decomposition characteristic can contain complex XPATH logic, which can be used to determine which branch of the tree must be included in the final execution plan for the offering. The design takes into account the new product catalog characteristic called DECOMPOSITION. The decomposition engine processes the characteristics and determines which branch in the product hierarchy is required for the final execution plan.

If the order access type E is specified, then branch F is not included in the execution plan and E is included. If access type F is specified then E is not included in the execution plan and F is included in the plan.

Evaluation expression with "Order"

The logic can check user-defined fields along with productID in orderline within the order.

Example:

```
exists($var/Order/orderlines[productID='PO_TV1']/OrderlinesUDF
[name='OLUDF_Name2' and value='OLUDF_Value2'] or $var/Order/orderlines
[productID='PO_TV1']/OrderlinesUDF[name='OLUDF_Name1' and value='OLUDF_
Value1'])
```

It can also be evaluated against OrderHeaderUDF.

Example:

```
exists($var/Order/OrderHeaderUDF[name=UDFNAME and value="UDFVALUE"])
```

- As part of this extended behavior, now xpath evaluation is supported on the order object for attribute-based decomposition.
- The `com.tibco.af.aopd.flags.includeonlyproductorderline` flag is not supported when you choose to evaluate xpath on the order xml, as this flag was introduced to avoid line-udf linking conflict in xpath evaluation with "Product". With the new xpath evaluation technique there is no such conflict.
- `includeproductmodelcharacteristics` flag is not supported when you choose to evaluate xpath on order xml, as product model characteristics, are not required when evaluating xpath using the order object.

When you are not evaluating xpath on the exact `OrderRequest.xml`, which is submitted from a northbound-system as parsing the entire `OrderRequest.xml`, it is performed on the following xml file. You can configure xpath from the following object xml representation:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<order>
  <businessTransactionID>Order_1</businessTransactionID>
  <currentTime>2022-01-14T16:40:57.071+05:30</currentTime>
  <customerref>CustomerRef 1</customerref>
  <jmsPriority>4</jmsPriority>
  <orderHeaderUDF>
    <name>HeaderUDFName1</name>
    <value>HeaderUDFValue1</value>
  </orderHeaderUDF>
  <orderHeaderUDF>
    <name>HeaderUDFName2</name>
    <value>HeaderUDFValue2</value>
  </orderHeaderUDF>
  <orderID>Order_1</orderID>
  <orderRef>Order_1</orderRef>
  <orderlines>
    <action>Provide</action>
    <dynamic>true</dynamic>
    <lineID>1</lineID>
    <lineUsed>true</lineUsed>
    <linkID>1</linkID>
    <linkParentID>1</linkParentID>
    <linkedParentID>1</linkedParentID>
```

```

    <offerId>offerID</offerId>
    <orderlinesUDF>
      <name>OLUDF_Name1</name>
      <value>OLUDF_Value1</value>
    </orderlinesUDF>
    <orderlinesUDF>
      <name>OLUDF_Name2</name>
      <value>OLUDF_Value2</value>
    </orderlinesUDF>
    <parentID>parentID</parentID>
    <productID>PO_TV1</productID>
    <quantity>1</quantity>
    <requiredByDate>2022-01-14T16:40:57.078+05:30</requiredByDate>
    <status>Pending</status>
    <subscriberID>Subscriber</subscriberID>
    <timeDelay>2022-01-14T16:40:57.078+05:30</timeDelay>
  </orderlines>
  <orderlines>
    <action>Provide</action>
    <dynamic>true</dynamic>
    <lineID>2</lineID>
    <lineUsed>true</lineUsed>
    <linkID>1</linkID>
    <linkParentID>1</linkParentID>
    <linkedParentID>1</linkedParentID>
    <offerId>offerID</offerId>
    <orderlinesUDF>
      <name>OLUDF_Name1</name>
      <value>OLUDF_Value1</value>
    </orderlinesUDF>
    <orderlinesUDF>
      <name>OLUDF_Name2</name>
      <value>OLUDF_Value2</value>
    </orderlinesUDF>
    <parentID>parentID</parentID>
    <productID>PO_TV2</productID>
    <quantity>2</quantity>
    <requiredByDate>2022-01-14T16:40:57.078+05:30</requiredByDate>
    <status>Pending</status>
    <subscriberID>Subscriber</subscriberID>
    <timeDelay>2022-01-14T16:40:57.078+05:30</timeDelay>
  </orderlines>
  <originator>Member1</originator>
  <parentID>Orphan</parentID>
  <requiredByDate>2022-01-14T16:40:57.078+05:30</requiredByDate>
  <rollback>true</rollback>

```

```
<sessionID>Order_1</sessionID>  
<status>Execution</status>  
<timeDelay>1000</timeDelay>  
</order>
```

With this order xml, you can generate custom xpath on all other order fields.

ProductDependsOn and ProductRequiredFor Relationships

ProductDependsOn relationship: ProductDependsOn (PDO) is a product dependency relationship to sequence the associated target and source plan items. The flexible product decomposition occurs through the ProductDependsOn relationship. This establishes a relationship between two products and is evaluated during the decomposition process.

ProductRequiredFor relationship: The ProductRequiredFor (PRF) relationship is a prerequisite relationship for a product to add a target plan item.

The ProductDependsOn and ProductRequiredFor relationships enable you to create product offers without defining sequencing for the products. You can create ProductDependsOn relationship to lower-level products instead of using ProductComprisedOf links.

The ProductDependsOn functionality provides a base behavior that permits to sequence plan items corresponding to products related by ProductDependsOn when:

1. ProductDependsOn source and ProductDependsOn target product instances have no LINKID defined.
2. ProductDependsOn source and ProductDependsOn target product instances have LINKID defined and have the same LINKID value.

The feature can extend the base behavior and sequence additionally plan items corresponding to products related by ProductDependsOn when:

1. ProductDependsOn source product instance has LINKID defined and ProductDependsOn target product instances have no LINKID defined.
2. ProductDependsOn source the product instance has no LINKID and target product instances have a LINKID defined.

A ProductDependsOn source product instance can relate to multiple ProductDependsOn target product instances by using base and extended cases so that the following use cases are possible:

- A ProductDependsOn source product that has a LINKID defined and is related to a ProductDependsOn target instance that has the same LINKID defined can be also related to ProductDependsOn target product instances that have no LINKID defined.
- A ProductDependsOn source product that has no LINKID defined and is related to a ProductDependsOn target instance that has no LINKID defined can be also related to ProductDependsOn target product instances that have a LINKID defined. By default, for ProductDependsOn sequencing, the base behavior is enabled. To enable the extended behavior set the "EnableBiDirectionalLinkID" to true, by default it is false.

By default, only one instance of the required product per LinkID is available in the plan. If there is a requirement to override this behavior to include multiple instances of the required product with the same LinkID, then the Automated Order Plan Development flag "AllowMultipleRequiredProducts" must be set to true. By default, it is false.



Note: The ProductRequiredFor adds the required plan item without dependency, if you create only the ProductRequiredFor.

The ProductDependsOn and ProductRequiredFor relationships have the following two relationship attributes:

- Source Action
- Target Action

The ProductRequiredFor relationship also has the third relationship attribute named ocvValidationReq. This is a Boolean flag for validation. Based on the validation flag, Automated Order Plan Development adds the product configured with the ProductRequiredFor relationship in the plan.

The ProductDependsOn relationship also has the third relationship attribute named 'sequenceDirection'. The valid values of this attribute are either 'AFTER' or 'BEFORE'. This attribute is paired with the provided values of SourceAction and TargetAction. For each SourceAction and TargetAction, there is a value defined for the sequenceDirection attribute.

- A 'BEFORE' sequence direction creates a dependency of the target product on the source product.

- An 'AFTER' sequencing direction creates a dependency of the source product on the target product. This is the default.

If no value is provided in the sequenceDirection attribute, the attribute defaults to "AFTER", and the functionality works as it did before the introduction of the sequenceDirection relationship attribute. The backward compatibility occurs as a result.

The value defined in the sequenceDirection attribute creates a dependency of the target product on the source product or it creates a dependency of the source product on the target product.

If a ProductDependsOn Source Product has a dependency on child products, then those child products have a dependency on the ProductDependsOn target product by default. If there is a requirement to override this default behavior and set the dependency of the source product directly on the target product, then the value of the flag "IgnorePDOFirstChildDependency" needs to be set to true in the Automated Order Plan Development configuration file. By default, this value is false.



Note: ProductDependsOn relationship can be made conditional by using the XPATH statement stored in the optional product characteristic DECOMPOSITION_DEPENDS_ON.

ProductRequiredFor relationship can be made conditional by using the XPATH statement stored in the optional product characteristic DECOMPOSITION_REQUIRED_FOR.

Source and Target Attribute Values

The following table describes the different possible combinations:

SourceAction	TargetAction
PROVIDE	PROVIDE
PROVIDE	UPDATE
PROVIDE	CEASE
PROVIDE	CANCEL

SourceAction	TargetAction
UPDATE	PROVIDE
UPDATE	UPDATE
UPDATE	CEASE
UPDATE	CANCEL
CEASE	PROVIDE
CEASE	UPDATE
CEASE	CEASE
CEASE	CANCEL
CANCEL	PROVIDE
CANCEL	UPDATE
CANCEL	CEASE
CANCEL	CANCEL

You can also define source action and target action to match the following combination by using uppercase, comma-separated values. For example:

SourceAction: PROVIDE,PROVIDE,UPDATE,CEASE,CANCEL,CEASE

TargetAction: UPDATE,CANCEL,PROVIDE,UPDATE,PROVIDE,UPDATE

You can also define sequenceDirection to match the following combination by using uppercase, comma-separated values. For example:

SourceAction: PROVIDE,PROVIDE,UPDATE,CEASE,CANCEL,CEASE

TargetAction: UPDATE,CANCEL,PROVIDE,UPDATE,PROVIDE,UPDATE

SequnceDirection: AFTER,BEFORE,AFTER,BEFORE,BEFORE,AFTER

i Note: There cannot be any space between the commas and the values.

Dependency between planitems occurs when both the following occur:

- The sequenceDirection attribute has valid values, that is, either 'AFTER' or 'BEFORE.'
- The number of sequenceDirection attributes matches with the number of Source Actions and the number of Target Actions.

i Note: There is only one target action for any given source action.

The following table explains the ProductDependsOn and ProductRequiredFor relationships and their impact on orders and plans.

Product Configuration	Order	Plan
Product A has a ProductRequiredFor relationship with Product B having source action and target action PROVIDE and PROVIDE	OL1=ProductA	Two plan item (A and B) do not depend on each other
Product A has ProductRequiredFor and ProductDependsOn relationship with B and ProductRequiredFor and ProductDependsOn has source action and target action PROVIDE and PROVIDE	OL1=ProductA (Action=Provide) OL2=ProductB (Action=Provide)	planItemA depends on planItemB
Product A has ProductRequiredFor and ProductDependsOn relationship with B and ProductRequiredFor and ProductDependsOn has source action and target action PROVIDE and PROVIDE	OL1=ProductA	planItemA depends on planItemB
Product A has ProductDependsOn relationship with B having source action and target action PROVIDE and PROVIDE	OL1=ProductA (Action=Provide) OL2=ProductB (Action=Provide)	planItemA depends on planItemB

The following table explains ProductDependsOn with Sequence direction and their impact on orders and plans.

Product Configuration	Order	Plan
Product A has ProductDependsOn relationship with B having SA & TA as PROVIDE & PROVIDE. SequenceDirection is AFTER.	OL1=ProductA (Action=Provide) OL2=ProductB (Action=Provide)	Two planitem having planItemA depends on planItemB
Product A has ProductDependsOn relationship with B having SA & TA as PROVIDE & PROVIDE. SequenceDirection is BEFORE.	OL1=ProductA (Action=Provide) OL2=ProductB (Action=Provide)	Two planitem having planItemB depends on planItemA
Product A has ProductDependsOn relationship with B having SA & TA as PROVIDE & PROVIDE. SequenceDirection is AFTER. Product B has ProductDependsOn relationship with C having SA & TA as PROVIDE & PROVIDE and SequenceDirection is BEFORE.	OL1=ProductA (Action=Provide) OL2=ProductB (Action=Provide) OL3=ProductC (Action=Provide)	Three planitems having planItemA depends on planItemB and planItemC depends on planItemB
Product A has ProductDependsOn relationship with B having SA & TA as PROVIDE & PROVIDE. SequenceDirection is BEFORE. Product B has ProductDependsOn relationship with C having SA & TA as PROVIDE & PROVIDE and SequenceDirection is AFTER.	OL1=ProductA (Action=Provide) OL2=ProductB (Action=Provide) OL3=ProductC (Action=Provide)	Three planitems having planItemB depends on planItemA and planItemB depends on planItemC

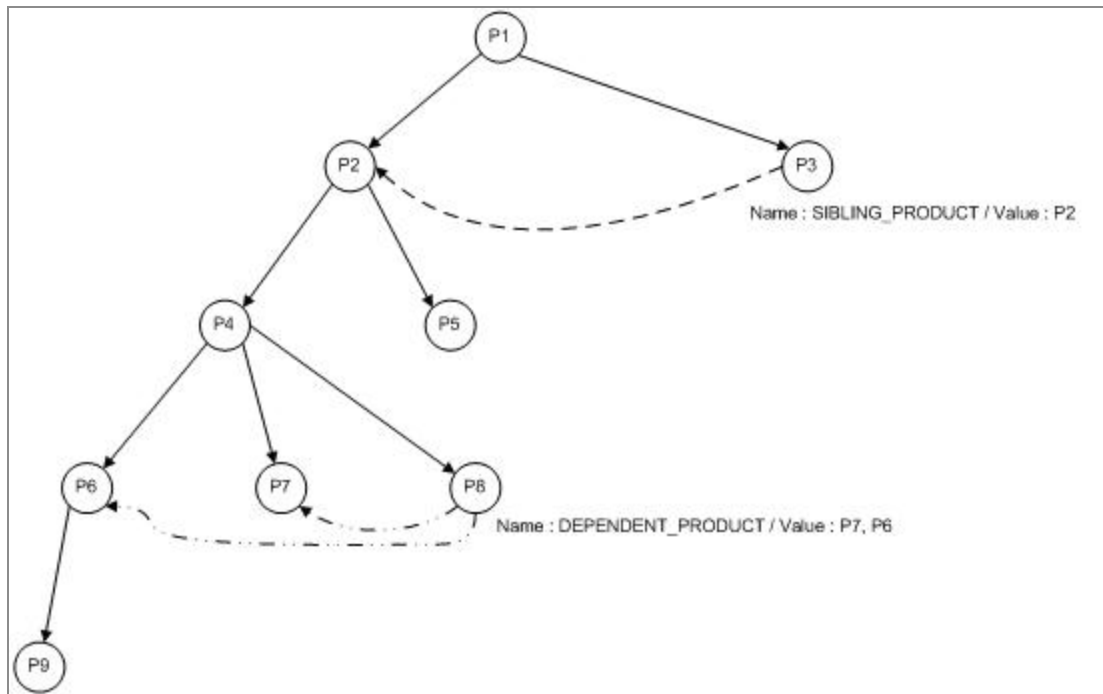
Dependent and Sibling Products

TIBCO Order Management provides the ability in the product catalog to indicate that a product is dependent on its peer [DEPENDENT_PRODUCT] or peer's hierarchy [SIBLING_PRODUCT].

The only difference between dependent products and sibling products is that the dependent is not added the peer product's children to be included in the subsequent southbound service calls when the sibling product adds the peer's children on the southbound service calls.

The diagram shows the product model.

Data Model



P8 has a dependent product link to P7 and P6. This means that the process component corresponding to P8 can use the output User-Defined Fields of P7 and P6 during the execution provided P7 and P6 have been ordered directly or indirectly in the order and corresponding process components have been run.

P3 has a sibling product link to P2. This means that the process component corresponding to P3 can use the output User-Defined Fields of P2, P4, P5, P6, P7, P8, and P9 during the execution provided P2 has been ordered directly or indirectly in the order and the corresponding process components have been run.

The 6 product characteristics as explained in the table below must be added in the product model defined in the TIBCO® Fulfillment Catalog or offline model XML. The dependent or sibling link can be defined for a product by creating the **Characteristic** relationship with one of the above relevant products [as per the scenario] with the value of the **RelationshipValue** attribute as the comma-separated IDs of the dependent or sibling products.

Name	Description
DEPENDENT_PRODUCT	Dependent product characteristic for PROVIDE scenario

Name	Description
DEPENDENT_PRODUCT_Cease	Dependent product characteristic for CEASE scenario
DEPENDENT_PRODUCT_UPDATE	Dependent product characteristic for UPDATE scenario
SIBLING_PRODUCT	Sibling product characteristic for PROVIDE scenario
SIBLING_PRODUCT_Cease	Sibling product characteristic for CEASE scenario
SIBLING_PRODUCT_UPDATE	Sibling product characteristic for UPDATE scenario

For example, the Dependent link for P8 in case of the PROVIDE scenario can be specified by creating a Characteristic relationship between P8 and DEPENDENT_PRODUCT with the value of RelationshipValue as "P6, P7".

Shared Attributes

This section describes the Shared Attributes and its sample test scenarios.

Shared Attributes are used when two Products (parent to child and sibling) share the attribute and its corresponding value and an update in the value of one needs to be reflected in the other. If an attribute is deemed as a shared attribute and when the value was passed on the order, then during decomposition the value is copied to the other products based on the EvaluationPriority set on the other products.

EvaluationPriority

Multiple products can have the same shared attribute. Hence, if the value for a shared attribute needs to be merged with the same shared attribute in other products, the user needs to define the EvaluationPriority, which indicates the priority of merging the specified characteristic from the target Product.

During the plan development process, Automated Order Plan Development checks if the characteristic (that is, the attribute) is of type 'Shared', if yes then it checks the EvaluationPriority for the characteristic. If the products mentioned on the EvaluationPriority have the same shared attribute, then the value for the characteristic is taken from the product. If none of the products mentioned on the EvaluationPriority have the same shared attribute OR EvaluationPriority is not defined, then the value is taken from the order line (if passed in the order line) or product model.

Second part of the Shared attribute definition mentions that the update in the value of the shared attribute in one product needs to be reflected in other products having the same shared attribute.

During execution, the process component might have to update the attribute (user-defined field) values. To update the value of a user-defined field, the process component calls setPlanItem on the Transient Data Store mentioning the User-Defined Fields to be updated. Process component sends the following details for the user-defined field:

1. **Name** - name of the user-defined field to update
2. **Value** - updated value
3. **Flavor** - 'output' (this indicates that process component has updated the value from set/get methods), Input (this indicates that process component has updated the value from order line), Config (this indicates that process component has updated the value from Product model)
4. **Type** - Shared (if user-defined field is of type Shared)

On the subsequent calls to getPlanItems on Transient Data Store (the process component might make this call to get details such as User-Defined Fields for plan items from Transient Data Store), Transient Data Store checks if the requested plan items have any user-defined field (that is, attributes) with the type as 'Shared'. If Shared User Defined Fields are present, then the Transient Data Store checks the EvaluationPriority for that user-defined field.

For the products mentioned on the EvaluationPriority, for each product (in the order of priority) the Transient Data Store checks if it contains the user-defined field with the same name and flavor = output. If the Transient Data Store finds such user-defined field, then the value from this user-defined field is returned. If EvaluationPriority is not defined OR products mentioned on the EvaluationPriority do not contain the user-defined field with the same name and "output" flavor, then the value from the order line/product model is returned (that is, merging is not done).

Below are the sample of EvaluationPriority:

- For a single product, product ID is followed by priority with colon in between them.

```
<productId>:<priority>  
Example:  
<ns0:evaluationPriority>SIM_TECNICO_BP1:1</ns0:evaluationPriority>
```

- For multiple products, sets of product-priority are separated by comma.

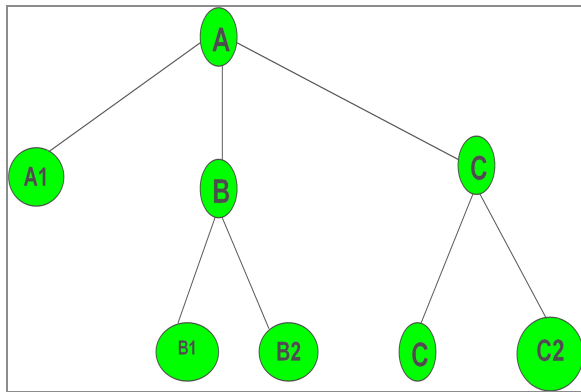

```
<productId>:<priority>,<productId>:<priority>
Example:
<ns0:evaluationPriority>SIM_TECNICO_BP1:1,SIM_TECNICO_BP2:
2</ns0:evaluationPriority>
```

Shared Attributes - Sample Test Scenarios

This section describes the simple cases to test shared attributes in different scenarios.

1. Publish Product Model. The processes must be running in Test harness.

Here is an example for shared attributes with values of one reflecting in the other.



Scenario 1:

For the above product model structure, submit orders for SharedAttribute_B and SharedAttribute_B1. Refer to the [Order Submission](#) topic for more information on the order submission process. Send new value for the shared attribute in the user-defined field format and values for both the attributes.

The value of B must reflect in B1, which conforms to the explanation of the Shared Attributes.

Scenario 2:

Submit order and send new value for the shared attribute (in the user-defined field format). Through order submission, send values for both the attributes, SharedAttribute_B and SharedAttribute_B1. Using SetPlanItemRequest service, set Shared Attributes value of B.

In this case also, the value of B must reflect in B1. Using the GetPlanItemrequest service for B1 returns a new value, which is reflected in B, thus corresponding value and an update in the value of one is reflected in the other.

Intermediate Milestones Dependencies

For a plan item to reach its completion state, the status of all intermediate milestones must be in COMPLETE state.

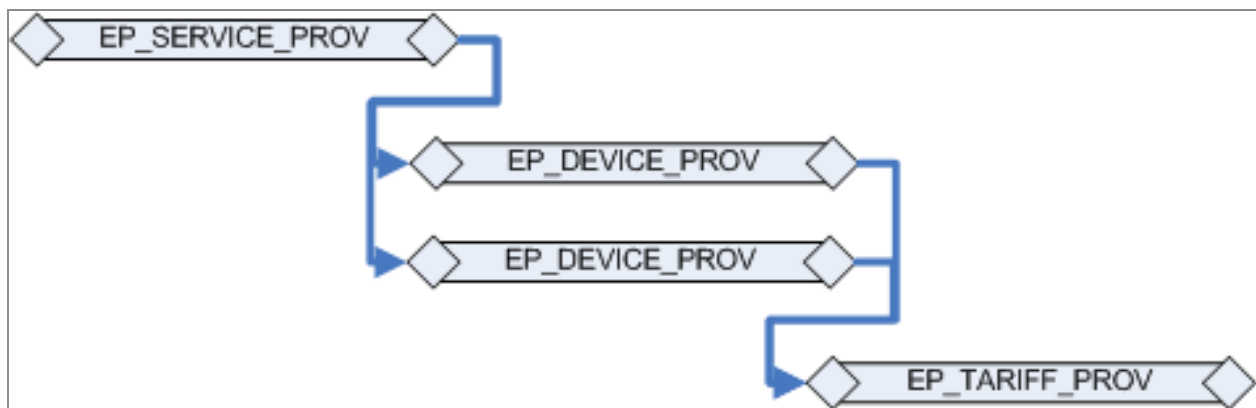
For example, if a plan item (PI-1) has a sequence of START-M1-END, both M1 and END milestones must be in COMPLETE state for PI-1's status to be marked as COMPLETE. Otherwise, the plan item status remains in the EXECUTION state. This ensures that all necessary steps are fully completed before a plan item is considered finished.

The actual fulfillment of a product is done by orchestrating the back-end process components. By default, any process component has two milestones:

1. START
2. END

These milestones represent the starting and the end parts of it. There is a direct dependency between the process components due to sequencing of the products in the catalog. This dependency is of type END-to-START, or once a process component is completely run, then only the dependent process component can start its execution as shown in the following figure:

END-to-START dependency



The process component EP_DEVICE_PROV can start only when EP_SERVICE_PROV is completed and EP_TARIFF_PROV can start only when EP_DEVICE_PROV is completed.

TIBCO Order Management also supports the following complex types of dependencies between the running process components:

- [Milestone to START Dependency](#)

- [END to Milestone Dependency](#)
- [Milestone to Milestone Dependency](#)
- [Milestone without Dependency](#)
- [Conditional Milestones Dependency](#)

These dependencies are supported with the implementation of Intermediate Milestones within the process component in addition to the START and END.

The functionality provides a base behavior that permits plan items to be sequenced corresponding to products related by MDO when:

1. MDO-related product instances have no LINKID defined.
2. MDO-related product instances have LINKID defined and have the same LINKID value.

This feature can extend the base behavior and sequence additionally plan items corresponding to products related by MDO when:

1. MDO-related parent product instance has LINKID defined and child product instances have no LINKID defined.
2. MDO-related parent product instance has no LINKID and child product instances have a LINKID defined.

An MDO-related parent product instance can relate to multiple child product instances using base and extended cases so that the following use cases are possible:

- An MDO-related parent product that has a LINKID defined and is related to a child instance that has the same LINKID defined can be also related to MDO-related child product instances that have no LINKID defined.
- An MDO-related parent product that has no LINKID defined and is related to a child product instance that has no LINKID defined can be also related to MDO-related child product instances that have a LINKID defined.

For a plan item to reach its completion state, the status of all intermediate milestones must be in COMPLETE state.

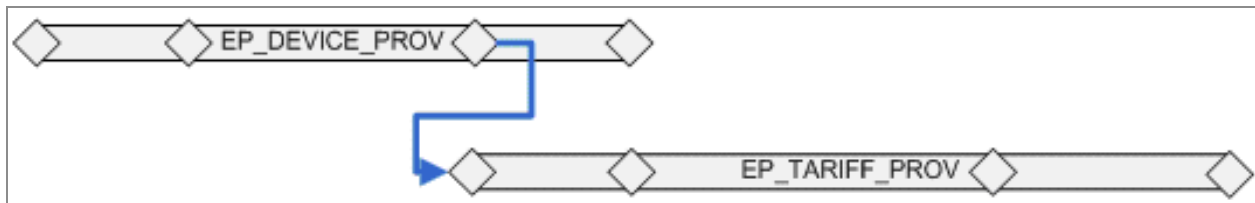
For example, if a plan item (PI-1) has a sequence of START-M1-END, both M1 and END milestones must be in COMPLETE state for PI-1's status to be marked as COMPLETE. Otherwise, the plan item status remains in the EXECUTION state. This ensures that all necessary steps are fully completed before a plan item is considered finished.

Milestone to START Dependency

START milestones of a process component have a dependency on the completion of an intermediate milestone in another process components.

The following figure shows the Milestone to START dependency:

Milestone to START Dependency

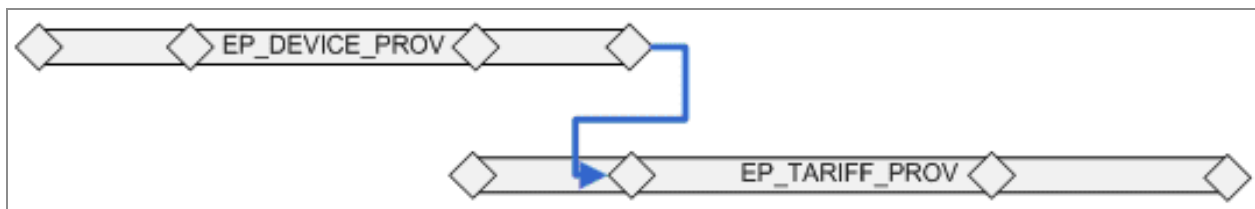


END to Milestone Dependency

Intermediate milestones in a process component have a dependency on the completion of the END milestones in another process components.

The following figure shows the END to Milestone dependency:

END to Milestone Dependency

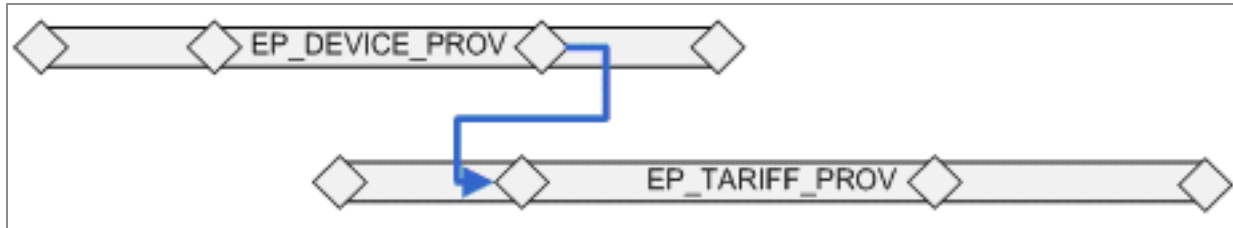


Milestone to Milestone Dependency

Intermediate milestones in a process component has a dependency on the completion of the intermediate milestones in another process components.

The following figure shows the milestone to milestone dependency:

Milestone to Milestone Dependency

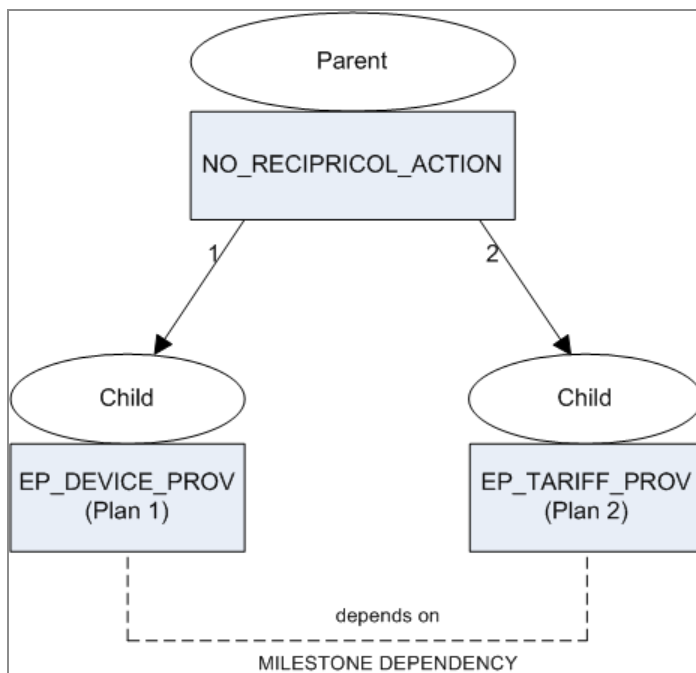


ProductComprisedOf Sequence and Intermediate Milestone Dependency

Sequencing is an indication of the order in which the plan items are executed. If there is not an intermediate milestones dependency set, a default START-END dependency is created. When using the ProductComprisedOf sequence and intermediate milestones at the same time, the intermediate milestone dependencies takes a precedence over the conventional End-->Start dependencies.

The following figure shows a START-->END dependency with an intermediate milestone dependency:

Start Milestone Dependency



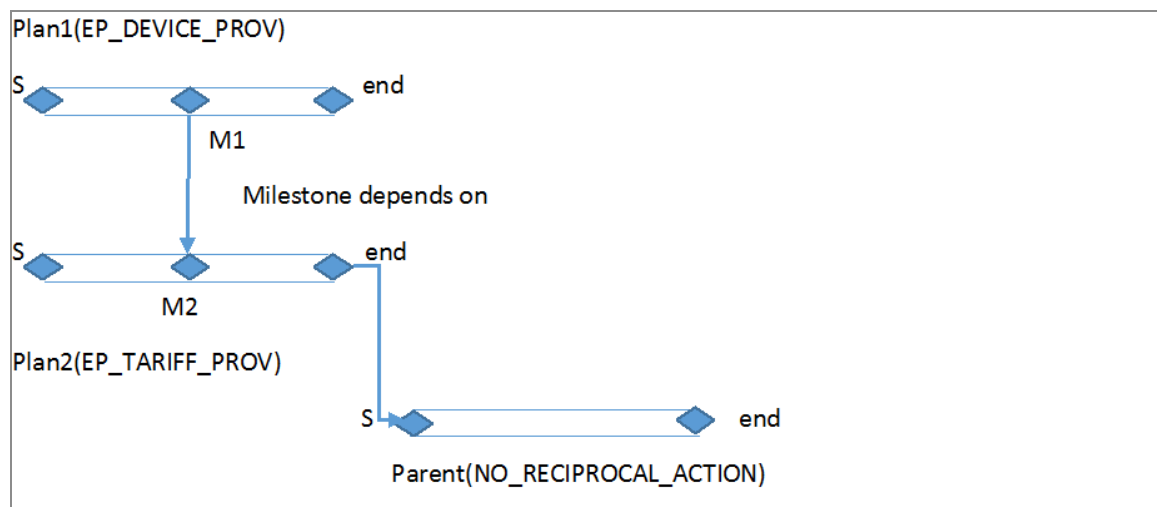
1. The plan for an order is generated in the usual manner. There are no intermediate milestone dependencies yet. The following is the plan based on the sequences:
Start_Plan1_End --> Start_Plan2_End --> Start_NO_RECIPROCAL_ACTION _End.

- The generated plan is evaluated against the intermediate milestone dependencies defined in the product model so as to rearrange the dependencies. In this step as per the product model, the milestone 'Milestone_Dependency_2' of Plan2 depends on the milestone_M1 of Plan1.

Start_Plan1End ---> Milestone DependencyPlan2_End

- There is a check for whether to add the existing dependencies or not. If the intermediate milestone dependency in a plan item is on the same plan fragment as the one in existing dependency, it ignores the existing dependency on that plan fragment, for example, the start of Plan2 does not depend on the end of Plan1 anymore.

The following is the final plan:



Start_Plan1_End ---> Start_Milestone DependencyPlan2_End ---> StartNO_RECIPROCAL_ACTION_End

The intermediate milestone dependencies take a precedence over the conventional End-->Start of dependencies.

Milestone without Dependency

There can be intermediate milestones defined in a process component, which does not have any dependencies. However, milestones in another process component might depend on any of these milestones.

The following figure shows the Milestone without any dependency:

Milestone without Dependency



These types of dependencies help manage the complex order fulfillment process. For instance, start activating the broadband service once the broadband device fulfillment reaches a certain status, say, INSTALLED.

The product model schema has been updated to support the milestones and dependency definitions. See *TIBCO® Fulfillment Catalog Product Catalog Guide* for newly added repository and relationships to support the intermediate milestones dependencies.

Conditional Milestones Dependency

The dependency between intermediate milestones can be conditional. This is specified using the relationship attribute called Condition in TIBCO® Fulfillment Catalog. It is represented in the product model as illustrated below:

Conditional Milestones Dependency

```
<ns0:plan>
  <ns0:name>EP_TEST_PROVIDE_11</ns0:name>
  <ns0:description>Product 11 PROVIDE</ns0:description>
  <ns0:action>PROVIDE</ns0:action>
  <ns0:affinity>false</ns0:affinity>
  <ns0:milestone>
    <ns0:name>START</ns0:name>
    <ns0:dependency>
      <ns0:planName>EP_TEST_PROVIDE_10</ns0:planName>
      <ns0:milestoneName>MILE1</ns0:milestoneName>
      <ns0:type>InLink</ns0:type>
      <ns0:condition>Match:Database=Middleware</ns0:condition>
    </ns0:dependency>
  </ns0:milestone>
</ns0:plan>
```

In this sample, the START milestone of EP_PROVIDE_11 is dependent on MILE1 of EP_PROVIDE_10, only if the specified condition is satisfied.

The condition syntax can be one of the following three types:

- Parent user-defined field Syntax
- Child user-defined field Syntax

- Match Parent-Child user-defined field Syntax

i Note: There is no provision to specify the XSLT statement in the condition as it is there for Attribute-Based Decomposition. Also, the flavor must be configured as `input` instead of `config` to make the conditional milestone dependency function correctly.

Note the following definitions:

- **Parent:** The plan fragment whose milestone has a dependency on another plan fragment. The parent user-defined field is referred to a user-defined field passed in the order line in the order for that product, which is propagated into the plan item for that order line.
- **Child:** The plan fragment on whose milestone a milestone in another plan fragment depends. The child user-defined field is referred to a user-defined field passed in the order line in the order for that product, which is propagated into the plan item for that order line.

In the plan illustrated above, EP_TEST_PROVIDE_11 [START] is dependent on EP_TEST_PROVIDE_10 [MILE1]. It is assigned to PROD11 as the PROVIDE plan fragment. PROD11 is the parent and PROD10 is the child.

Parent user-defined field Syntax

```
Value:Parent(ParentUDFName=ExpectedValue)
```

The condition is satisfied only if there is a user-defined field in the parent plan item with the same value as passed in the condition as "ExpectedValue".

Child user-defined field Syntax

```
Value:Child(ChildUDFName=ExpectedValue)
```

The condition is satisfied only if there is a user-defined field in the child plan item with the same value as passed in the condition as "ExpectedValue".

Match Parent-Child user-defined field Syntax

```
Match:ParentUDFName=ChildUDFName
```


The condition is satisfied only if the value of the user-defined field [ParentUDFName] in the parent plan item is equal to the value of the user-defined field [ChildUDFName] in the child plan item.

Order Amendment

Order Amendment allows the order, which is being fulfilled, to be modified for different parameters such as action, requiredByDate, and User-Defined Fields in the existing order lines. It also allows adding a new order line in the request. The parameters and their reason for change are mentioned as follows:

- The parameter values passed in the original request were incorrect. The corrected values can be passed by sending an amendment request.
- The parameter values require a modification as per the change request from the end user. For example, the bandwidth of a product named Internet Bundle is passed as a user-defined field named Bandwidth in the order line. The bandwidth in the original order was 1 Mbps. When the order was being fulfilled, the customer requested the bandwidth to be updated to 2 Mbps. This is done by sending an amendment request by changing the value of the user-defined field named Bandwidth to 2 Mbps.
- An additional product requires fulfillment when the current one is being fulfilled.

An order can be amended when it is in one of the following states:

FEASIBILITY	If the order is amended in these states, then it is called a <i>pre-plan development</i> amendment, because the execution plan has not yet been created. In this scenario, the execution plan generated for the amendment request is considered and run.
OPD	
PREQUALIFICATIONFAILED	
EXECUTION	If the order is amended in these states it is <i>post-plan development</i> amendment, because the execution plan was already created and had begun execution. In this scenario, the existing plan requires modification and merging with the plan that has been generated as per the amendment request. <i>Post-plan development</i> amendment is the most frequently used amendment.
SUSPENDED	
ERROR_HANDLER	

An order cannot be amended when it is in any of the following final states:

- COMPLETE
- CANCELED
- WITHDRAWN

i Note: If you want to generate a redo plan item without changing any thing in the amendment request, you must set the `enableAmendmentValidation` flag to `false` to bypass any validation. Also, the REDO for the plan item is generated without any changes in amendment. For this, you must set the `EMPR_ACTION_*` with the value as REDO.

Delta Amendment

You can request for the required changes only. You do not have to include the existing orderlines or order details in the request for amendment.

Use Case: 1

You have an order, which contains A, B, and C items on line numbers 1, 2, and 3 respectively.

Here is a requirement to add item D on line number 4.

In this case, you do not have to send the entire order in the request. You can send item D in the request with 'add' as the required action.

Use Case: 2

You have an order, which contains A, B, and C items on line numbers 1, 2, and 3 respectively.

Here is a requirement to modify item B on line number 2.

In this case, you do not have to send the entire order in the request. You can send item B in the request with 'modify' as the required action.

Delta amendment is also available for `requiredByDate` and User-Defined Fields.

Amendment Workflow

Since an order amendment involves the modification of the current execution plan, a predefined process is adopted. The predefined process is as follows:

1. After accepting an order amendment request, the Orchestrator first tries to suspend the current execution plan by sending the suspend requests (`PlanItemSuspendRequest` message) to all the plan items that are in `EXECUTION` state. Based on the implementation of the process components, and the point at which the process component is executed, the process components might send a successful suspend response (`PlanItemSuspendResponse` message) or a successful completion response (`PlanItemExecuteResponse`). Any one of the responses is acceptable by the Orchestrator.
2. Once the execution plan (and order) reaches the `SUSPENDED` state, the Orchestrator sends a plan generation request to Automated Order Plan Development to generate the execution plan as per the order lines in the amendment request.
3. The new execution plan generated by the core Automated Order Plan Development is merged with the existing plan to add, or to modify the plan items as per the changes in the amendment request.
4. Based on the modification rule characteristics defined in the product model, the compensatory plan items are added in the new execution plan to let the undoing of the tasks that were performed by the earlier corresponding plan items. If required, the `REDO` plan items are also added in the new execution plan to redo the tasks that need to be performed by a particular plan item.
5. After receiving the consolidated execution plan for the amendment request from Automated Order Plan Development, the Orchestrator activates the `SUSPENDED` plan and starts orchestrating it as per the latest dependencies.
6. All `SUSPENDED` plan items is activated, either for cancellation (`cancelWithNoRollback` or `cancelAndRollback`) or resume execution (`resumeExecution`) by sending the `PlanItemActivateRequest` messages.
7. Any compensatory and redo plan items, created during the amendment process, is executed in the same way as the regular plan items by sending the `PlanItemExecuteRequest` messages, so as to either complete or cancel the order.

Modeling of the Required Characteristics in the Fulfillment Catalog

As per the requirement of the amendment use case, some or all of the following characteristics are required to be available in the product model published to TIBCO Order Management. The modeling of these characteristics and relating them with the required

products, needs to be done in the TIBCO® Fulfillment Catalog at the design time. For more information about modeling, see the following procedure:



Note:

This section covers just the high level modeling steps specific to the characteristics required for amendments. Refer the TIBCO® Fulfillment Catalog documentation for details.

1. Create the following records in the CHARACTERISTIC repository:
 - EPMR_ACTION_PROVIDE
 - EPMR_ACTION_Cease
 - EPMR_ACTION_UPDATE
 - EPMR_ACTION_WITHDRAW
 - COMPENSATE_PROVIDE
 - COMPENSATE_Cease
 - COMPENSATE_UPDATE
2. For more granular EPMR actions based on the plan item statuses, the user can add additional characteristics mentioned below in generic format. Note that these characteristics are used only in case of the new and improved user-defined field modification functionality. Refer the New Characteristics subsection in OrderLine user-defined field change.
 - EPMR_ACTION_<<action>>_UDF_CHANGE_<<Plan Item Status>>. For example, EPMR_ACTION_PROVIDE_UDF_CHANGE_SUSPENDED.
3. Create the Characteristic relationship between the records in the PRODUCT repository and one or more EPMR_ACTION_* records in the CHARACTERISTIC repository mentioned in point 1 and 2 above. The logically valid value for the RelationshipValue attributes in all these Characteristic relationships can be one of the four EPMR actions - COMPENSATE, RESTART, COMPENSATE_RESTART, or IGNORE. Refer the [Execution Plan Modification Rules \(EPMR\)](#) topic for the significance of these four actions. For example, the technical product Router can have a Characteristic relationship with EPMR_ACTION_PROVIDE characteristic, with the value of RelationshipValue attribute as COMPENSATE_RESTART.
4. Create the Characteristic relationship between the records in the PRODUCT repository and one or more COMPENSATE_* records in the CHARACTERISTIC repository

mentioned in point 1 above. The logically valid value for the RelationshipValue attributes in all these Characteristic relationships must be the ID of the plan fragment record that is required to be run as the compensation task for the corresponding action. For example, the technical product Router can have a Characteristic relationship with COMPENSATE_PROVIDE characteristic, with the value of the RelationshipValue attribute as the planFragmentID Router_Cancel.

Types of Amendment

At a high level, order amendments can be classified into the two heads and further into each subtype as described below:

1. Changes at order line level
 - a. Action Change - Changing the action in one or more order lines.
 - b. RequiredByDate Change - Changing the requiredByDate in one or more order lines.
 - c. User-defined field Change - Changing the user-defined field in one or more order lines.
2. Changes at the order header level
 - a. RequiredByDate Change - Changing the requiredByDate at order header level.
 - b. OrderLine Addition - Adding a new order line in the order.
 - c. OrderPriority Change - Changing the orderPriority at the order header level.

The amendment types at the order line level are MUTUALLY EXCLUSIVE. Only one of the amendment types is allowed for a particular order line in an amendment request. For example, the action and User-Defined Fields cannot be simultaneously updated in an order line. If multiple amendment types are tried simultaneously for an order line in a single amendment request, an exception with the appropriate code and an appropriate message is generated. For example, if action and requiredByDate both are changed in an order line in the amendment request, an exception with the error code "TIBCO-AFF-OMS-100064" and an error message "Action and requiredByDate cannot be modified simultaneously in an order line", is generated.

However, different amendment types can be applied in different order lines as part of the same amendment request. That is, the action change in one order line and the user-defined field or requiredByDate change in another one can be done simultaneously.

OrderLine Action Change

In this amendment type, the fulfillment action in an order line can be changed as part of the amendment request. For example, the action was PROVIDE in an order line in the original order request and changed to UPDATE in the amendment request. Since CANCEL can also be passed as the action in one or all the order lines in the amendment request, order line cancellation and entire order cancellation are the subtypes of this amendment type.

OrderLine Cancellation

To cancel a particular order line, CANCEL must be passed as the action in the amendment request. The PENDING plan items associated with the order line are directly CANCELLED. Execution Plan Modification Rules are applied on the plan items that were COMPLETE or SUSPENDED before the amendment to compensate them, as per the Execution Plan Modification Rules action defined in the product model. Once all the associated plan items are either cancelled or compensated, the order line is marked as canceled by changing its status to CANCELLED.

Entire Order Cancellation

To cancel the entire order, the CANCEL must be passed as the action in all the order lines in the amendment request. All the PENDING plan items in the execution plan are directly CANCELLED. Execution Plan Modification Rules are applied on the plan items that were COMPLETE and SUSPENDED before the amendment to compensate them, as per the Execution Plan Modification Rules action defined in the product model. Once all the plan items are either canceled or compensated, all the order lines and also the order is marked as canceled by changing the statuses to CANCELLED. The Execution Plan Modification Rules are applied in case of order line or entire order cancellation, based on the Boolean value (true/false) of the rollback user-defined field passed in the order header. The modification rules are applied if the rollback UDF's value is true, otherwise it is not applied.

**Note:**

The default behavior is always to roll back, which is, if the rollback user-defined field is not passed in the order header, it is considered to be true.

An order can also be canceled using the CancelOrderRequest SOAP service and from the Order Management Server user interface.

Preconditions for Action change

Following are the preconditions for the order line action change amendment type.

1. The number of order lines in the amendment request must match with those in the original order request.
2. The lineID, productID, requiredByDate, and User-Defined Fields in all the order lines in the amendment request must match with those in the original order request.

When the fulfillment action in an order line is changed, the plan items associated with that order line in the existing plan are handled in different ways.

1. The action in the amendment request is set as the fulfillment action in all PENDING plan items.
2. Execution Plan Modification Rules (EPMR) are applied to all SUSPENDED or COMPLETE plan items to take the appropriate actions on these plan items such as compensating the earlier tasks and/or redoing the tasks from the beginning.

For any action change in the amendment request other than CANCEL, the Execution Plan Modification Rules characteristic corresponding to the action in the original plan item, from the product being fulfilled by that plan item, is considered when applying the modification rule.

OrderLine Action in Original Request	OrderLine Action in Amendment Request	Execution Plan Modification Rules Characteristic Considered
PROVIDE	Any, except for CANCEL and PROVIDE	EPMR_ACTION_PROVIDE
UPDATE	Any, except for CANCEL and UPDATE	EPMR_ACTION_UPDATE
CEASE	Any, except for CANCEL and CEASE	EPMR_ACTION_Cease

If CANCEL is passed as the order line action in the amendment request, the EPMR_ACTION_WITHDRAW characteristic from the product being fulfilled by the corresponding plan item is considered always, regardless of the action in the original request.

In addition to the above condition, if the plan item action is CANCEL, the EP_MR_ACTION_WITHDRAW characteristic would be considered. For example: In case of amendment due to a certain condition if any plan item is not present in the newly generated plan, then the action for that particular plan item is considered as CANCEL.

Based on the value of the Execution Plan Modification Rules characteristic that was considered, the modification rules are applied on the required plan items. See topic [Execution Plan Modification Rules \(EPMR\)](#) to understand the effect of each action.

i Note: If the Execution Plan Modification Rules characteristic to be considered (Example: EP_MR_ACTION_PROVIDE) is not present in the corresponding product model. COMPENSATE_RESTART is considered as the default EPMR action, only if the flag CompensateRestartForNoEPMRChar is set in Automated Order Plan Development configurations. See the topic [Amendment Configuration Flags](#) to understand the significance of each flag.

Once the EPMR action is applied on all the required plan items and the compensatory and/or redoes plan items are generated, the dependencies in the parent plan items are updated appropriately. See topic [Impact on Dependencies](#) to understand how the dependencies are modified. The amendment plan is sent out to the Orchestrator to fulfill the order sent in the amendment request.

RequiredbyDate Change

RequiredByDate for an order defines the time at which the order plan must be executed. It can be mentioned at both the order header level or/and the order line level. In terms of dependency in the order plan, it generates a time dependency (with absolute time) for a plan item along with dependency on other executing plan items (point dependency) if any. Once the absolute time is reached, time dependency is considered as satisfied.

Following are the preconditions for the order line requiredByDate change amendment type:

1. The number of order lines in the amendment request must match with those in the original order request.
2. The lineID, productID, action, and User Defined Fields in all the order lines in the amendment request must match with those in the original order request.

Following is the process of calculating a time dependency about requiredByDate.

- If requiredByDate is set on the order level only, the start time dependency applies to

all plan items with no leading dependencies

- If requiredByDate is set on the order line level only, the start time dependency applies to plan items for that order line
- If requiredByDate is set on the order header level and on the order line level, the following behavior applies:
 - If requiredByDate in Order Header is later than requiredByDate in order line, then the start time used is the one at order level
 - If requiredByDate in Order Header is earlier than requiredByDate in line item, then the start time used is the one at order line level

RequiredBydate Amendment type allows for changing the required date for an order when it is not in its FINAL stages as mentioned earlier. The following matrix defines the conditions to identify a RequiredByDate change amendment type:

Original header date	Original line date	New header date	New line date	IsAmendment
Past Dated	Past Dated	past dated but greater than originalheader date	past dated but greater than originalheader date	No
Past Dated	Past Dated	Same as original	Future Dated	Yes, for that particular Order Line
Past Dated	Past Dated	Future Dated	Same as original	Yes, for all order lines
Future Dated	Past Dated	Back Dated	Same as original	Yes, for all order lines
Future Dated	Past Dated	Future date than original	Same as original	Yes, for all order lines
Past Dated	Future Dated	Same as original	Same as original	No
Past Dated	Past Dated	Future Dated	Future Dated	Yes, for all order lines. The

Original header date	Original line date	New header date	New line date	IsAmendment
				time dependency is calculated as explained earlier.
No Date	Past Date	Back dated	Same as original	No
No Date	Future Date	Back dated	Same as original	No
No Date	No Date	Future Dated	Future Dated	Yes, for all order lines. The time dependency is calculated as explained earlier.

The default behavior in 2.1.1 for required by date change is not to create compensation or restart any plan items. Below matrix defines the amendment behavior based on plan item status

Plan Item Status	Description
Pending	Plan item dependency time is updated so the plan item triggers at the amended required by date.
Suspended	Not permitted. Any required by date changes are ignored. As the plan item is already started, it is not possible to change the start date.
Complete	Not permitted. Any required by date changes are ignored. As the plan item is already completed, it is not possible to change the start date.

The value of roll back user-defined field in order is ignored in this case as no compensation or restart plan items are created.

OrderLine user-defined field Change

OrderLine user-defined field change is a type of an order amendment where the amended order lines contain changed user-defined fields and/or newly added user-defined fields along with the user-defined fields from the original order request. All other order request attributes remain unchanged. This application can identify if the orders have changed only about User-Defined Fields by inspecting the order lines to identify if the User-Defined Fields have been modified or added.



Note: If you want to modify the user-defined fields and want to avoid your order going through the complex amendment process, then you should consider using the Data Service.

Sample Order Line in TIBCO Order Management

```
<ord1:line>
  <ord1:lineNumber>1</ord1:lineNumber>
  <ord1:productID>MODEM</ord1:productID>
  <ord1:productVersion>1.0</ord1:productVersion>
  <ord1:quantity>1</ord1:quantity>
  <ord1:uom>UOM</ord1:uom>
  <ord1:action>PROVIDE</ord1:action>
  <ord1:requiredByDate>2011-04-30T13:20:00-
05:00</ord1:requiredByDate>
  <ord1:udf>
    <ord1:name>Region</ord1:name>
    <ord1:value>Asia</ord1:value>
  </ord1:udf>
</ord1:line>
```

Identifying user-defined field Amendment

TIBCO Order Management checks the following conditions to identify the user-defined field change amendment scenario. All the following conditions, which are mentioned, hold true for the user-defined field change amendment:

- The number of order lines in the initial order must match the number of order lines in the amended order.
- The product Id in order line in the initial order must match with the product Id in the

corresponding order line of the amended order.

- The action in order line in the initial order must match with the action in the corresponding order line of the amended order.
- The RequiredByDate in order line in the initial order must match with the RequiredByDate in the corresponding order line of the amended order.

Execution Plan Modification Rules (EPMR) Characteristics

The application provides more granular execution plan modification rules actions to be configured for user-defined field modifications based on the status of the plan items, to have more control when generating the COMPENSATE or REDO plan items.

The format of execution plan modification rule characteristics is as follows:

1. EPMR_ACTION_<<action>>_UDF_CHANGE: Using this format Execution Plan Modification Rules action can be configured per amendment type. The supported values of <<action>> are:
 - a. PROVIDE
 - b. CEASE
 - c. UPDATE
 - d. WITHDRAW

The following is an example of the characteristic configured in the product model with Execution Plan Modification Rules:

```
<ns0:characteristics>
  <ns0:name>EPMR_ACTION_PROVIDE_UDF_CHANGE</ns0:name>
  <ns0:description>Characteristic</ns0:description>
  <ns0:instanceOptional/>
  <ns0:instanceCeaseSequence/>
  <ns0:instanceUpdateSequence/>
  <ns0:instanceSequence/>
  <ns0:instanceMin>0</ns0:instanceMin>
  <ns0:instanceMax>0</ns0:instanceMax>
  <ns0:evaluationPriority/>
  <ns0:value>
    <ns0:type>PROVIDE</ns0:type>
    <ns0:discreteValue>COMPENSATE_RESTART</ns0:discreteValue>
    <ns0:mandatoryValue>true</ns0:mandatoryValue>
  </ns0:value>
</ns0:characteristics>
```

```

    </ns0:value>
    <ns0:simpleRule>
      <ns0:name>EPMR_ACTION_PROVIDE_UDF_CHANGE</ns0:name>
      <ns0:ruleSetOutcome>Characteristic</ns0:ruleSetOutcome>
    </ns0:simpleRule>
  </ns0:characteristics>

```

2. EPMR_ACTION_<<action>>_UDF_CHANGE_<<Plan Item Status>>: Using this format, the Execution Plan Modification Rules action can be configured per Amendment Type and Plan Item Status. The supported values of Plan Item Status are:
 - a. COMPLETE
 - b. SUSPENDED
 - c. PENDING
 - d. EXECUTION

The following is an example of the characteristic configured in the product model with Execution Plan Modification Rules:

```

<ns0:characteristics>
  <ns0:name>EPMR_ACTION_PROVIDE_UDF_CHANGE_
  SUSPENDED</ns0:name>
  <ns0:description>Characteristic</ns0:description>
  <ns0:instanceOptional/>
  <ns0:instanceCeaseSequence/>
  <ns0:instanceUpdateSequence/>
  <ns0:instanceSequence/>
  <ns0:instanceMin>0</ns0:instanceMin>
  <ns0:instanceMax>0</ns0:instanceMax>
  <ns0:evaluationPriority/>
  <ns0:value>
    <ns0:type>PROVIDE</ns0:type>
    <ns0:discreteValue>COMPENSATE_RESTART</ns0:discreteValue>
    <ns0:mandatoryValue>true</ns0:mandatoryValue>
  </ns0:value>
  <ns0:simpleRule>
    <ns0:name>EPMR_ACTION_PROVIDE_UDF_CHANGE</ns0:name>
    <ns0:ruleSetOutcome>Characteristic</ns0:ruleSetOutcome>
  </ns0:simpleRule>
</ns0:characteristics>

```

Backward Compatibility with TIBCO Order Management

TIBCO Order Management supports the use of `MODIFICATION_IDNETIFYING_ATTR` udf to denote the user-defined field being changed through the use of a flag. This flag, called `EnableModificationIdentifyingAttribute`, can be configured from the Configurator UI for the AOPD service application.

The default value of this flag is `FALSE`.

Predefined User-Defined Fields

Changes in the following user-defined fields are ignored by the application:

- `ORDERLINE`
- `GLOBAL_PRODUCT_NAME`
- `EOL`
- `ACTION`
- `M_EPS_UDFS`

i Note: The changes done only in the User-Defined Fields at the order header level in an amendment request does not have any impact on the existing plan in terms of the creation of compensatory and redo plan items. There are no changes in the dependencies between the plan items either. However, the amendment plan contains the updated value of the User-Defined Fields. The plan items, which go into the `EXECUTION` post amendment can get the updated value of the header level User-Defined Fields using the `GetPlan JMS` data interface or the `GetOrderExecutionPlan` service.

OrderLine Addition

In this amendment type, one or more new order lines can be added as part of the amendment request to fulfil the additional products. The plan items corresponding to the newly added order lines are added into the execution plan and the dependencies are updated accordingly. At a high level, there are two main cases.

1. If an optional child product from a `ProductComprisedOf` relationship was ordered in the original request and the parent product is then ordered in a new order line in the

amendment request, the newly created plan item for parent products have a dependency on the existing plan item of the child product.

For example, if B is an optional child product of A, in case of the above mentioned scenario; the newly added plan item for A have a dependency on the plan item of B, which was there in the original plan. This is done regardless of the status of the plan item for child product B.

This is logical and is the case even when both products were ordered in the original request itself. Once the amendment plan is activated, the plan item for the parent product goes into EXECUTION after the plan item for child product is successfully completed. If the plan item for child product was already COMPLETE, the plan item for parent product goes into EXECUTION immediately.


2. On the other hand, if a parent product from a ProductComprisedOf relationship was ordered in the original request and the child product is then ordered in a new order line in the amendment request, the dependency of the child plan item is added based on the status of the parent plan item, as explained in the following points:
 - a. If the parent plan item was PENDING before the amendment, the dependency is added into the existing parent plan item.
 - b. If the parent plan item was SUSPENDED or COMPLETE before the amendment, a REDO plan item is generated against it. The original parent plan item is cancelled if it was SUSPENDED. A REDO plan item is generated based on the Execution Plan Modification Rules action configurations as mentioned in the following points:
 - i. If the value configured in the Execution Plan Modification Rules characteristic for the corresponding order line action is either RESTART or COMPENSATE_RESTART. E.g. In case of PROVIDE action, the value of EPMR_ACTION_PROVIDE characteristic is referred.
 - ii. Or the required Execution Plan Modification Rules characteristic is missing in the product model and the CompensateRestartForNoEPMRChar flag is enabled in Automated Order Plan Development configurations.

See [Execution Plan Modification Rules \(EPMR\)](#) for details about Execution Plan Modification Rules actions. The dependency of the child plan item is added into the newly created REDO plan item for the parent product. Once the amendment plan is activated, the newly added plan item for the child product goes into EXECUTION. After it is successfully completed, the REDO plan item for parent goes into EXECUTION.

Preconditions for OrderLine Addition

Following is the only precondition for the order line addition amendment type.

The lineID, productID, requiredByDate, and User Defined Fields in all the order lines in the amendment request must match with those in the original order request.

 **Note:** Unlike addition, the deletion or removal of an existing order line is not allowed and supported in an amendment request. For cancelling the fulfillment of the product in an order line, the order line action must be changed to CANCEL instead.

Execution Plan Modification Rules (EPMR)

The execution plan, for the amendment types mentioned earlier, is modified based on the predefined rules that are specified as values in the following characteristics in the product model:

1. EPMR_ACTION_PROVIDE
2. EPMR_ACTION_Cease
3. EPMR_ACTION_UPDATE
4. EPMR_ACTION_WITHDRAW

Only one of the appropriate characteristics, based on the action passed in the original order, is referred to when applying the modifications on the execution plan. For user-defined field change functionality, additional set of characteristics can be defined to have a granular control based on the status of the plan item.

As mentioned in earlier, these rule actions are applied on the plan items that are either in SUSPENDED or COMPLETE state. There are four standard Execution Plan Modification Rules actions, which are explained in the following paragraphs. Only one of the four actions can be specified as the value for a particular Execution Plan Modification Rules characteristic for a particular product.

COMPENSATE_RESTART

This Execution Plan Modification Rules action is assigned as the value of an Execution Plan Modification Rules characteristic if a compensatory and redo plan item is to be created

against an existing plan item as a part of the amendment processing.

Compensatory Plan Item

The purpose of a compensatory plan item is to compensate, or reverse, the tasks that were done by the existing plan item before the amendment request was initiated. The important aspect of a compensatory plan item is as follows:

1. The planItemId of the compensatory plan item is derived using the planItemId of the existing plan item and has the following format: COMP-<amendment number>_<planItemId of the existing plan item>. For example, if the planItemId of the existing plan item is 04ceddc8-60fc-4800-82b9-4f3382400000 and a compensatory plan item is created against it during the first amendment request, the planItemId assigned to that compensatory plan item is COMP-1_04ceddc8-60fc-4800-82b9-4f3382400000.
2. The action and planFragmentUniqueID (processComponentID) in the compensatory plan item is assigned on the basis of the action in the existing plan item, which is described in the following table:

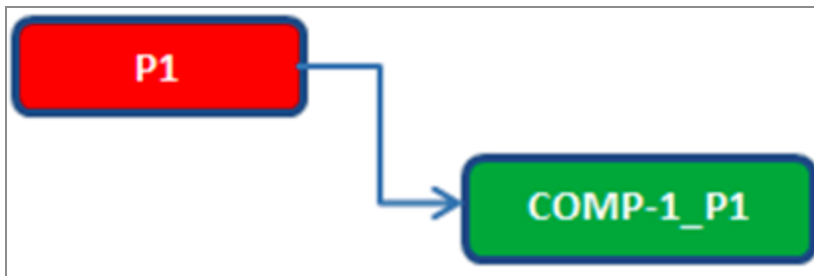
Action in Existing Plan Item	Action in Compensatory Plan Item	processComponentID in Compensatory Plan Item
PROVIDE	CEASE	Value of Characteristic COMPENSATE_PROVIDE
UPDATE	UPDATE	Value of Characteristic COMPENSATE_UPDATE
CEASE	PROVIDE	Value of Characteristic COMPENSATE_Cease



Note: If the required COMPENSATE_<ACTION> characteristic (for example COMPENSATE_PROVIDE) is not present in the product model, the regular plan fragment specified for CANCEL action is assigned.

3. The compensatory plan item, by default, have a simple END-START point dependency on the existing plan item being canceled, as shown in the following figure. This is to ensure that the compensatory task must be started only after the existing task, being run, is activated and canceled.

Dependency between the compensatory plan item COMP-1_P1 and the existing plan item P1 that is canceled



To enable the backward compatibility of having no dependency in the compensatory plan items in TIBCO Fulfillment Order Management 6.0, the flag `com.tibco.af.aopd.flags.amendment.noDependencyInCOMPPlanItems` must be set in Automated Order Plan Development configurations. Refer to the topic [Amendment Configuration Flags](#) to understand the significance of each flag.

4. The action and the processComponentID in the existing plan item is set to CANCEL and NO_RECIPROCAL_ACTION respectively to cancel the existing plan item. Note that the Orchestrator changes the processComponentID to NO_RECIPROCAL_ACTION only for the PENDING plan items that are canceled. The processComponentID for the SUSPENDED plan items remains the same.

i Note: A compensatory plan item, if requiring creation, is created always against a regular plan item in the first amendment. However, in the subsequent amendment requests, it can be created against a regular or a REDO plan item from the earlier amendment based on the execution plan at that point, as shown in the following figure. A compensatory plan item is never created against another compensatory plan item that was created during the last amendment.

Dependency between the compensatory plan item COMP-2_REDO-1_P1 created during the second amendment and the REDO plan item from the last amendment REDO_P1, which is canceled.

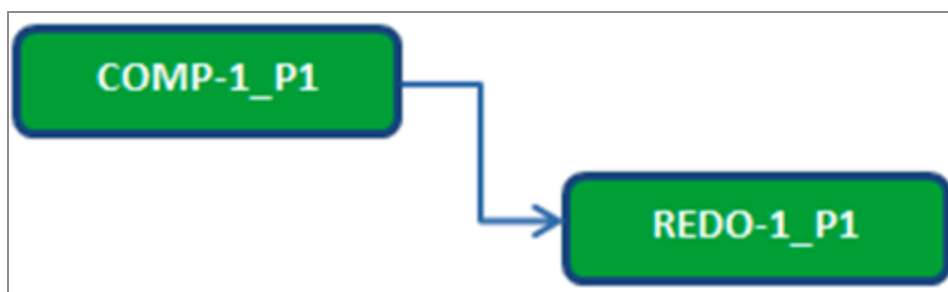


Redo Plan Item

The sole purpose of redo (restarting) plan item is to restart or redo the tasks that were supposed to be done in the existing plan item before the amendment request was initiated. The important aspects of redo plan item are as follows:

1. The planItemId of redo plan item is derived using the planItemId in the existing plan item and has the following format: REDO-<amendment number>_<planItemId of the existing plan item>. For example, if the planItemId of the existing plan item is 04ceddc8-60fc-4800-82b9-4f3382400000 and redo plan item is created against it during the first amendment request, the planItemId assigned to that redo plan item is REDO-1_04ceddc8-60fc-4800-82b9-4f3382400000.
2. The action in redo plan item is the one that is passed in the corresponding order line in the amendment request.
3. The planFragmentUniqueID (processComponentID) in redo plan item is the same as the one in the existing plan item, except in case of order line action change amendments. In such cases, the plan fragment associated with the action in the amendment request is assigned in redo plan item.
4. Redo plan item always have a simple END-START point dependency on the compensatory plan item that is created, as shown in the following figure. This ensures that the designated tasks are restarted only after the compensatory tasks are finished.

Dependency between the REDO plan item REDO_P1 and the compensatory plan item COMP-1_P1



- i Note:** redo plan item, requiring creation, is always created against a regular plan item in the first amendment. However, in the subsequent amendment requests, it can be created against a regular or a REDO plan item from the earlier amendments based on the execution plan at that point, as shown in the following figure. redo plan item is never created against a compensatory plan item that was created during the last amendment.

Dependency between redo plan item REDO-2_REDO-1_P1 created during the second amendment and the REDO plan item from the last amendment REDO_P1, which is canceled



In case of OrderLine cancellation or Entire Order Cancellation, even if the Execution Plan Modification Rules action is COMPENSATE_RESTART, only the compensatory plan item is created. There is no need to create redo plan item on the order line, or the entire order is canceled.

- i Note:** RESTART is not a logical Execution Plan Modification Rules action in case of an order line or an entire order cancellation. If RESTART action is encountered when processing the order line or the order cancellation, no action is taken on the corresponding plan item. A relevant message is logged, instead, to report the same.

COMPENSATE

This Execution Plan Modification Rules action is assigned as the value of an Execution Plan Modification Rules characteristic if only a compensatory plan item is to be created against an existing plan item as a part of the amendment processing.

See [Compensatory Plan Item](#) for understanding all the aspects of a compensatory plan item.

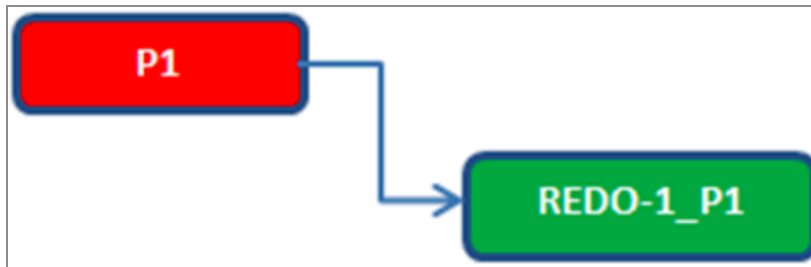
RESTART

This Execution Plan Modification Rules action is assigned as the value of an Execution Plan Modification Rules characteristic if only a redo plan item is created against an existing plan item as a part of the amendment processing.

See [Redo Plan Item](#) for understanding all the aspects of a redo plan item.

The redo plan item always have a simple END-START point dependency directly on the existing plan item that is going to be activated and canceled, due to the non-existence of a compensatory plan item as shown in the following figure. This is to ensure that the designated task must be restarted only after the cancellation of the existing task.

Dependency between the REDO plan item REDO_P1 and the existing plan item P1, which is canceled.



IGNORE

This Execution Plan Modification Rules action is assigned as the value of an Execution Plan Modification Rules characteristic, if no explicit action is required to be taken against an existing plan item as part of the amendment processing. In case of OrderLine cancellation or Order cancellation, the planFragmentUniqueID (processComponentID) of the plan item is set to NO_RECIPROCAL_ACTION.

No Execution Plan Modification Rules Characteristic in Product

In case a product model does not contain the required Execution Plan Modification Rules characteristic, then behavior of amendment to generate redo or compensate plan item can be controlled using the flag, CompensateRestartForNoEPMRChar, in Automated Order Plan Development configurations. See the topic [Amendment Configuration Flags](#) for this flag.

Amendment Configuration Flags

The following are the flags available in Automated Order Plan Development configurations to tweak some of the functionalities around order amendments:

1. EnableModificationIdentifyingAttribute

This flag, if true, enables the OrderLine user-defined field modification functionality using the MODIFICATION_IDENTIFYING_ATTR characteristic as it was in 2.0.x.

2. NoDependencyInCOMPPlanItems

This flag, if true, enables the backward compatibility to 2.0.x of having no dependency of the existing plan item being cancelled, in the compensatory plan item. The compensatory plan item immediately goes into execution along with the activation request of the existing plan item.

3. EnableDateShiftCompRedo

This flag, if true, enables the backward compatibility to 2.0.x version of creating compensatory and redo plan items in case of requiredByDate change (Date Shift) type amendments. This value of roll back user defined field controls the behavior at runtime. The default value of roll back is true and the behavior is:

- Compensation and Restart plan items is created as per the Execution Plan Modification Rules characteristics for suspended and completed plan items.
- The original completed and suspended plan items do not have the new requiredByDate. New requiredByDate (date shift) is set for the corresponding “Redo” plan items.
- In case of pending items, the requiredByDate of that pending plan item is changed to the new requiredByDate. No Compensate or Redo Plan items are generated.

If mentioned as false, then

- Compensation and restart plan items are not created.
- Completed and suspended plan items do not contain the changed requiredByDate. Only pending plan items have the new date.

The behavior of requiredByDate amendments for compensation and restart plan items for Execution Plan Modification Rules characteristics is consistent with implementation of other amendment types configured for 2.0.x in this release.

4. CompensateRestartForNoEPMRChar

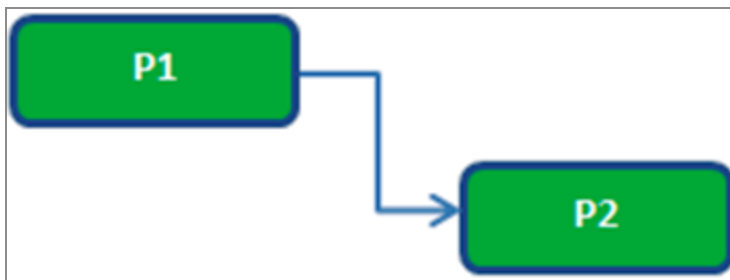
This flag, if true, considers COMPENSATE_RESTART as the Execution Plan Modification Rules action in case of the required Execution Plan Modification Rules characteristic not present in the product model. If this flag is false and the required Execution Plan Modification Rules characteristic is also not present in the product model, no action is taken on the plan items associated with that product, which are in COMPLETE or SUSPENDED state.

Impact on Dependencies

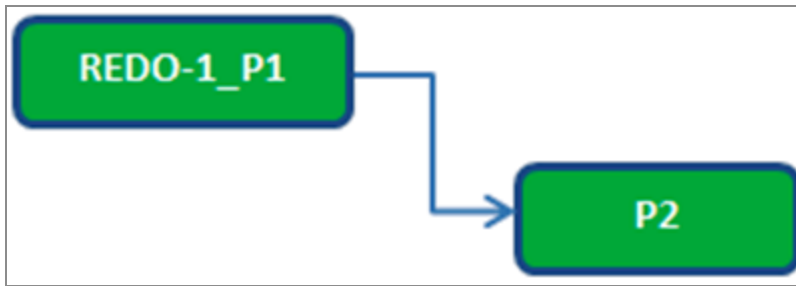
If both compensatory plan items and redo plan items or either of them are created against one or more existing plan items when processing an amendment request, the dependencies in the overall execution plan are impacted. The dependency on the existing plan item being cancelled is implicitly added in the compensatory and/or redo plan item when they are created. There can be some additional dependencies in the redo plan items in certain cases. Also, the dependencies in other regular plan items are modified, if required. The following points explain these modifications:

1. If a parent plan item in PENDING state, which is not being cancelled, has a dependency on such a child plan item against which a COMP and REDO plan items have been created, the dependency on the existing plan item is removed and a new dependency is added on the REDO plan item, as shown in the following figures. The REDO plan item has higher priority over the COMP plan item when replacing the dependency on the corresponding existing plan item. If the REDO plan item does not exist, the dependency on the existing plan item is replaced with a dependency on the COMP plan item. This keeps the dependency structure in the amendment plan in-line with the earlier plan.

Dependency on plan item P1 in PENDING plan item P2 in the original plan

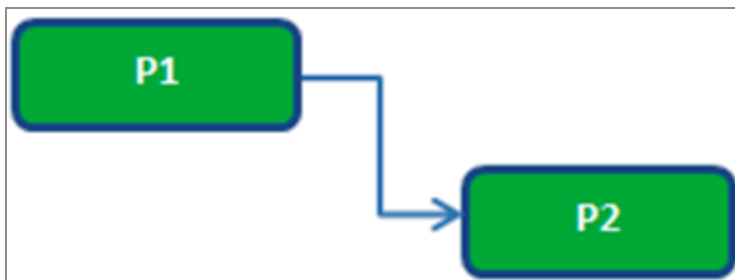


Dependency on the first level REDO plan item of P1 in PENDING plan item P2 in the amendment plan

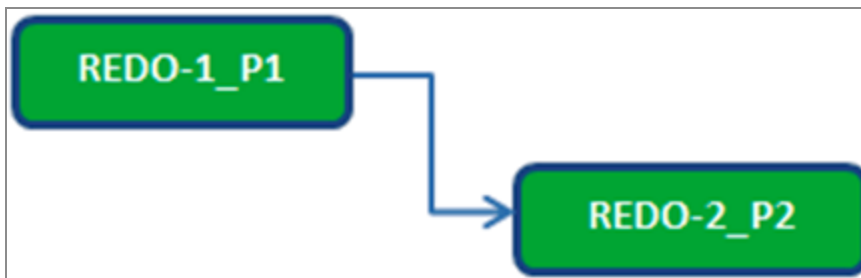


2. If REDO plan items have been created against an existing parent and child plan item, then the same dependency is maintained between the corresponding REDO plan items. The parent REDO plan item has a dependency on the child REDO plan item, in addition to the dependency on the existing original plan item being cancelled, as shown in the following figures:

Dependency on plan item P1 in plan item P2 in the original plan

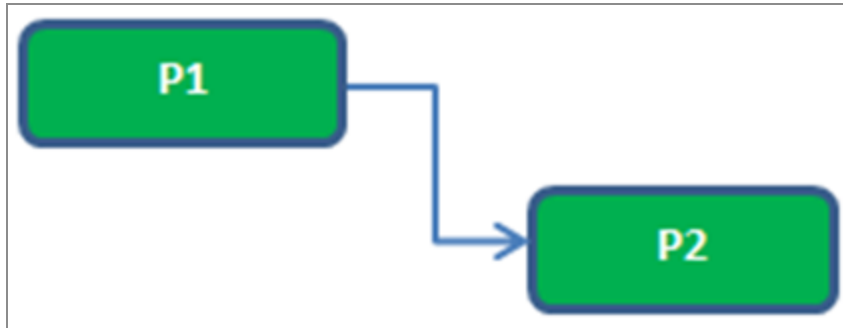


Dependency on REDO plan item P1 in REDO plan item P2 in the amendment plan

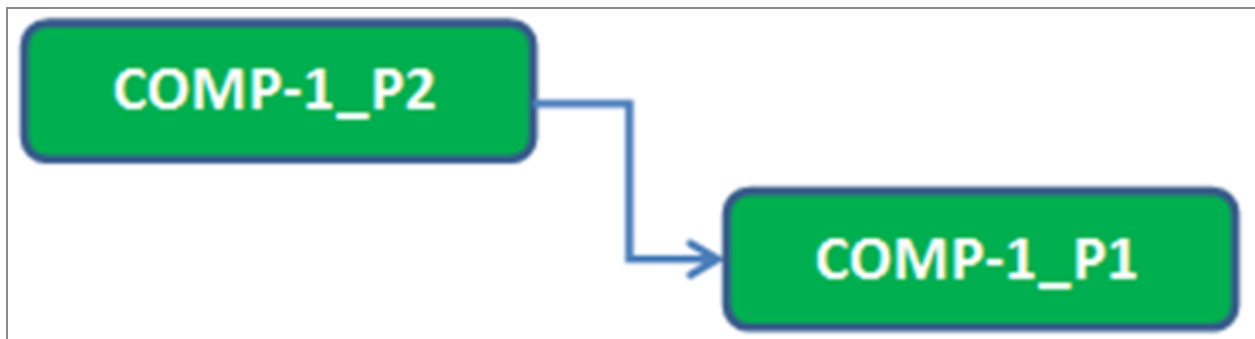


3. If COMP plan items have been created against an existing parent plan item and child plan item, a reverse dependency is maintained between the corresponding COMP plan items in the amendment plan. The child COMP plan item has a dependency on the parent COMP plan item, as shown in the figure. It is done to ensure that the exact compensation of the plan items, which is, the compensation of parent plan item occurs first, followed by the compensation of child plan item.

Dependency on plan item P1 in plan item P2 in the original plan



Dependency on COMP plan item P2 in COMP plan item P1 in the amendment plan



Multiple Amendments

TIBCO Order Management allows multiple amendments of an order however not concurrently. This means that the order, which is being amended cannot be amended again at the same time. Once the existing amendment request is successfully processed by TIBCO Order Management and the new plan is activated, the order can be very well amended again, provided the amendment conditions are satisfied. Each amendment request for an order is processed in the same way as explained in the section [Amendments Workflow](#).

The planItemId assigned to a compensatory or redo plan item created during an amendment contains the amendment number as one of the prefixes. See [Compensatory Plan Item](#) and [Redo Plan Item](#) for more information about the planItemId format.

Custom Action

You can define the set of actions to provide a way to define any number of unique fulfillment actions.

Custom actions are loaded into Automated Order Plan Development as Action Models and is referred to at the time of plan generation.

Custom action enables you to submit an order for custom actions, depending on which Automated Order Plan Development assigns the planfragment.

i Note: The planfragment is selected based on the PlanFragmenthasCustomAction relationship from the product datamodel.

Product Id and Product Id Ext.

You can now use the same unique productID for two different products. Previously, a unique productID was assigned to each product.

You can differentiate the two products by using the PRODUCTIDEXT characteristic.

Example:

Product 1: Value of productID is 'A' but has no PRODUCTIDEXT.

Product 2: Value of productID is 'A' and PRODUCTIDEXT is 1.

Jeopardy Management System

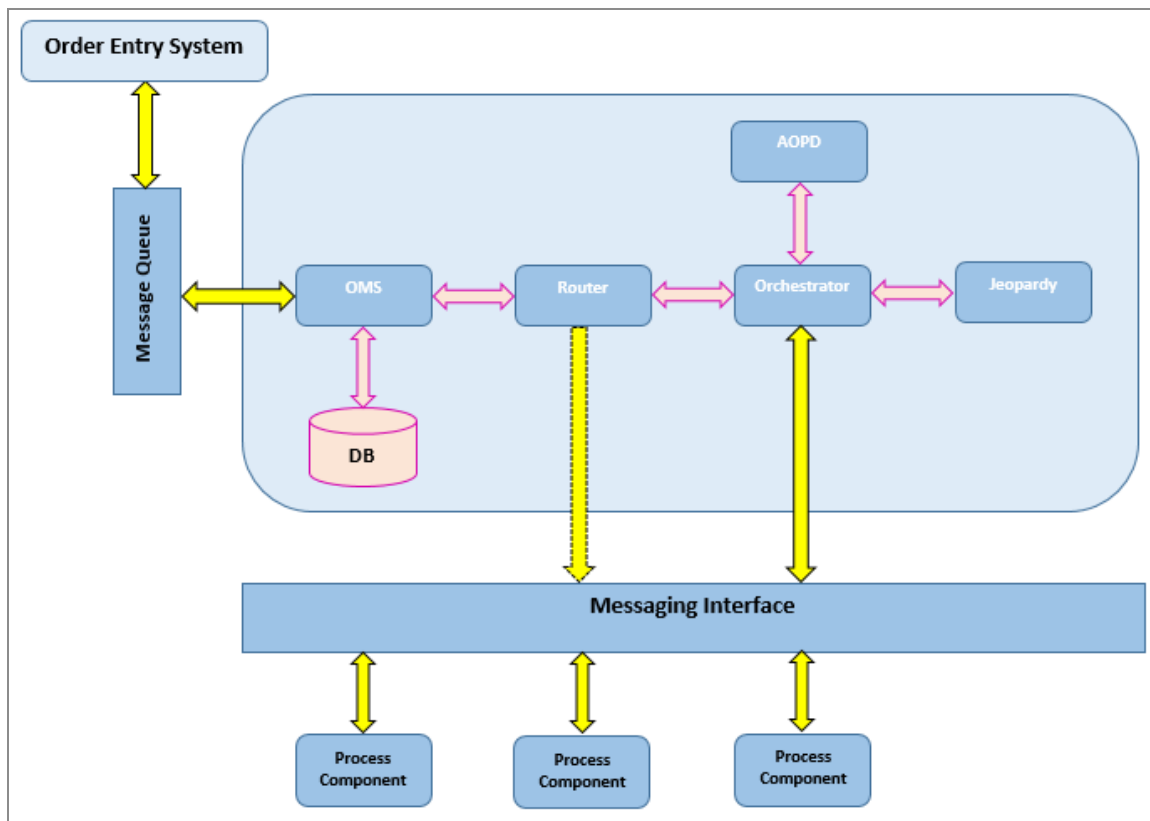
Service level agreements (SLAs) are commonly used to ensure the Quality of Service (QoS). The conventional way to manage service is to measure the Quality of Service and then determine whether the requirements have been met. This means that problems are detected and then corrective action is taken. By contrast, the Jeopardy Management module relies on predicting the result of Quality of Service compliance of process components that are part of the order fulfillment ecosystem. Therefore, it is frequently possible to take corrective action before a problem occurs, thus minimizing its impact.

Jeopardy is implemented as a tightly coupled component in TIBCO Order Management along with existing features such as Order Management Server, Automated Order Plan Development, and Orchestrator. Notifications to Jeopardy are sent as low-level API calls or through in-process communication, which is similar to other component communication. The advantages of the Jeopardy Management System are as follows:

- Jeopardy runs in cluster mode.
- Jeopardy notifications are processed in synchronous, or asynchronous (batch) modes.
- Improved performance of Jeopardy performance due to in-process (low-level API calls) communication between this application's components.
- Jeopardy is tightly coupled with Orchestrator and it follows all the finite state automata states.
- Orchestrator has complete control over the jeopardy functionality improving its performance.
- In case of any issues, the Orchestrator rolls back any updates to a plan or plan item, and jeopardy automatically reads the latest changes.
- Instead of saving plan, plan item, and process component information as part of Jeopardy, all the information is now saved as part of the state machine. Jeopardy reads and updates the information as part of the state machine itself.

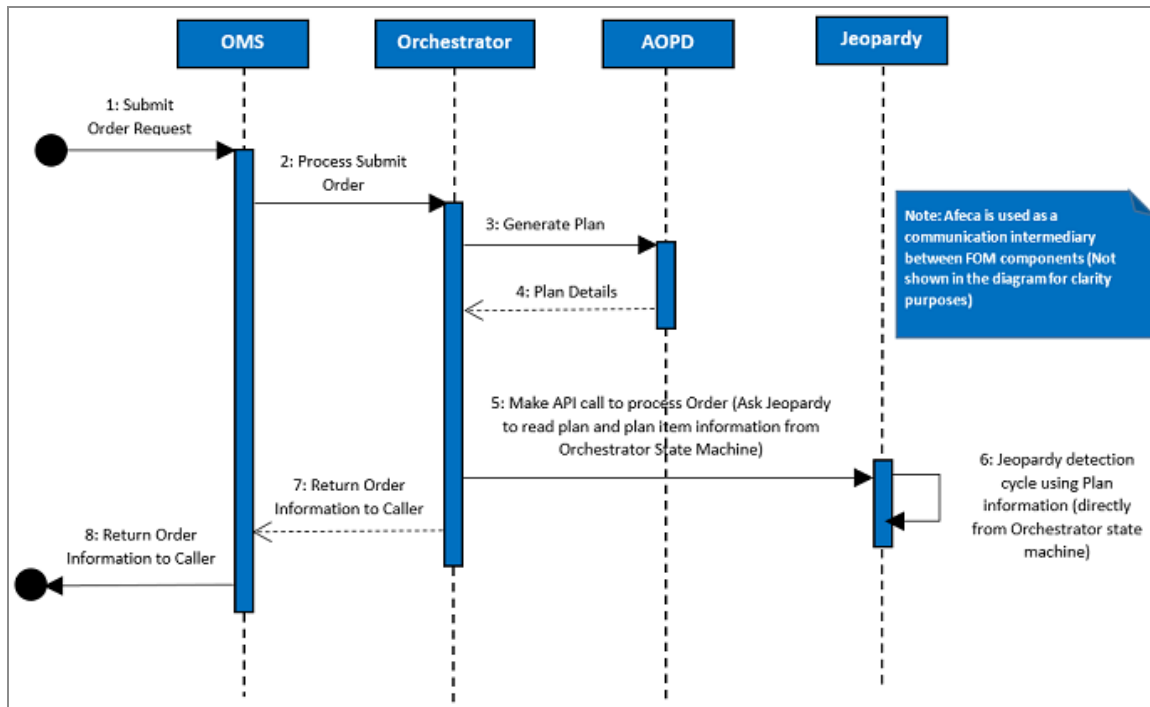
The following diagram is a representation of the architecture of Fulfillment Order Management:

Architecture of TIBCO Order Management



The following diagram is a representation of the Plan and Order Execution by using Jeopardy:

Plan and Order Execution



Jeopardy Management

The jeopardy management process involves three main activities:

1. Monitoring the Quality of Service
2. Reporting the Quality of Service
3. Predicting the Quality of Service

The objectives of Jeopardy Management are as follows:

1. Continuous collection of performance data and status information of all execution plan
2. Detect SLA violation
3. Predict Jeopardy Conditions for execution plan
4. Perform consequential actions
 - a. Send notification

b. Invoke web service

The Jeopardy Management System enables you to manage the risk associated with plan tasks falling behind schedule, and to prevent them from jeopardizing the timely fulfillment of an order. The Jeopardy Management System is a key component of Order Management. Jeopardy management is the process of monitoring the execution of a set of tasks in a plan to fulfill a customer order. In this application, execution plans are generated by decomposing orders based on the product model. Plans are orchestrated based on a schedule, and when a plan goes or is predicted to go outside the expected design of the schedule then the system notifies the stakeholders as early as possible to take the corrective steps.

A plan is composed of a series of plan items. Each plan item has at least two milestones:

- START milestone
- END milestone

Plan items might also have intermediate milestones that represent points of interest during the execution of that plan item. Service-level agreement of service provider that executes plan items are specified through the process component model or Plan fragment model, which stipulates among other things - the provided Services performance. SLAs have a typical duration and a maximum allowed duration to fulfill a plan item.

To manage the risk associated with plan tasks falling behind schedule, the jeopardy manager performs the following risk-management tasks:

1. Managing Critical Paths: Jeopardy management computes and keeps an account of critical paths through an execution plan. Two of these critical paths correspond to the typical and maximum durations of process components. The third type of critical path is based on the actual duration to date, once the execution plan has started processing. The critical paths are used to predict the completion date and time of the execution plan.
2. Monitoring jeopardy conditions at each of the following levels:
 - a. Plan Task
 - b. Execution Plan
3. Perform consequential actions at each of the following levels:
 - a. Plan Task
 - b. Execution Plan
 - c. Milestone

The Order Management UI shows a dashboard for jeopardy management containing the following elements along with Orders Summary, Amended Orders, Backlog Orders, Orders in Execution:

1. Orders in Jeopardy.
2. Jeopardy Live Alerts.
3. Jeopardy Recorded Alerts.

Jeopardy Events

The following are the types of jeopardy events:

Plan Item Jeopardy

The following table lists the jeopardy conditions for the plan item:

Plan Item Jeopardy Conditions	Description
AFF-JM-PLANITEM-0100	Plan item has exceeded the typical duration
AFF-JM-PLANITEM-0110	Plan item has exceeded the maximum duration
AFF-JM-PLANITEM-0120	Plan item has exceeded the required start
AFF-JM-PLANITEM-0200	Plan item start is predicted to exceed the required start and is increasing
AFF-JM-PLANITEM-0210	Plan item start is predicted to exceed the required start and is decreasing
AFF-JM-PLANITEM-0220	Plan item is no longer predicted to exceed the required start

Plan Jeopardy

The following table lists the jeopardy conditions for a plan:

Plan Jeopardy Conditions	Description
AFF-JM-PLAN-0100	Plan has exceeded the typical duration
AFF-JM-PLAN-0110	Plan has exceeded the maximum duration
AFF-JM-PLAN-0120	Plan has exceeded the out-of-scope threshold
AFF-JM-PLAN-0200	Plan is predicted to exceed the typical duration and is increasing
AFF-JM-PLAN-0210	Plan is predicted to exceed the typical duration and is decreasing
AFF-JM-PLAN-0220	Plan is no longer predicted to exceed the typical duration
AFF-JM-PLAN-0230	Plan is predicted to exceed the maximum duration and is increasing
AFF-JM-PLAN-0240	Plan is predicted to exceed the maximum duration and is decreasing
AFF-JM-PLAN-0250	Plan is no longer predicted to exceed the maximum duration

Order Selection for Jeopardy Management

Since Jeopardy Management System is designed to send an alert on the plan tasks that are falling behind schedule, and to prevent the plan tasks from jeopardizing the SLA for the entire order fulfillment, JeOMS makes some dynamic decisions when processing long and short-running orders.

There are chances of orders missing SLA requirements. If such a scenario occurs, then getting alerts on the short-running orders (orders expected to finish in 5 minutes or less) is not helpful as sending alerts to production support personnel, and the subsequent manual action on the alerts, take more time than the lifespan of the order.

Jeopardy, therefore, has a stronger focus on the long-running orders (orders that are expected to run for an hour or more). Jeopardy processes almost all the orders, but for short-running orders, jeopardy might skip some alerts that are not expected to be handled when the risk level for that alert changes to a severe one.

Understanding Plan

A plan is constituted of a series of plan items. Each plan item has at least two milestones - START and END. Plan items might also have intermediate milestones that represent points of interest during the execution of that plan item.

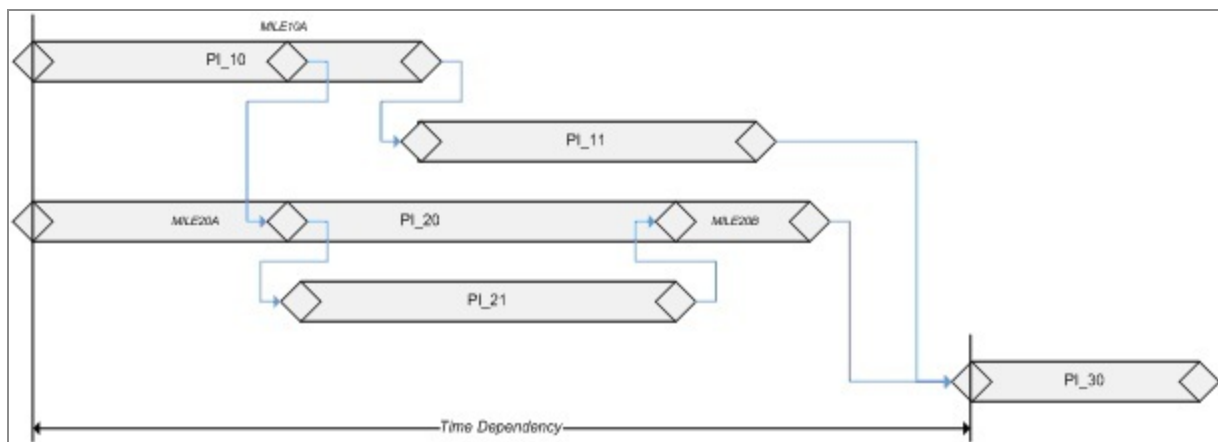
Note: If a plan item has intermediate milestones, all the intermediate milestones must be completed before the plan item is completed. In case some of the intermediate milestones are still pending and it gets a `PlanItemExecuteReply` request, then the request would be rejected.

There are two types of milestone dependencies:

Point Dependency	Time Dependency
dependency on the release of a given milestone in another plan item in the plan	dependency on a given date and time being exceeded

Note: Execution of a plan item stops at a milestone until that milestone has been released. A milestone with no dependencies is released immediately. However, a milestone with attached dependencies is only released once all the point and time dependencies are satisfied.

Plan Dependency



For details on dependencies, see [Understanding Dependencies](#).

Understanding Critical Path

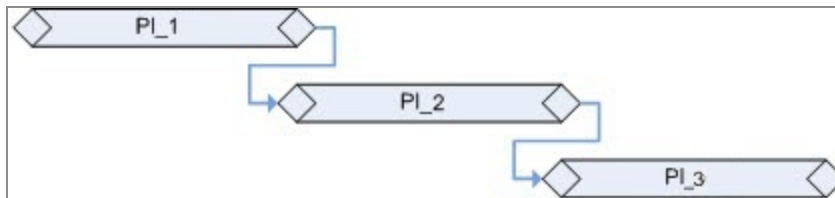
. The critical path is the longest sequence of plan item sections that determine the end time of a plan. The critical paths are used to project the completion date and time of the execution plan.

Paths are computed using the dependencies between plan item sections.

Critical Path Calculation

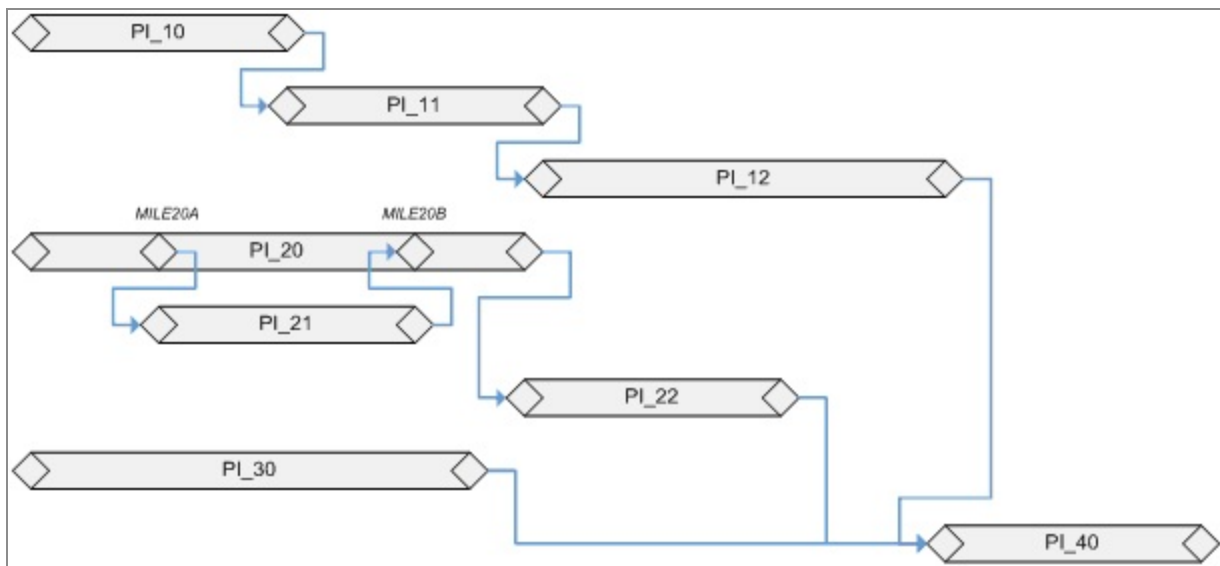
Critical path is used for both SLA and predictive jeopardy for monitoring a plan. A simple plan consists of a single execution path. For example:

Plan with Single Execution Path



Some plans have multiple execution paths as shown in the following figure:

Plan with Multiple Execution Paths



Understanding Dependencies

Dependency can be defined as a relationship between milestones in an execution plan. For example, Milestone B cannot start until Milestone A completes.

Milestone Dependencies

When a dependency on a milestone is determined, you can either depend on the milestone being started, or finished.

This means that Milestone 2 cannot be finished until Milestone 1 is started.

End Milestones

In case of an end milestone:

- another milestone cannot depend on the finish of an end milestone because there is no Finish (Release) on a Task Complete message
- the end milestone cannot be dependent on any other milestones

The following table lists the types of dependencies that can be set up between plan task milestones.

Dependency	Effect
Must Start On	The milestone must start on the date/time specified. If it cannot be started for some reason (for example, because a previous plan task is late), a jeopardy condition is triggered.
Finish to Finish	One milestone can be released when the other milestone is released
Start to Finish One	One milestone can be released only when the other begins

Jeopardy Management for Execution Plans

If a given plan task or milestone is in jeopardy, it might or might not indicate that the overall execution plan is in jeopardy. The jeopardy manager also provides facilities for monitoring whether the whole execution plan is running on time or taking longer to complete than expected. This is done by monitoring the forecast end date and time of the plan and comparing it against several threshold dates.

If you use start date scheduling or end date scheduling for your execution plans, you can set a different set of jeopardy conditions at the plan level.

Jeopardy Management for Plan Task

A simple process component that has only start and end milestones consists only of one section, but more complex components are made up of several sections. A section is the interval between two milestones.

At the level of process component sections, you can configure jeopardy conditions that enable you to detect both if the task has taken longer to complete than it must have, and also to detect if a task that is underway or has not yet started is predicted to take longer to complete than scheduled.

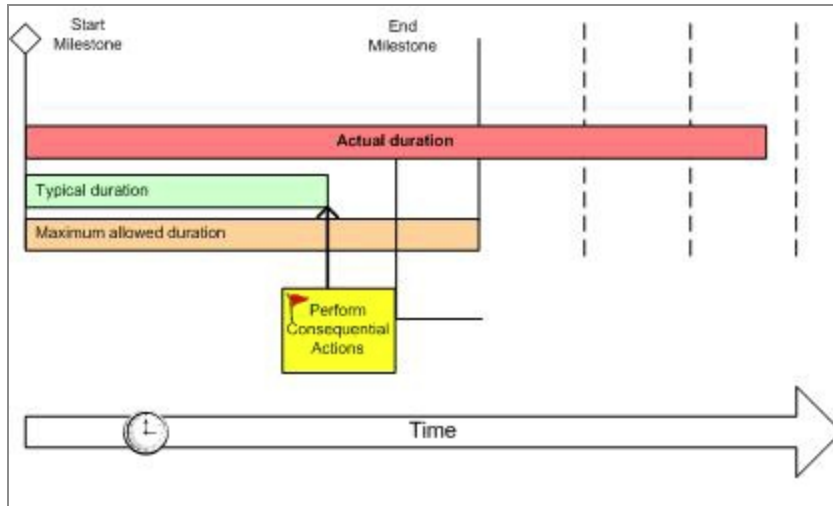
Jeopardy manager allows you to monitor the following durations:

- **Typical Duration:** you can specify this value when you create a process component, or when you define the plan task that uses the component in an execution plan.
- **Maximum Allowed Duration:** the maximum amount of time the activity represented by the task can take before it is considered to have overrun. You can specify this value when you create a process component, or when you define the plan task that uses the component in an execution plan.

There are critical paths identified in execution plans, constructed using the typical and maximum duration of the plan tasks included in those plans. If one of the process component sections being monitored has not completed before its defined typical duration, a monitor event is triggered (consequential action is performed). If the section has not been completed before the end of its maximum allowed duration, another monitor event is triggered. For Example, a plan has taken longer than expected/predicted. The plan task has exceeded its typical duration, its maximum duration, and two subsequent monitoring intervals. A monitor event has been fired at each stage to notify you of the following:

- After Typical Duration
- After Maximum Duration

Typical and Maximum Durations



Jeopardy Risk Region for Plan

NORMAL	• Plan running within typical duration
Hazard	• Plan running exceeding typical duration
Critical	• Plan running exceeding maximum duration
Out of Scope	• Plan running beyond last acceptable completion date

Must Start On Dependencies

The must start on dependency indicates that an activity must start at a specific point in time. You can apply these dependencies to milestones that denote the start of such activities; in normal circumstances, when the execution plan is running on schedule, these are used to schedule activities at the right time, by releasing the relevant milestones. However, if it is predicted that it is not possible to release a milestone at the scheduled

time, or if that time is reached and the milestone still cannot be released, the Jeopardy Management System recognizes the jeopardy condition.

Predictions are calculated during the jeopardy detection cycle. The frequency of Jeopardy detection cycle is configurable.

Consequential Actions

Jeopardy Manager raises the jeopardy event. If a plan item is in jeopardy, the `PlanItemJeopardy` event is raised. If a Plan is in jeopardy, the `PlanJeopardy` event is raised.

To reduce the number of Jeopardy notifications sent out for a particular plan, jeopardy sends either a predictive notification or the actual notification at the plan level. For example, if the jeopardy sends out a predictive notification `AFF-JM-PLAN-0200` for the Plan to possibly exceed the typical duration, then the jeopardy does not send out the `AFF-JM-PLAN-0100` notification if the plan actually exceeds typical duration, as they are the notifications for the same risk region.

Jeopardy event message contains payload with information about the jeopardy condition. You can configure the consequential that you want to perform when these events are raised by the system that configures the Jeopardy Rules.

Jeopardy Rules can be configured through Order Management Server UI rule configuration option.

For each of the listed jeopardy conditions, you can take any of the following possible consequential actions:

- Alert notification.
- Fulfillment Action.

Predictive Jeopardy

Predictive jeopardy is measured on several metrics:

Plan Item Start Date	Plan Duration
<p>For plan item milestones with both point and time dependencies, it is possible that the specified time dependency is not feasible based on the durations of the previously executed plan items that form the point dependencies on the same milestone</p>	<p>The overall duration of the plan can be calculated by performing a critical path analysis on the plan items that compose the plan</p>
<p>Plan item start date predictive jeopardy determines whether the plan item is later than the specified start date due to the other dependencies</p>	<p>Plan duration predictive jeopardy determines whether the execution duration of the plan exceeds the design duration of the plan</p>

Jeopardy Services

The Jeopardy services ensure that planned tasks stay on track, there are minimal risks, and the orders are completed on time.

Calculating all possible paths from the plan and identifying the critical path is a time-consuming task. Jeopardy must perform this task every time a state change notification is sent from the orchestrator. Since these lengthy calculations are resource-intensive, users can consider that Jeopardy effectively monitors long-running orders. However, Jeopardy is not designed to monitor orders that complete very quickly.

Let us refer to orders that complete very quickly as short orders. You can update the definition of short orders by modifying the "propName":

"shortLivedThresholdInMinutes" in the configuration of the Jeopardy service. By default, it is set to one minute.

Let us understand how the Jeopardy service calculates short-lived orders.

Suppose you have defined the following PlanFragments in the PlanFragment catalog, and the plan is generated such that there is only one plan item containing the following planFragment:

```
<pf:PlanFragmentModels
xmlns:pf="http://www.tibco.com/AFF/V4.0.0/classes/planFragment">
  <pf:planFragment>
    <pf:planFragmentID>PF01</pf:planFragmentID>
    <pf:planFragmentName>PF01</pf:planFragmentName>
    <pf:planFragmentVersion>V01</pf:planFragmentVersion>
    <pf:owner/>
    <pf:record_Type>Process</pf:record_Type>
    <pf:errorHandler/>
    <pf:retry>
      <pf:retryOverride>true</pf:retryOverride>
      <pf:retryFailed>true</pf:retryFailed>
      <pf:retryCount>3</pf:retryCount>
      <pf:retryDelay>60000</pf:retryDelay>
    </pf:retry>
    <pf:section>
      <pf:startMilestoneID>START</pf:startMilestoneID>
      <pf:endMilestoneID>END</pf:endMilestoneID>
      <pf:typicalDuration>1000</pf:typicalDuration>
    </pf:section>
  </pf:planFragment>
</pf:PlanFragmentModels>
```



```

    <pf:maximumDuration>6000</pf:maximumDuration>
  </pf:section>
</pf:planFragment>
</pf:PlanFragmentModels>

```

Here, the configured `maximumDuration` is 6000 milliseconds, which is less than one minute (default configuration for short-lived orders). The Jeopardy service does not monitor this plan as this order is short-lived.

For short-lived orders, on the **planTimeLine** tab, the following message is displayed:

Timeline cannot be computed for a short-lived plan.



Note: If you do not want to deploy the Jeopardy service, delete all the bridges and channels by using the **DeleteJeopardyEMSChannel.txt** script.

Following aspects play important roles in the Jeopardy service:

- [Durations](#)
- [Sections](#)
- [Plan Path](#)
- [Critical Path](#)

Durations

Jeopardy service handles the following types of durations:

- **Typical Duration:** The expected or standard amount of time required for the completion of the activity represented by the task under normal or average conditions, without any unforeseen delays or exceptional circumstances.
- **Hazard Duration:** The Hazard duration represents the projected upper limit of time for the completion of an activity, accounting for the specified risk threshold. It is derived by adjusting the Typical duration with the risk threshold percentage, allowing for a margin that considers potential hazards or risks that might extend the task duration beyond the standard expectation.
- **Maximum Duration:** The standard upper limit of time allocated for the completion

of the activity is represented by the task under optimal conditions. This duration represents the maximum permissible time frame within which the task is expected to be completed without unforeseen delays or exceptional circumstances.

- **OutOfScope Duration:** The OutOfScope duration signifies the expected upper boundary of time allocated for task completion, incorporating the out-of-scope threshold. This is calculated by modifying the maximum duration based on the out-of-scope threshold percentage. This duration accommodates potential extensions arising from conditions or requirements beyond the originally defined scope of the task.

Sections

Sections or milestone to milestone relationships, represent combinations of milestone pairs that might appear in the execution plan generated by AOPD (Automated Order Processing and Design).

For example, if START, M1, M2, and END are four milestones of a plan fragment, the following milestone combinations can be possible:

- START, M1, END
- START, M2, END
- START, M1, M2, END

To support any of these milestone combinations, the plan fragment contains the following section information:

- START to M1
- START to M2
- M1 to M2
- M1 to END
- M2 to END
- START to END

Plan Path

A Plan Path refers to a sequence of plan item sections in an execution plan. The plan paths are computed based on the dependencies between plan item sections by following a specific sequence.

- **Starting Point:** Initiate the path at a START milestone without any point dependencies.
- **Path Traversal:** Navigate through the plan item section for which this is the starting milestone. During traversal, determine if the terminating milestone has any dependencies. If a terminating milestone has multiple trailing dependencies, the path is divided and created a path of execution.
- **Terminating Point:** At the terminating milestone, no plan item section has a dependency on the trailing milestone of the current plan item.

Critical Path

The critical path in the plan is the longest path. It is determined by analyzing all available paths and selecting the one with the longest design duration. This path plays a crucial role in both SLA (Service Level Agreement) and predicting jeopardy, serving as a key aspect for monitoring a plan.

Design and Implementation

This section describes the design of the Jeopardy service and its implementation.

- [Plan Fragment Migration](#)
- [Start and End Time Computation of Section](#)
- [Communication Flow for Plan Monitoring](#)

Plan Fragment Migration

Every plan fragment has details about the possible sections for its associated plan item, including Typical and Maximum duration for each section. Jeopardy relies on this information for its functioning. When a plan is created, Jeopardy uses these details to define all potential plan paths based on the provided sections in the plan fragment. By considering the Typical and Maximum duration for each section, Jeopardy figures out how much time each plan path might take. This helps Jeopardy identify the critical path.

To ensure smooth performance, Jeopardy needs to move plan fragments from the catalog service to its own database before handling order status changes. Jeopardy extracts the necessary details from each plan fragment and stores them in its database for quicker and more efficient processing.

Configurations

Property	Purpose
catalogServiceBaseUrl	The base URL of the catalog service to fetch the plan fragments.
catalogRetryCount	Retry count in case a request failed due to server-side exceptions from the catalog service.
catalogRetryInterval	Interval in seconds between each retry.
catalogServiceTrustStorePassword	In case the catalog service is exposed on HTTPS, this property holds the trust store password to establish the SSL handshake.
catalogServiceTrustStoreType	In case the catalog service is exposed on HTTPS, this property holds the trust store type to establish the SSL handshake.
catalogServiceTrustStoreFileName	In case the catalog service is exposed on HTTPS, this property holds the trust store file's name, available in classpath, to establish the SSL handshake.
enableSecureAPI	This is the security enabled on the catalog service.
apiKey	This is the key to determine the HashKey for inter-service communication in case enableSecureApi is true.
riskThreshold	Percentage of Typical Duration used to calculate the Hazard Duration.

Property	Purpose
outOfScopeThreshold	Percentage of Maximum Duration used to calculate the out-of-scope Durations.

Process

The following information is extracted or calculated from the plan fragment:

- PlanFragmentID
- PlanFragmentName
- PlanFragmentVersion
- Sections (list of all the sections mentioned in the plan fragment)
- PerfValues (this is the map that contains the duration of all the sections available in the plan fragment)
 - **Typical Duration:** Extracted directly from the plan fragment.
 - **Hazard Duration:** Calculated using the riskThreshold and Typical Duration.
 - **Maximum Duration:** Extracted directly from the plan fragment.
 - **OutOfScope Duration:** Calculated using the outOfScopeThreshold and Maximum Duration.
- InferredPerfValues
 - There could be a possibility that not all possible traversable section information is provided in the Plan Fragment. In this case, Jeopardy can infer the PerfValues of those traversable sections.
 - Jeopardy prepares an undirected graph of milestones, acting as a map that shows connections between milestones.
 - Each milestone is a point on the map (Vertex), and the distance between them indicates which milestones are connected (Edge).
 - Using the Breadth-First Search (BFS) path-finding algorithm, Jeopardy identifies paths and calculates distances between milestones based on performance values.

Approaches

Plan fragment migrations are performed using the following methods:

- **Rest API:** Jeopardy introduces a new REST API for plan fragment migration. When triggered, this API initializes the migration process by making a REST call to the `v1/planfragmentmodel/all` endpoint of the catalog service. It fetches 20 plan fragments in a single call. For handling multiple calls efficiently, it employs Java's `CompletableFuture` and maximizes parallelism.

```
/v1/plan-fragment/migrate
```

- **Plan Fragment Refresh:** Whenever a plan fragment is added to the catalog service through REST or JMS, the catalog service dispatches a notification to the `tibco.fos.global.cache.clean.publish` topic. Jeopardy subscribes to this topic, and on receiving a notification, it initiates the migration process for that specific plan fragment. This is done by making a REST call to the `/v1/planfragmentmodel/bulk` endpoint of the catalog service.

Start and End Time Computation of Section

Jeopardy calculates two duration maps for each plan item section, each serving a distinct purpose:

- **EarlyStartMap:** This map captures the earliest possible start time of a section among all plan paths. It represents the initiation time of a section, considering dependencies and the critical path.

For example,

If the section is in **Execution** state,

`earlyStartTime` = `actualStartTime` of Section

Otherwise,

If the node is the Virtual Start Node, `earlyStartTime` is set to the plan start time.

For other nodes, the early start time is determined by selecting the maximum value between the parent's start time (if the dependency is based on the start milestone) and the parent's end time (if the dependency is based on the end milestone) in the parent-child relationship of the nodes.

- **EarlyFinishMap:** This map denotes the earliest finish time of a section among all the plan paths. It signifies the earliest point at which a section could be completed, considering dependencies and the critical path.

For example,

If the section is in the **Completed** state,

`EarlyFinishTime = ActualEndTime`

Otherwise,

`EarlyFinishTime` is calculated as `EarlyStartTime + Duration Value + Total Suspension Time` (of the section).

The values in these maps depend on all the dependent sections as a plan item section can belong to multiple plan paths.

For each duration type, the calculations are as follows:

Communication Flow for Plan Monitoring

- [Submit Order Execution Plan](#)
- [Status Change Notification Listener](#)
- [Plan Path Computation](#)
- [PlanPathRequestEventListener](#)

Submit Order Execution Plan

Jeopardy maintains records of plan item, milestone, and plan completion timestamps. To effectively process status change notifications dispatched by the Orchestrator, Jeopardy requires prior knowledge of the plan details. AOPD ensures this by submitting the plan to Jeopardy via a REST API before sending it to Jeopardy. This ensures that Jeopardy processes the plan before receiving status change notifications from the orchestrator.

Configurations

Property	Purpose
<code>riskThreshold</code>	Percentage of Typical Duration used to calculate the Hazard Duration.
<code>outOfScopeThreshold</code>	Percentage of Maximum Duration used to calculate the out-of-scope Durations

Process

On receiving the plan, Jeopardy populates the Plan, Plan_Instance, and Milestone tables, including Virtual Start and Virtual End plan items.

Database tables

It populates the following tables:

- Plan_instance
- Plan_item_instance
- Milestone

Status Change Notification Listener

To monitor the completion status of plans, plan items, and milestones, Jeopardy subscribes to the outbound status change notifications dispatched by the Orchestrator. It selectively processes the following types of notifications:

- [Order Status Change Notification](#)
- [Plan Status Change Notification](#)
- [Plan Item Status Change Notification](#)
- [Milestone Status Change Notification](#)

Order Status Change Notification

The purpose of monitoring the order status change notifications is to enable Jeopardy to respond to specific statuses, particularly the **Withdrawn** status. When the Orchestrator dispatches an order status change notification indicating that an order has been withdrawn, Jeopardy listens to this notification and takes appropriate actions to reflect the updated status.

Behavior

When Jeopardy receives an order status change notification with the newStatus set to "Withdrawn":

- Jeopardy updates the status of the associated plan to "Withdrawn" without deleting the plan instance from the database. This approach ensures graceful handling of any other pending notifications.
- Additionally, Jeopardy deletes the corresponding entry from the Time Window table

if it exists. This action ensures that JeopardyDetectionCycle stops monitoring the plan for this order.

Plan Status Change Notification

This section describes about the notifications whenever there is a change in the status of plans.

-
- [Transition to Suspend](#)
- [Transition from Suspend to Execution](#)
- [Transition to Complete or Canceled](#)

Transition from Pending to Execution

When a plan transitions from "Pending" to "Execution", Plan execution is started. Jeopardy systematically processes the transition from "Pending" to "Execution", updates relevant tables, and prepares the necessary data structures to start plan monitoring and management. The detailed process includes:

1. Move Plan to Execution

Jeopardy updates specific columns in the plan_instance table to reflect the transition:

- planStartTime: Set to eventTimeInMillis.
- actualStartTime: Set to eventTimeInMillis.
- lastStatusChangeTime: Updated to eventTimeInMillis.
- status: Changed to "Execution".
- startNotificationReceived: Marked as "true".
- currentRiskRegion: Set to "NORMAL".

2. Complete Virtual Start Plan Item

The Virtual Start Plan Item (identified by id = "__START_PLAN_ITEM") is marked as completed by updating the relevant columns in the Plan_Item_Instance table:

- status: Set to "COMPLETE".
- actualEndTime: Updated to eventTimeInMillis.

3. Complete All Milestones in Virtual Plan Item

All milestones within the virtual plan item are marked as completed by updating the `status` and `actualRelease` columns in the Milestone table:

- `status`: Set to "COMPLETE".
- `actualRelease`: Updated to `eventTimeInMillis`.

4. Dispatch Initial Plan Path Request

Request to prepare the Initial Plan Path is dispatched to the `planPathRequestNotificationDeliveryQueue` queue.

For more information, see the [PlanPathRequestEventListener](#) section.

Transition to Suspend

When a plan undergoes suspension, Jeopardy ensures it is cognizant of this change, allowing for the incorporation of plan suspension time into the monitoring process. The detailed process involves the following steps:

1. Move Plan to Suspension

Jeopardy updates the `plan_instance` table to represent accurately the latest change in the plan

- `status`: Set to "Suspended".

2. Suspend the Started Adjacency

All sections currently in "Execution" status are marked as "Suspended" to acknowledge the plan's suspension

- `sectionStatus`: Set to "Suspended".
- `previousSectionStatus`: Set to "Execution".
- `lastStatusChangeTime`: Updated to `eventTimeInMillis`.

3. Removing Entries from the Time Window table

As the plan enters a suspended state, Jeopardy ceases monitoring by eliminating all corresponding entries for the plan from the `time_window` table.

Transition from Suspend to Execution

When a plan transitions back to the **Execution** state from **Suspension**, Jeopardy considers the duration the plan spent in suspension. During the Jeopardy Detection Cycle, if the plan experienced a period of suspension, Jeopardy adds this duration to the `predictedEndTime` to assess if the plan is at risk. The detailed process is outlined as follows:

1. Move plan to **Execution**

Jeopardy updates the plan_instance table to accurately reflect the latest change.

status = Execution

2. Restart all suspended adjacency

- a. For all sections that are suspended, Jeopardy calculates the duration the section spent in the suspended state.

suspensionTime = eventTimeInMillis - lastStatusChangedTime

- b. It updates the earlyFinishMap with the additional suspensionTime.

- c. Jeopardy then updates the plan_adjacency table with the following:

- sectionStatus = Start
- previousSectionStatus = SUSPENDED
- suspensionTime = Computed suspensionTime
- lastStatusChangeTime = eventTimeInMillis

3. Dispatch Rebuild Plan Path Request

The request to rebuild the plan path is dispatched to the planPathRequestNotificationDeliveryQueue.

Refer to [PlanPathRequestEventListener](#) section for more information.

Transition to Complete or Canceled

When a plan reaches a final state, Jeopardy takes specific actions to account for this transition and appropriately updates its records. The process involves the following steps:

1. Move plan to **Complete** or **Canceled**

Jeopardy updates the plan_instance table with the following:

- status = Complete or Canceled
- actualEndTime = eventTimeInMillis
- lastStatusChangeTime = eventTimeInMillis

2. Complete Virtual End Plan Item

Virtual End Plan item (id = "__END_PLAN_ITEM") is marked as completed, signifying that the plan has reached its final state.

- `status = Complete`
- `actualEndTime = eventTimeInMillis`

3. Complete all Milestones in Virtual End Plan Item

All milestones of the Virtual End Plan Item are marked as completed in the milestone table:

- `status = Complete`
- `actualRelease = eventTimeInMillis`

4. Purging All data for Short Lived

Short-lived orders are not monitored by Jeopardy. Thus, once the plan reaches its final state for a short-lived order, all corresponding data are removed from the following tables:

- `plan_instance`
- `plan_item_instance`
- `milestones`

5. Compute Plan Expected Finish Times and Determine Risk Region

- Plan Expected Finish Times are computed based on the last real nodes of the Critical Paths.
- Plan Expected Typical Finish Time (`planExpectedTypicalFinishTime`) is calculated using the `TypicalEarlyFinishTime` state of the last real node of the Typical Critical Path.
- Plan Expected Maximum Finish Time (`planExpectedMaximumFinishTime`) is calculated using the `MaximumEarlyFinishTime` state of the last real node of the Maximum Critical Path.
- Plan Expected Out of Scope Finish Time (`planExpectedOosFinishTime`) is calculated using the `OosEarlyFinishTime` state of the last real node of the Maximum Critical Path.
- The Risk Region is determined based on the `actualEndTime` in comparison to the expected finish times:
 - If `actualEndTime < planExpectedTypicalFinishTime`, `riskRegion = Normal`

- If `planExpectedTypicalFinishTime < actualEndTime < planExpectedMaximumFinishTime`, `riskRegion = Hazard`
 - If `planExpectedMaximumFinishTime < actualEndTime < planExpectedOosFinishTime`, `riskRegion = Critical`
 - If `actualEndTime > planExpectedOosFinishTime`, `riskRegion = Out of Scope`
- f. The computed `riskRegion` is updated in `plan_instance`:
- `currentRiskRegion = riskRegion`
6. Complete all Sections of Virtual End Plan Item
- All sections for the Virtual End Plan Item are marked as completed in the `plan_adjacency` table.
7. Removing Entries from the Time Window table
- As the plan enters a final state, Jeopardy stops monitoring by removing all corresponding entries for the plan from the `time_window` table.

Plan Item Status Change Notification

Whenever a plan item undergoes state changes, the orchestrator dispatches plan item status change notifications. These notifications fall into two categories based on the Action header in the JMS message: REQUEST and RESPONSE.

The Action REQUEST indicates that a `PlanItemExecuteRequest` was dispatched during this transition, typically occurring when a plan item shifts from Pending to Execution. Notifications with Action RESPONSE signify that this transition occurred based on the response from the Southbound System.

Processing Plan Item Status Change Notification with Action REQUEST

1. Complete the Start milestone
 - a. As the Plan Item Execute response is dispatched during this transition, indicating the completion of the start milestone for this plan item.
 - b. Jeopardy updates the Start milestone of this plan item by modifying the milestone table with the following information:
 - `status = COMPLETE`
 - `actualRelease = eventTimeInMillis`
2. Mark the plan item as under processing

isUnderProcessing is updated to true in the plan_item_instance table for this plan item.

3. Move the plan item to Execution

The plan_item_instance table is updated with the following information:

- status = "EXECUTION"
- riskRegion = "NORMAL"
- actualStartTime = eventTimeInMillis
- typicalEndTimestamp = eventTimeInMillis + planItemTypicalDuration
(Typical Duration of Start to End section available in PC)
- maximumEndTimestamp = eventTimeInMillis + planItemMaximumDuration
(Maximum Duration of Start to End section available in PC)

4. Start all sections with the start milestone as Plan Item Start Milestone

The plan_adjacency table, where start_milestone = "START", is updated with the following information:

- sectionStatus = "Start"
- actualStartTime = eventTimeInMillis

5. Unmark the plan item from under processing

isUnderProcessing is updated to false in the plan_item_instance table for this plan item.

Processing Plan Item Status Change Notification with Action RESPONSE

For non-executing plan items, the transition occurs directly from PENDING to COMPLETE. In this case, the orchestrator dispatches the plan item status change notification with Action as RESPONSE. Therefore, handling such plan items involves some steps similar to those done when the action is REQUEST.

1. Steps specific For Non-Executing Plan Item

a. Update Actual Start Time

- Non-executing plan items transition directly from PENDING to COMPLETE. Hence, startTime and endTime are the same for such plan items.
- Jeopardy updates the plan_item table with the following information:

- `actual_start_time = eventTimeInMillis`

b. Complete the Start Milestone

Jeopardy updates the milestone table, where the milestone id is START, with the following information:

- `actualRelease = eventTimeInMillis`
- `status = Complete`

c. Start all sections with the start milestone as Plan Item Start Milestone

The `plan_adjacency` table, where `start_milestone = "START"`, is updated with the following information:

- `sectionStatus = "Start"`
- `actualStartTime = eventTimeInMillis`

2. Mark the plan item as under processing

`isUnderProcessing` is updated to true in the `plan_item_instance` table for this plan item.

3. Complete the plan item

Jeopardy updates the `plan_item_instance` table with the following information:

- `status = COMPLETE`
- `actualEndTime = eventTimeInMillis`

4. Complete the END Milestone

Jeopardy updates the milestone table, where `milestoneid = 'END'`, with the following information:

- `status = Complete`
- `actualRelease = eventTimeInMillis`

5. Update all sections for this plan item where `endMilestone = 'END'`

a. Compute the risk region

i. Compute the time taken for this section to complete:

`timeTaken = eventTimeInMillis - sectionStartTime - sectionSuspensionTime`

- ii. If `timeTaken > section's Maximum Duration`, `riskRegion = CRITICAL`
 - iii. If `timeTaken > section's Typical Duration`, `riskRegion = HAZARD`
 - iv. Else `riskRegion = NORMAL`
 - b. Update the section in the `plan_adjacency` table with the following information:
 - `actualEndTime = eventTimeInMillis`
 - `sectionStatus = COMPLETE`
 - `riskRegion = Computed Risk Region`
 - c. Remove the section from the Time window table as this section is completed and no longer requires monitoring.
6. Unmark the plan item from under processing
`isUnderProcessing` is updated to `false` in the `plan_item_instance` table for this plan item
 7. Dispatch rebuild plan path request
 Request to rebuild the plan path is dispatched to the `planPathRequestNotificationDeliveryQueue`.
 Refer to [PlanPathRequestEventListener](#) section for more information.

Milestone Status Change Notification

The orchestrator dispatches status notifications only for intermediate milestones. Jeopardy takes note of this status change and performs the following steps:

1. Complete the Milestone
 Jeopardy updates the milestone table with the following information:
 - `status = Complete`
 - `actualRelease = eventTimeInMillis`
2. Mark the Plan item as under processing
`isUnderProcessing` is updated to `true` in the `plan_item_instance` table for this plan item.
3. Process Sections where `startMilestoneId = given milestone`
 Jeopardy updates such sections in `plan_adjacency` with the following information:

- `sectionStatus = START`
 - `actualStartTime = eventTimeInMillis`
4. Process Sections where `endMilestoneId = given milestone`
 For every such section,
 - a. Compute the risk region
 - Compute the time taken for this section to complete
 - `timeTaken = eventTimeInMillis - sectionStartTime - sectionSuspensionTime`
 - If `timeTaken > section's Maximum Duration`, `riskRegion = CRITICAL`
 - If `timeTaken > section's Typical Duration`, `riskRegion = HAZARD`
 - Else `riskRegion = NORMAL`
 - b. Update the section in the `plan_adjacency` table with the following information:
 - `actualEndTime = eventTimeInMillis`
 - `sectionStatus = COMPLETE`
 - `riskRegion = Computed Risk Region`
 - c. Remove the section from the Time window table as this section is completed and no longer requires monitoring.
 5. Unmark the plan item from under processing
`isUnderProcessing` is updated to false in the `plan_item_instance` table for this plan item
 6. Dispatch Rebuild Plan Path Request
 The request to rebuild the plan path is dispatched to the `planPathRequestNotificationDeliveryQueue`.
 Refer to [PlanPathRequestEventListener](#) section for more information.

Plan Path Computation

Jeopardy employs a depth-first approach to generate all plan paths and updates `earlyStart` and `earlyFinish` for each section. In this methodology, the Virtual Start Node is treated as the root node.

Process

- For the given node, compute the EarlyStartMap and EarlyFinishMap of the Virtual Start Node.
- If the node is not a virtual node,
 - Populate the time_window table for MUST_START detection with a typical earlyStartTime as the expectedTime.
 - Populate the time_window table for TYPICAL_DURATION detection with typical earlyFinishTime and MAX_DURATION detection with max earlyFinishTime as expectedTime.
- If the section has a dependency,
 - Repeat the entire process
 - If a section has multiple dependencies, the path branches, creating a path of execution
- If the section does not have any dependency
 - Consider the path as it ended.
 - Add this path to the list of generated paths.

PlanPathRequestEventListener

This component serves as a dedicated listener for handling plan path requests throughout various stages of the plan's lifecycle. These requests might be initiated as either an initial plan path request or a rebuild plan path request.

Process

Initial Plan Path Request

- Populate Plan Adjacency

The plan adjacency, a representation of plan item sections within the plan, is computed and stored in the plan_adjacency table.
- Prepare and Populate Plan Paths
 - On obtaining section information, the system initiates the preparation of all possible plan paths.
 - Computed plan paths are subsequently saved in the plan_path table.
- Determine Critical Plan Path

The critical plan path, denoting the longest sequence through the plan, is computed and stored in the plan_critical_path table.

- Determine Plan Expected End Time

The predicted end time in the critical path is considered as the plan's expected end time.

- Determine if the Plan is ShortLived

- Jeopardy identifies short-lived plans, where the difference between predicted end time and plan start time is less than the specified threshold (shortLivedThresholdInMinutes).
- Short-Lived plans are excluded from monitoring.

- Populate the Time Window table for All Plan Sections

Sections not yet completed are stored in the time_window table, enabling Jeopardy to commence monitoring during the next Jeopardy Detection Cycle.

Amendment Plan Path Request

- Populate Plan Adjacency

- The plan adjacency, a representation of plan item sections within the plan, is computed and stored in the plan_adjacency table.
- In the case of an amendment, the plan might contain sections that were previously present and some newly introduced sections.
- Jeopardy would delete sections no longer present in the plan and add new sections while keeping existing ones intact.

- Prepare and Populate Plan Paths

- On obtaining section information, the system initiates the preparation of all possible plan paths.
- Computed plan paths are subsequently saved in the plan_path table.

- Determine Critical Plan Path

The critical plan path, denoting the longest sequence through the plan, is computed and stored in the plan_critical_path table.

- Determine Plan Expected End Time

The predicted end time in the critical path is considered as the plan's expected end

time.

- Determine if the Plan is ShortLived
 - Jeopardy identifies short-lived plans, where the difference between predicted end time and plan start time is less than the specified threshold (shortLivedThresholdInMinutes).
 - Short-Lived plans are excluded from monitoring.

- Populate the Time Window table for All Plan Sections

Sections not yet completed are stored in the time_window table, enabling Jeopardy to commence monitoring during the next Jeopardy Detection Cycle.

Rebuild Plan Path Request

- Stop Plan Monitoring during Plan Path Rebuild

Delete all entries from the time_window table corresponding to the given planId and tenantId.

- Check if any plan items are still being processed
 - Given the possibility of multiple plan items being processed simultaneously by Jeopardy, the system updates the isUnderProcessing flag to true before processing each plan item. This flag is then set to false once Jeopardy completes the processing of that specific plan item.
 - If any plan item for the plan is still being processed, skip the plan path rebuild request.
- Prepare and Populate Plan Paths
 - By now some of the sections are updated with their actualStartTime and actualEndTime.
 - Jeopardy uses this information and prepares plan paths again.
 - The system then saves the updated plan paths in the Plan_Path table.

- Determine Critical Plan Path

The critical plan path, denoting the longest sequence through the plan, is computed and stored in the plan_critical_path table.

- Determine Plan Expected End Time

The predicted end time in the critical path is considered as the plan's expected end

time.

- Populate the Time Window table for All Plan Sections

Sections not yet completed are stored in the `time_window` table, enabling Jeopardy to commence monitoring during the next Jeopardy Detection Cycle.

Pending Jeopardy Events

The orchestrator dispatches status change notifications for all orders. In multi-instance scenarios, there is a possibility that certain order status change notifications are picked by one instance while another instance is still processing plan development notifications. To handle this, Jeopardy introduces the concept of pending jeopardy events.

Plan Availability

Jeopardy processes incoming notifications only if the plan is available for processing. If the plan is not available for processing, the incoming event is saved in the `pending_jeopardy_events` table. These events are processed after a plan path is created for the respective plan.

Conditions of Plan Availability

- Is the plan available in the `plan_instance` table? If not, the plan is not available.
- If the plan is available, is the plan under amendment? If yes, the plan is not available.
- If the plan is available and not under amendment, do plan paths exist for the given plan? If not, the plan is not available.
- If the plan is available, not under amendment, and the plan path request is processed, then the plan is available.

Processing of Pending Jeopardy events

After a plan path request is processed for a newly created plan or for a plan that was amended, Jeopardy dispatches a notification to the `jeopardy.pending.events.notification` queue. This queue is used by Jeopardy to process all pending jeopardy events for a given plan asynchronously. After the jeopardy events are processed, an event is deleted from the `pending_jeopardy_events` table.

Jeopardy Detection Cycle

The primary function of Jeopardy is to continuously monitor the plan throughout its lifecycle and generate notifications if the plan deviates from its anticipated timeline.

Types of Detections

Jeopardy identifies various types of detections:

- [Must Start](#)
- [Duration](#)
- [Plan Item Timeline](#)
- [Plan Timeline](#)

Must Start

Purpose

- Indicates that a specific milestone or plan item must have started by the current time.
- If a section's typical earlyStart time is earlier than the current time, Jeopardy considers the section in Jeopardy.
- Dispatches a Must Start message for a plan item if the section's start milestone ID is 'START' or for a milestone must start message for intermediate milestones.

Notification Message

Message Type	Message
MUST_START (For Plan Item)	Plan item has exceeded the required start
MUST_START (For Milestone)	Milestone {ABC} must have been completed at {Predicted Time}

Duration

TYPICAL_DURATION

- When a section is not completed before it is typical earlyFinish time, Jeopardy considers the section in Jeopardy
- Dispatches a TYPICAL_DURATION message for plan item if the section's start milestone ID is 'START' or for a milestone must start message for intermediate milestones.

MAXIMUM_DURATION

- When a section is not completed before it is maximum earlyFinish time, Jeopardy considers the section in Jeopardy
- Dispatches a MAXIMUM_DURATION message for plan item if the section's start milestone ID is 'START' or for a milestone must start message for intermediate milestones.

Notification Message

Message Type	Message
TYPICAL_DURATION (For Plan Item)	Plan item has exceeded its typical duration.
TYPICAL_DURATION (For Milestone)	Milestone {ABC} has exceeded its typical duration.
MAX_DURATION (For Plan Item)	Plan item has exceeded maximum duration.
MAX_DURATION (For Milestone)	Milestone {ABC} has exceeded maximum duration.

Plan Item Timeline

Purpose

For each plan item, Jeopardy monitors the following time lines:

Timeline	Description
MUST_START	The plan item must have started before its

Timeline	Description
	predicted start time (as explained above).
TYPICAL_PLAN_ITEM_DURATION	The plan item must have been completed before its predicted typical end time.
HAZARD_PLAN_ITEM_DURATION	The plan item must have been completed before its predicted hazard end time.
MAXIMUM_PLAN_ITEM_DURATION	The plan item must have been completed before its predicted maximum end time.
OOS_PLAN_ITEM_DURATION	The plan item must have been completed before its predicted out-of-scope (OOS) end time.

Notification Message

Message Type	Message
HAZARD_PLAN_ITEM_DURATION	Plan Item has exceeded hazard duration and is in Hazard riskRegion
MAXIMUM_PLAN_ITEM_DURATION	Plan Item has exceeded Maximum duration and is in Critical riskRegion
OOS_PLAN_ITEM_DURATION	Plan Item has exceeded out-of-scope duration and is in out-of-scope riskRegion

Plan Timeline

Purpose

For each plan, Jeopardy monitors the following time lines:

Timeline	Description
TYPICAL_PLAN_DURATION	The plan must have been completed before its predicted typical end time (as explained above).
HAZARD_PLAN_DURATION	The plan must have been completed before its predicted hazard end time.
MAXIMUM_PLAN_DURATION	The plan must have been completed before its predicted maximum end time.
OOS_PLAN_DURATION	The plan must have been completed before its predicted out-of-scope (OOS) end time.

Notification Message

Message Type	Message
PLAN_TYPICAL_MESSAGE	Plan has exceeded typical duration and is expected to be in Hazard riskRegion in {time remaining to reach hazard}
PLAN_HAZARD_MESSAGE	The plan has exceeded hazard duration and is in Hazard riskRegion.
PLAN_MAX_MESSAGE	Plan has exceeded Maximum duration and is in Critical riskRegion
PLAN_OOS_MESSAGE	Plan has exceeded OOS duration and is in OOS riskRegion

Population Of Time_Window table

Whenever a plan path is computed or recomputed, the following time lines are predicted:

- EarlyStart and EarlyFinish for every section
- ExpectedEndTime for every duration type for every plan item

- ExpectedEndTime for every duration type for the plan.

After these time lines are predicted, Jeopardy inserts an entry for every detection type in the TIME_WINDOW table with the following data:

- OrderId
- PlanId
- PlanItemId -> Populated only if the detection is for a plan item or milestone
- StartMilestoneId -> Populated only for MUST_START and DURATION types
- EndMilestoneId -> Populated only for MUST_START and DURATION types
- TenantID
- detection_type
- expected_time -> Refers to the corresponding time for the detection type.

Jeopardy Detection Cycle (JDC)

The Jeopardy detection cycle is a cron job configurable through the jeopardyDetectionCronExpression property.

Properties

Property Name	Property Description
jeopardyDetectionCronExpression	JDC Cron Schedule
databaseType	Database type. Either PostgreSQL or Oracle
numOfOrdersPerCycle	Number of orders to process per cycle

Process

- Check for Jeopardy in Time_Window
 - JDC goes to the database and checks for any jeopardy using the following

query, where TIME_OFFSET is the current time and NUMOFROWS refers to the number of orders to be detected per cycle and is configurable via the numOrdersPerCycle:

- PostgreSQL: `SELECT * FROM time_window WHERE orderid IN (SELECT DISTINCT orderid FROM time_window WHERE expected_time <= 'TIME_OFFSET' AND status = 0 LIMIT 'NUMOFROWS')`
- Oracle: `SELECT * FROM time_window WHERE orderid IN (SELECT DISTINCT orderid FROM time_window WHERE expected_time <= 'TIME_OFFSET' AND status = 0 FETCH FIRST 'NUMOFROWS' ROWS ONLY)`
- If any row is returned from this query, Jeopardy considers that a jeopardy has been detected because the expected_time is before the current time.
- For every row returned, jeopardy would update the status column to 1, thus marking them as being in process.
- Depending on the detection type, Jeopardy then dispatches the notification from the database.
- After a message is dispatched, it would go on to delete the row from the TIME_WINDOW table.

Reset Stuck Jeopardy Cron Job

This exceptional handling or fail-safe job is designed to address situations where a particular Jeopardy Detection Cycle run fails and fails to update the status column back to 0.

Properties

Property Name	Property Description
jeopardyEventResetCronExpression	Reset Stuck Jeopardy Cron Schedule
jeopardyResetOffsetInSeconds	Offset that is used to find the detection tasks that are stuck.

Process

- This job runs according to the schedule defined in

jeopardyEventResetCronExpression.

- It updates any detection task that has been stuck for the last jeopardyResetOffsetInSeconds seconds and sets the status to 0 by running the following query:

```
UPDATE time_window SET status = 0 WHERE lastchangetimestamp <= ?1 AND
status = 1
```

Calculation to find stuck detection task

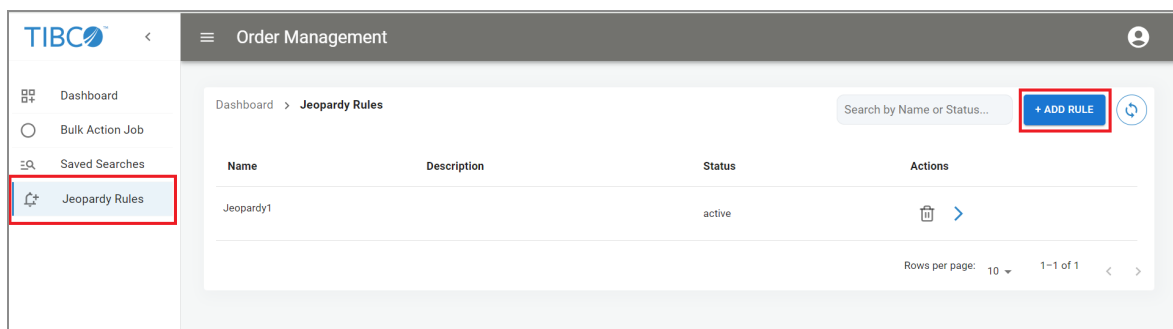
`lastchangetimestamp <= Current_time - jeopardyResetOffsetInSeconds.`

Adding Jeopardy Rules On OMS UI

Jeopardy services ensure that planned tasks stay on track, minimizing risks, and ensuring timely order fulfillment. You can add jeopardy rules on the Order Management System UI (OMS UI).

Procedure

1. From the side bar on the OMS UI, go to **Jeopardy Rules** and click the **ADD RULE** button.



2. In the **General Info** window, enter the details and click the **SAVE & CONTINUE** button.

The screenshot shows the 'Order Management' application interface. The breadcrumb trail is 'Jeopardy Rules > Add Rule'. A 'SAVE' button is in the top right. On the left, a sidebar shows 'General Info' (selected), 'Conditions', and 'Actions'. The main area is titled 'General Info' and contains four input fields: 'Name', 'Description', 'Status' (a dropdown menu), and 'Event Type' (a dropdown menu). At the bottom center, a 'SAVE & CONTINUE' button with a right-pointing arrow is highlighted with a red rectangle.

3. On the **Conditions** tab, expand **Plan**, select your desired option, and then click the **NEXT** button.

The screenshot shows the 'Order Management' application interface with the breadcrumb trail 'Jeopardy Rules > Add Rule > Rule sample 1'. A 'SAVE' button is in the top right. The sidebar shows 'General Info', 'Conditions' (selected), and 'Actions'. The main area is titled 'Conditions' and has two sub-tabs: 'Condition Editor' (selected) and 'Expression'. Above the editor are buttons for 'AND', 'MATCH ALL' (with a dropdown arrow), and '+ ADD CONDITION'. The 'Condition Editor' shows 'Condition 1' with a visual expression builder. It has three main components: 'plan/orderRef (Left Operand)' with a blue circle icon, 'Operators' with a grey circle icon, and 'Right Operand' with a grey circle icon. Below the 'Left Operand' is a 'BACK' button and a list of options: 'Event Type' (with a minus icon) and 'Plan' (with a minus icon). The 'Plan' option is highlighted with a red rectangle. Below this list are several other options: 'Order ID', 'Order Ref.' (highlighted in blue), 'Plan ID', 'Status', 'Risk Region', and 'Predicted End'. At the bottom right of the editor, a 'NEXT' button is highlighted with a red rectangle. At the bottom of the main area are two buttons: 'SHOW EXPRESSION' and 'SAVE & CONTINUE' with a right-pointing arrow.

4. Select any of the following options as per your requirement and then click the **NEXT** button.

- Equals
- Equals Ignore Case
- Not Equals

The screenshot displays the 'Jeopardy Rules' configuration interface. The breadcrumb trail at the top reads 'Jeopardy Rules > Add Rule > Rule sample 1'. A 'SAVE' button is located in the top right corner. On the left, a sidebar contains 'General Info', 'Conditions', and 'Actions'. The main area is titled 'Conditions' and has two tabs: 'Condition Editor' (active) and 'Expression'. Above the editor, there are buttons for 'AND', 'MATCH ALL', and '+ ADD CONDITION'. The 'Condition Editor' shows 'Condition 1' with a visual representation: 'plan/orderRef (Left Operand)' connected to 'equalsIgnoreCase (Operators)' which is connected to 'Right Operand'. Below this, a 'BACK' button is on the left, and a 'NEXT' button is on the right, both highlighted with red rectangles. In the center, a red-bordered box contains three radio button options: 'Equals', 'Equals Ignore Case' (which is selected), and 'Not Equals'. At the bottom of the editor, there are 'SHOW EXPRESSION' and 'SAVE & CONTINUE' buttons.

5. On the **Right Operand** tab, provide the value in the box.

Jeopardy Rules > Add Rule > Rule sample 1

SAVE

General Info

Conditions

Actions

Conditions

Condition Editor Expression

AND MATCH ALL + ADD CONDITION

Condition 1

plan/orderRef (Left Operand)

equalsIgnoreCase (Operators)

ORDER_REF_1 (Right Operand)

BACK NEXT

Order Ref. ORDER_REF_1

This is helper text

CREATE CONDITION

SHOW EXPRESSION SAVE & CONTINUE

- Click the **CREATE CONDITION** button and then click the **SAVE & CONTINUE** button. On the **Conditions** tab, you can verify the Groovy syntax of the conditions by clicking on the **SHOW EXPRESSION** option.

Jeopardy Rules > Add Rule > Rule sample 1

SAVE

General Info

Conditions

Actions

Conditions

Condition Editor Expression

AND MATCH ALL + ADD CONDITION

```
(eventObject.getPlan().getOrderRef().equalsIgnoreCase(new String("ORDER_REF_1")))&&
(eventObject.getPlan().getPlanID().equalsIgnoreCase(new String("PLAN_ID_1")))
```

SAVE & CONTINUE

- On the **Actions** tab, enter the details for **Name** and **Description**, and then click the **SAVE & CONTINUE** button.

Jeopardy Rules > Add Rule > Rule sample 1 SAVE

General Info
Conditions
Actions

Actions + ADD ACTION

Action : 1 🗑️ ⬆️

General Notification Parameters Notification Template

Name
Action 1

Description
This is the description of action (optional)

SAVE & CONTINUE ➤

8. On the **Notification Parameters** tab, select EMS or Email from the **Channel** dropdown as per your requirement. Fill the details and click the **TEST CONNECTION** button.

EMS connection & Queue validation successful

Jeopardy Rules > Add Rule > Rule sample 1 SAVE

General Info
Conditions
Actions

Actions + ADD ACTION

Action : Action 1 🗑️ ⬆️

General Notification Parameters Notification Template

Outbound Message Type
String

Channel
EMS

Server Details

Destination
Queue

sample

▶ TEST CONNECTION SAVE & CONTINUE ➤

The screenshot shows the 'Actions' configuration page for a notification action. On the left, a sidebar contains 'General Info', 'Conditions', and 'Actions'. The main area is titled 'Actions' and has a '+ ADD ACTION' button in the top right. Below the title, there's a tabbed interface with 'General', 'Notification Parameters', and 'Notification Template'. The 'Notification Parameters' tab is active. It contains fields for 'Outbound Message Type' (set to 'String'), 'Channel' (set to 'Email'), 'Server Details' (Protocol: 'SMTP', Server Port: '567', Password: masked), 'Hostname' (set to 'smtp.gmail.com'), 'Username' (set to 'user@tibco.com'), 'From' (set to 'user@tibco.com'), 'To' (set to 'user1@tibco.com,user2@tibco.com'), and 'Subject' (set to 'This is the email subject'). There are also 'TEST CONNECTION' and 'SAVE & CONTINUE' buttons at the bottom.

9. Click the **SAVE & CONTINUE** button.
10. On the **Notification Template** tab, define the **Message Template** by using the operand of **Plan** or **Plan Item**. This is a layout to pick the respective attribute when sending the notification to the end user.

The screenshot shows the 'Notification Template' configuration page for a notification action. The breadcrumb trail at the top reads 'Jeopardy Rules > Add Rule > Rule sample 1'. A red box highlights a 'SAVE' button in the top right corner. The main area is titled 'Actions' and has a '+ ADD ACTION' button in the top right. Below the title, there's a tabbed interface with 'General', 'Notification Parameters', and 'Notification Template'. The 'Notification Template' tab is active. It contains a 'Message Template' field with the text 'The \${plan/orderID} status is changed to \${plan/status}'. A red box highlights this field. Below the field, there's a 'CREATE ACTION' button, also highlighted with a red box. The left sidebar contains 'General Info', 'Conditions', and 'Actions'. The 'Actions' section is expanded, showing 'Event Type' and 'Plan' (with a dropdown arrow). The 'Plan' dropdown is open, showing options: 'Order ID', 'Order Ref.', 'Plan ID', 'Status', 'Risk Region', and 'Predicted End'.

11. Click the **CREATE ACTION** button and then click the **SAVE** button.

Internal Error Handler

Internal Error Handler marks the failed plan items in ERROR state and gives you the control to select appropriate action for the failed plan item.

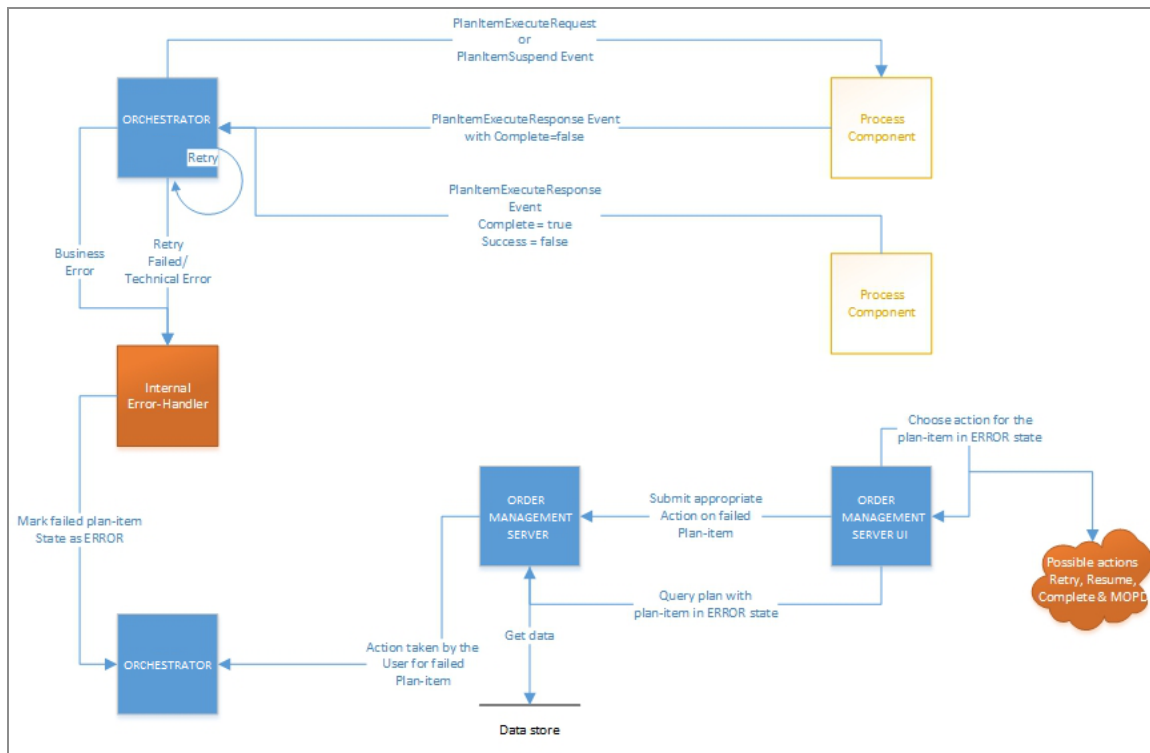
Internal Error Handler is an optional component and you can choose to configure internal error handler. By default, the external error handler is configured.

Internal Error Handler is designed to handle the failed plan-items. It handles the failed plan-items in two step process:

1. Mark the plan item state as ERROR for the plan item that failed.
2. Choose an appropriate action for the plan item in ERROR state from Order Management Server UI. You can choose appropriate action from the following options:
 - a. Retry
 - b. Resume
 - c. Complete

Internal Error Handler Data Flow Diagram

The following is the data flow diagram for Internal Error Handler:



Understanding Data Flow in Internal Error Handler

The following steps help you understand the data flow in the Internal Error Handler:

1. The Orchestrator sends the PlanItemExecuteRequest or PlanItemSuspend event to the process component for each plan item.
2. In response, the process component sends PlanItemExecuteResponse event.
3. In PlanItemExecuteResponse event we have two flags: Completed and Success, and based on value of these flags the orchestrator takes appropriate action. The following tables illustrates the same:

	Complete	Success	Description
Technical	False	False/True	Orchestrator retries the process

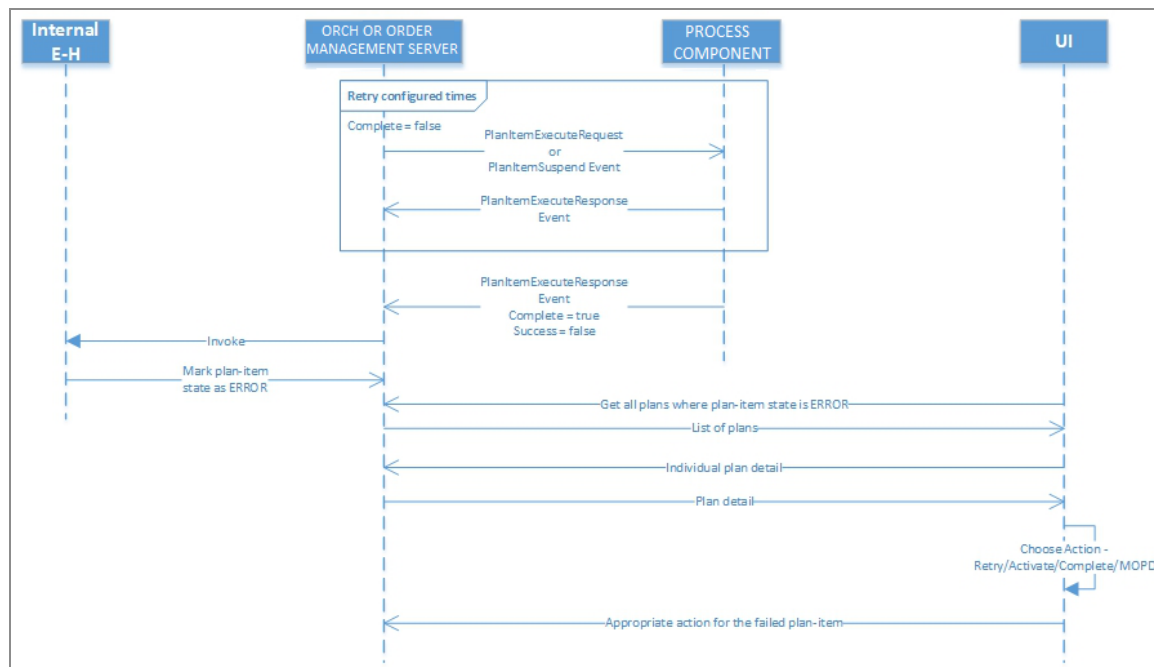
	Complete	Success	Description
Error			component call for the defined number of retries with the defined retry interval. If the process component call continues to fail, then it refers the plan item to the Plan Item Failed Handler.
Business Error	True	False	Orchestrator refers the plan item to the Plan Item Failed Handler.
Success	True	True	Processing continues as normal.

Steps 1 through 3 are part of existing implementation

4. The orchestrator invokes the plan item Error Handler according to your configuration. Going forward the system considers that you have configured Internal Error Handler, and refer the Internal Error Handler as Error Handler.
5. Plan item Error Handler changes the failed plan item state to ERROR.
6. Plan remains in EXECUTION with one or more plan items in ERROR state.
7. You can search for the plans with one or more plan items in ERROR state in Order Management Server UI.
8. After searching for the plan with planItem in the ERROR state, you can view the plan details.
9. You can access the plan item in ERROR state and take appropriate action on it by adding that order in the worktray from UI. Action on the plan item in ERROR state can be from one of the following options:
 - a. Retry
 - b. Resume
 - c. Complete
10. You have to submit the action taken for each failed plan item.
11. After your submission, the orchestrator initiates the action on the respective plan items.


Internal Error Handler Sequence Diagram

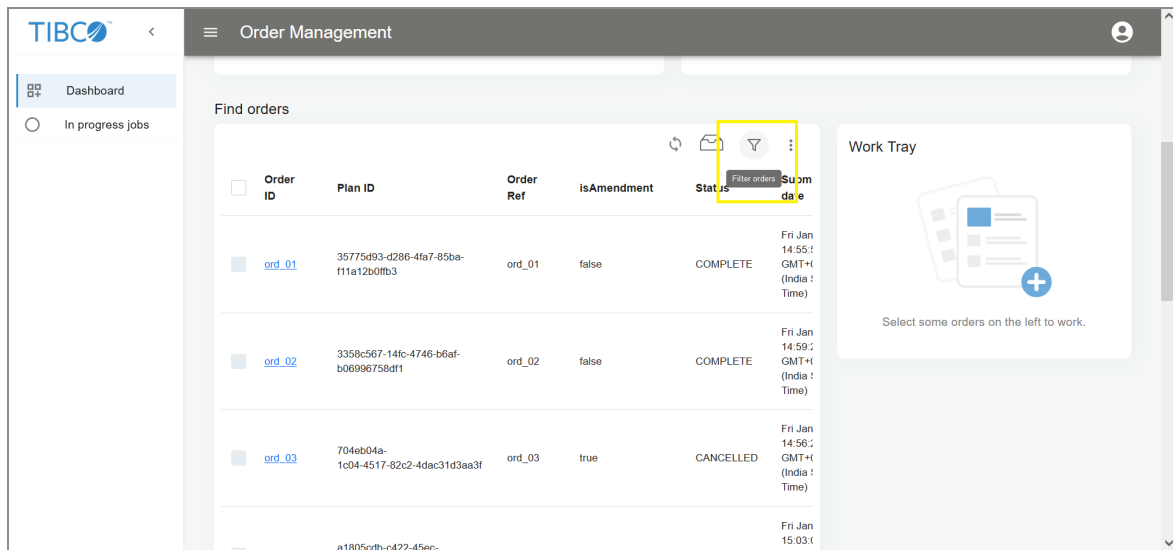
The following is the sequence diagram to show the actual sequence of the flow of data in Internal Error Handler:



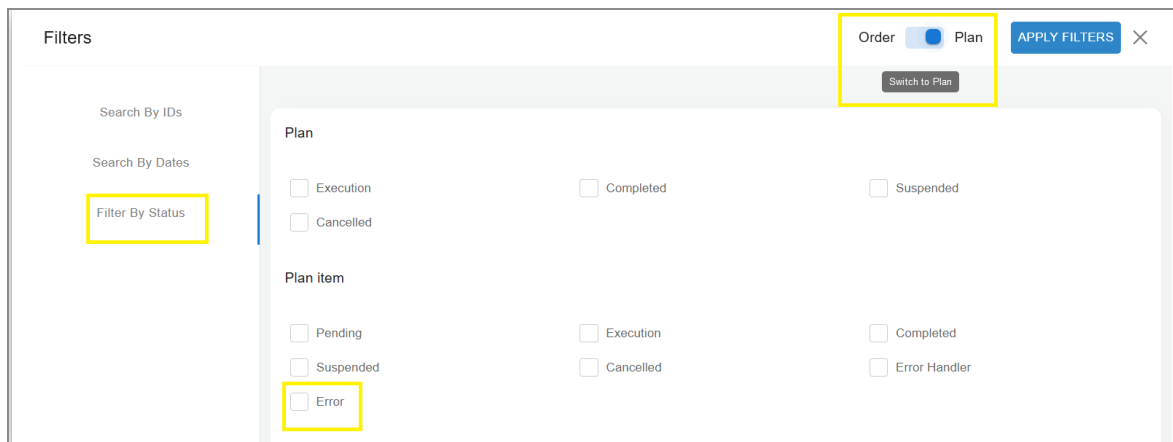
Searching for Plans with planItem in ERROR State

Procedure

1. Log in to Order Management System UI with valid credentials.
2. Click the  Filter orders icon on the **Find orders** table.



3. Toggle to plan filter.
4. Click **Filter By Status**, select the **Error** checkbox under **Plan item**.



Modifying the Plan Item State

After you have searched the plans with plan item or plan items in ERROR state, you have to add the desired plan in the **Worktray**.

You can choose the appropriate action on the plan item and submit the chosen action.

Find orders

<input checked="" type="checkbox"/>	Order ID	Plan ID	Order Ref	isAmendment	Status
<input checked="" type="checkbox"/>	ord_31	df52c4f8-df3f-4db4-a006-683847400ef4	ord_31	false	EXECUTION
<input checked="" type="checkbox"/>	ord_32	f3ac11c7-1b13-4d77-9abf-de87aee99594	ord_32	false	EXECUTION

Rows per page: 10 ▾ 1–2 of 2 < >

☐ Dense rows

Worktray

☒ Order ID Status

<input checked="" type="checkbox"/> ord_31	EXECUTION
<input checked="" type="checkbox"/> ord_32	EXECUTION

Rows per page: 10 ▾ 1–2 of 2 < >

CONTINUE ➤

Choosing the Error Resolution for the Plan Item in Error State

After adding the plan to the **Worktray**, click **CONTINUE**. You get the details for the plan item.

Click **Show Error based**. This plan item has the Status as ERROR and have the option so that you can choose the appropriate Error Resolution.

Plan items

Take an action ▾

☒ Show Error based error

<input type="checkbox"/>	Order ID	Plan ID	Plan item ID	Action	Status	Plan item name
<input type="checkbox"/>	ord_31	df52c4f8-df3f-4db4-a006-683847400ef4	1	PROVIDE	ERROR	EP_PF_PO_TV_Provide
<input type="checkbox"/>	ord_32	f3ac11c7-1b13-4d77-9abf-de87aee99594	1	PROVIDE	ERROR	EP_PF_PO_TV_Provide

Rows per page: 10 ▾ 1–2 of 2 < >

Error Resolution is a dropdown box that contains actions that you have to perform to fix the plan item in ERROR state. You have the following choices for the plan item's Error Resolution:

- Retry
- Resume
- Complete

The screenshot shows a table titled "Plan items" with a filter for "error" and a "Take an action" dropdown menu open, displaying "Retry", "Resume", and "Complete" options. The table has columns for Order ID, Plan ID, Action, Status, and Plan item name. Two items are listed, both in ERROR state.

Order ID	Plan ID	Action	Status	Plan item name
ord_31	df52c4f8-df3f-4004-a006-683847400ef4	PROVIDE	ERROR	EP_PF_PO_TV_Provide
ord_32	f3ac11c7-1b13-4d77-9abf-de87aee99594	PROVIDE	ERROR	EP_PF_PO_TV_Provide

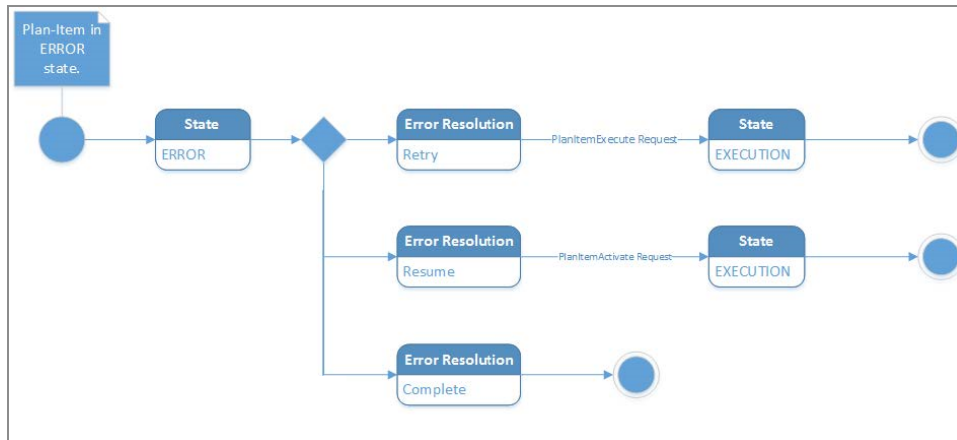
Rows per page: 10 1-2 of 2

There can be more than one plan item in ERROR state and you have to choose the error resolution for each plan item in ERROR state. In case you do not choose the error resolution for some plan items and submit the choice to the server, then the error resolution is run as per your choice for those few plan items and the rest of the plan items remain in ERROR state.

Details of Each Resolution Choice

You can choose any appropriate action from the error resolution dropdown box and the error resolution chosen is considered as a resolution choice for that particular plan item.

The following diagram shows the State Machine diagram for different states of the plan item after the user chooses a resolution type for the plan item in the ERROR state:



Error Resolution - RETRY

When you submit the error resolution with RETRY as the appropriate action, a new Plan Item Execute Request is sent for the plan item, and the orchestrator moves the plan item state from ERROR to EXECUTION.

Error Resolution - RESUME

When you submit the Error Resolution with RESUME as the appropriate action, a new Plan Item Activate Request is sent for the plan item, and the orchestrator moves the plan item state from ERROR to EXECUTION.

Error Resolution - COMPLETE

When you submit the error resolution with COMPLETE as the appropriate action, the plan item is marked as COMPLETE by the orchestrator. The orchestrator moves the plan item state from ERROR to COMPLETE.

Submit the Error Resolution

After taking the error resolution on the plan item in ERROR state, you can submit your changes. With error resolution choice as Retry, Resume, or Complete, you can submit these error resolutions for the plan items in ERROR state all in one go, or you can submit each error resolutions for the plan item in ERROR state separately.

Rows per page: 10 1-10 of 14

Request submitted successfully!

OK

☐ Show Error based

Search by Order ID, Plan

			Action	Status	Plan item name
ord_31	df52c4f8-df3f-4db4-a006-683847400ef4	1	PROVIDE	ERROR	EP_PF_PO_TV_Provide
ord_32	f3ac11c7-1b13-4d77-9abf-de87aee99594	1	PROVIDE	ERROR	EP_PF_PO_TV_Provide

Rows per page: 10 1-2 of 2

Order Management System User Interface

This section describes the TIBCO® Order Management System User Interface.

The application UI provides the following features:

- a visual interface to view order details, order execution plans.
- facility to search orders fast
- a complete view of orders that were fulfilled or failed during the fulfillment process
- end-to-end tracking, storing and monitoring capability for orders in the order fulfillment system
- capability to perform actions on the orders being executed in the system



Note:

The date is in the MM/DD/YYYY HH:MM:SS Z format, where Z is the time zone where the request is processed. For instance, UTC-7. You can configure the date from the Configurator.

You can perform the following actions on Order Management:

Order Management Actions	Description
Dashboard-specific actions	<p>Viewing Dashboard: View the Order Management Dashboard for summarized information about:</p> <p>Orders Panel</p> <ul style="list-style-type: none"> • Order Summary • Different charts <p>For details, refer to Dashboard.</p>
Order-specific	Order Management allows you to:

Order Management Actions	Description
actions	<ul style="list-style-type: none"> • View order Details • Search orders <p>For details, see Orders.</p>
Plan-specific actions	<p>Perform the following plan specific actions in the Order Management.</p> <ul style="list-style-type: none"> • View Plan Details • Search plan • Dependency View for the plan
Activity Log	<p>Shows the status and revision history of an object (order or a plan) logs based on the following criteria:</p> <ul style="list-style-type: none"> • Order Ref • Plan ID • Plan Item ID • Order Line <p>Note: All the Order Management related options are displayed only if Order Management configuration is enabled.</p> <p>For details, see Activity Log.</p>

Navigation

Access to the Dashboard is controlled through basic username, password, and tenantid authentication.

To access the TIBCO Order Management System from a browser window, perform the following steps:

Note: All the latest versions of Google Chrome, Mozilla Firefox, and Internet Explorer are supported by OMSUI.

Procedure

- To access the Login page, visit the `http://<host>:<port number>` URL, where:
 - Host is the computer where you installed the Order Management UI.
 - Port is the port number of the machine where the Order Management installation listens to the requests. The default port number is 9097.

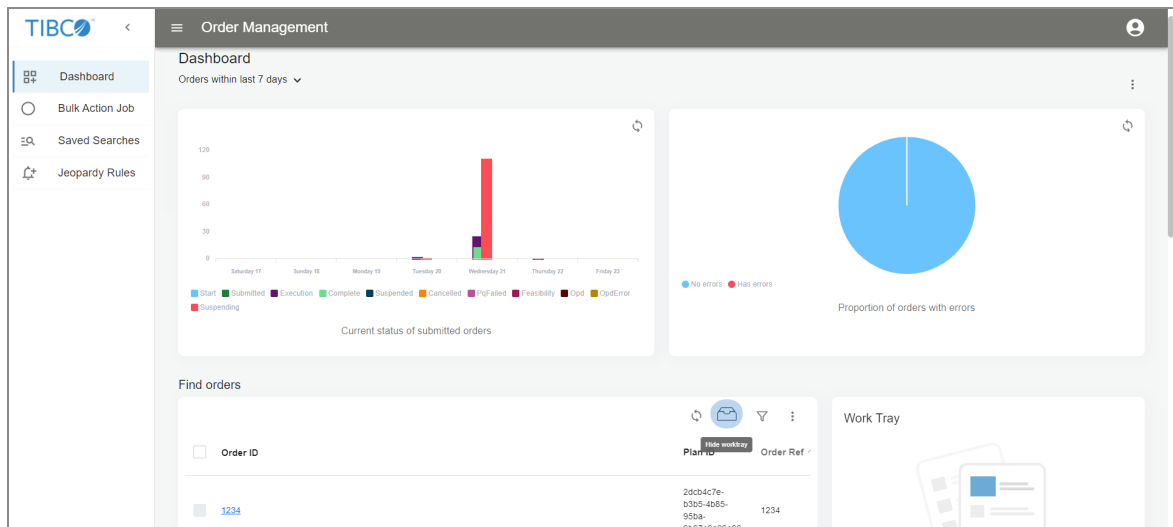
Note: You can configure the port from the `$OM_HOME/roles/configurator-ui/standalone/config/application.properties` file.

Order Management Log In

The screenshot displays the TIBCO Order Management login interface. At the top left is the TIBCO logo and 'Order Management' text. The main login area contains a 'Login' header with a lock icon. Below it are three input fields: 'TenantID *' (containing 'TIBCO'), 'Username *', and 'Password *'. To the right of the password field is a 'LOGIN WITH MICROSOFT' button. Below the password field is a blue 'LOGIN' button. A vertical line separates the password field from the Microsoft login button, with an 'OR' label in the middle. At the bottom of the login area, a small note reads 'TenantID is mandatory in all login types.'

- Enter the credentials in the **TenantID**, **Username**, and **Password** fields and then click **LOGIN**. The UI dashboard opens. (In case of OpenID Connect (OIDC), enter the **TenantID** details and click **LOGIN WITH MICROSOFT**.)

Order Management Dashboard



Note: In the case of OpenID Connect (OIDC), only the users with AdminROLE and UserROLE privileges can view the content. Users who are not in the added roles in the organization cannot view the content.

Dashboard

An Order Management Dashboard is a graphical user interface that organizes and presents rich and enhanced information in a format that is easy to read and interpret. The Dashboard is the default view when the user accesses the Order Management.

Features of the dashboard include:

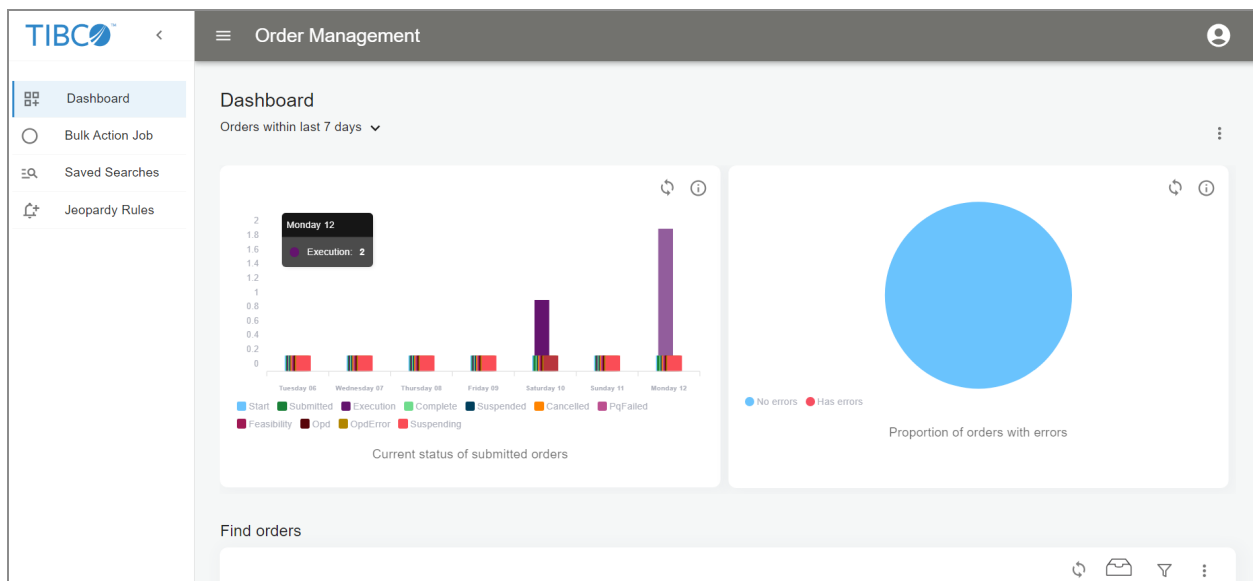
- **An intuitive graphical display that is easy to navigate** - A rich Graphical User Interface (GUI) with user/role security to manage/view orders.
- **A logical structure that makes information easily accessible** - Ability to view all orders through graphical Dashboard summary.
- **Data displays that can be customized and categorized** - Ability to drill down into order details by setting display preferences.
- **Regular and frequent updates of dashboard information for accuracy and relevance** - Ability to auto-refresh to display updated details for an order cancellation, amendment, suspension, and resumption.
- **Information from multiple sources can be viewed simultaneously** - Ability to

manage, search, and filter lists of existing orders.

The Dashboard allows effective order management with a comprehensive operations view. The information displayed is a combination of text and graphical views, as:

- Current number of orders being processed.
- Current number of orders completed in the last 24 hrs.
- Current number of orders in the Execution state.
- Current number of orders error out in the last 24 hrs.
- Current number of orders amended in the last 24 hrs.
- Current number of suspended orders.
- Current number of orders in Jeopardy.

OMS UI Auto-Refresh

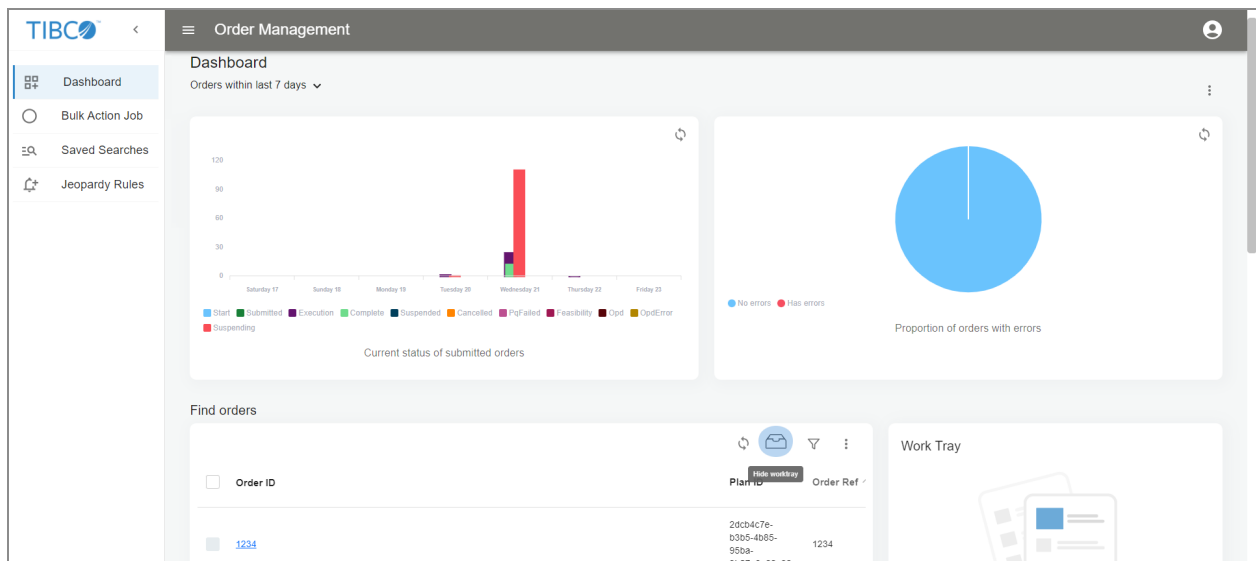


The OMS UI includes an auto-refresh feature that keeps the bar charts and pie charts up-to-date in real time by refreshing them every 20 seconds. This feature ensures that you always view the most current data without requiring manual refreshes.


When you hover over the bar chart, the count or value appears, providing immediate context and detailed information about the data point.

You can also customize the auto-refresh interval to suit your requirements. Navigate to **Order Management System UI > Authorization Server Configuration Properties** and set the dashboardAutoRefresh property as per your requirements.

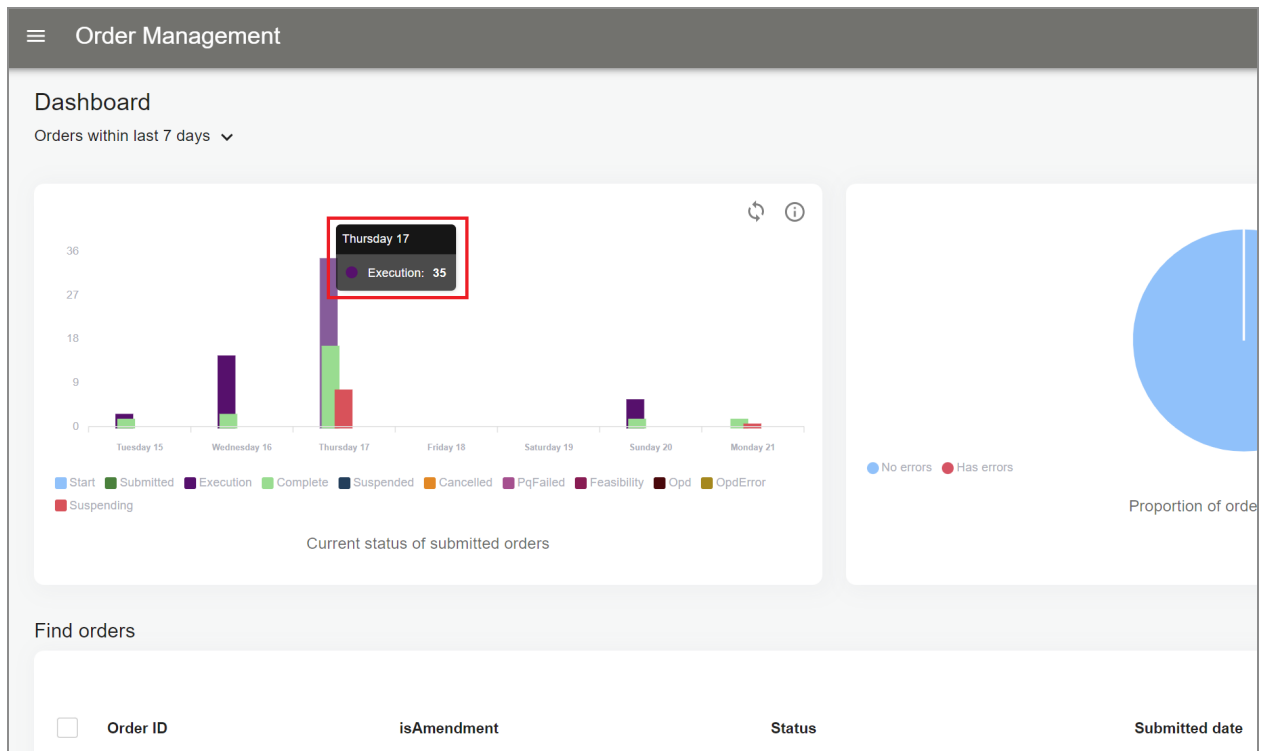
Charts



You can select the order date range from the dropdown options (such as 24 hours, 7 days, 15 days, and 30 days) below the **Dashboard**. By default, only 24 hours of data is fetched on the UI.

Click the vertical ellipsis icon , **Charts**, and then choose the desired charts from the box. The orders are shown in the following charts:

- **Current status of submitted orders**
Shows different status (such as Complete, Withdrawn) of the submitted orders
- **Proportion of orders with errors**
Shows the orders whose plan item statuses are in error or error handler state



When you hover over on an order chart, a query info icon shows the start date and end date of the order.

Pie chart calls to count only the number of orders with error and not the whole data. Thus loading time is reduced and any possible error for large plan items is avoided.

For bar chart, only **Order Status**, **Submitted Date**, and **Order Id** data is pulled instead of the whole data.

Find Orders

Order ID	Plan ID	Order Ref	Customer ID	Subscriber ID	isAmendment	Status	Submitted date
oe_1	50c15d8f-8485-48dc-afe7-63247ed1072b	oe_1		poja	false	EXECUTION	Wed Jun 21 2023 14:05:20...
ode_1	db6e8787-7b62-47ef-a86f-d274209e4287	ode_1		poja	false	EXECUTION	Wed Jun 21 2023 14:04:55...
ode_1234	b4e80226-4578-4b1a-ae9d-63ee891686d4	ode_1234		poja	false	EXECUTION	Wed Jun 21 2023 14:04:42...
ode_1235	fc09195-6980-4298-84b6-983a2499405d	ode_1235		poja	false	EXECUTION	Wed Jun 21 2023 14:04:46...
ode_15	bc72ab79-ab29-4687-8a46-a4863cee902	ode_15		poja	false	EXECUTION	Wed Jun 21 2023 14:04:51...

Here orders are displayed with their associated criteria such as **Order ID, Plan ID, Status** and so on.

You can select all the orders in one go by selecting the checkbox on the header row.

In this window, you can sort the orders to view them in an ascending or descending sequence of **Order ID, Plan ID, Status** and so on.



Note: The sorting is string-based and not the number-based.

Example of orders shown in an ascending manner:

- 1) 1
- 2) 125
- 3) 2
- 4) 7

The order Id 125 is placed on the second line, although its numeric value is greater than 2 and 7.

On the **Find orders** window, the following icons are present on the top-right corner:



Refresh icon: To refresh the page.



Work Tray icon: To navigate to the work tray.



Filter icon: To apply the search criteria by order level or plan level.

⋮ **Vertical ellipsis icon:** For settings

The screenshot shows the 'Find orders' interface. A table lists orders with columns: Order ID, Plan ID, Order Ref, Customer ID, Subscriber ID, and isAmendment. A dropdown menu titled 'Column Picker' is open, showing a search bar and a list of columns to choose from: Plan ID, Order Ref, Customer ID, Subscriber ID, isAmendment, and Status. The 'Column Picker' option is highlighted in the dropdown.

Order ID	Plan ID	Order Ref	Customer ID	Subscriber ID	isAmendment
Order_22June02	b3b7b7a7-eee3-49ef-819f-d88f36417a97	Order_22June02		123	false
Order_22June03	1c3c7072-4a4c-4979-b534-402be57f31f9	Order_22June03		123	false
Order_22June05	49236e16-9e2d-4547-b923-e8213f7b75db	Order_22June05		123	false
Order_22June06	0abd9c57-a9fe-4027-a06d-097b478db8bb	Order_22June06		123	false
Order_22June07	5c2af504-ae3-4311-82de-d6930c522921	Order_22June07		123	false

Click the vertical ellipsis icon ⋮, **Column Picker**, and then choose the desired columns from the box.

Note: The **Order ID** and **Submitted date** columns are mandatory columns and they are shown on the window even if you hide the rest all the columns.

You can sort the orders to view them in an ascending or descending sequence of **Order ID**, **Plan ID**, **Status** and so on.

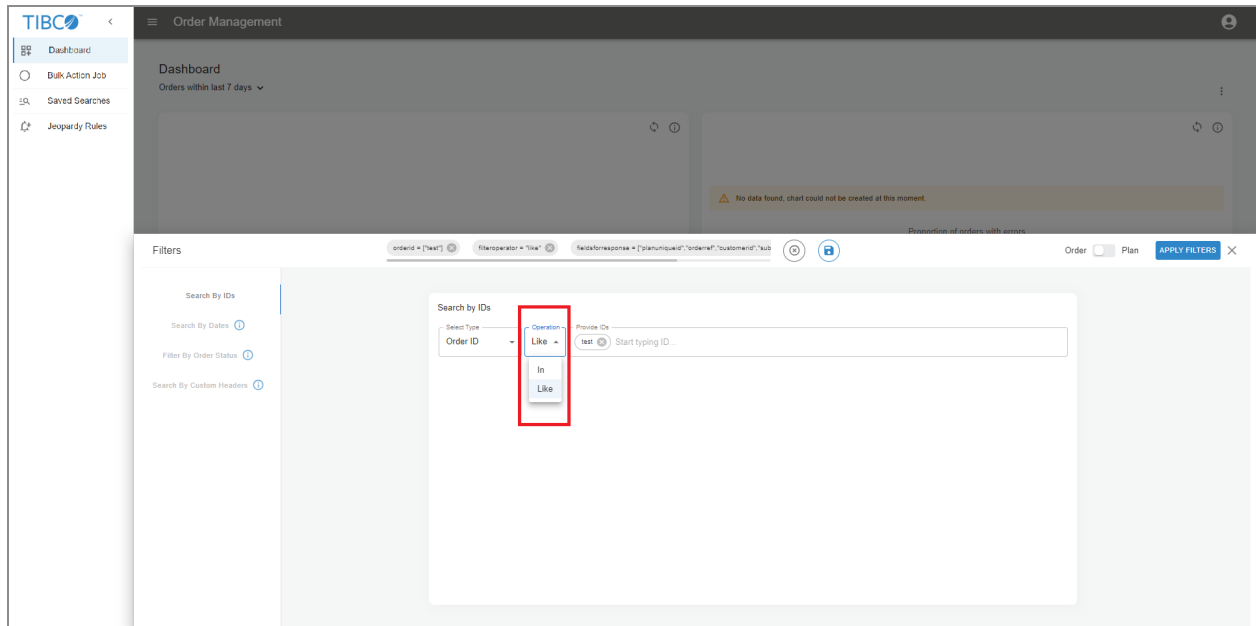
You can select 10 or 20 from the **Rows per page** dropdown options.

Filters

Initially, the entire orders are displayed here. You can create filters by clicking the **Filter orders** icon.

You can search an Order ID by typing some part of the ID instead of the entire order ID. From the **Operation** dropdown list, select **Like**. Then type some part of the order ID in the **Provide IDs** field and search for the order. All the orders with the order ID that match the given text are shown in the result.

When you do not want to use this functionality, you can select **In** from the **Operation** dropdown list. Here, only the exact order ID can be typed in the **Provide IDs** field.



Note: Whenever a user logs in to the OMS UI again, the last used search criteria are displayed by default.

On the **Filters** window, you can toggle between **Order** and **Plan** level.

Order Level Search Criteria

The orders can be searched by using the following criteria:

- [Search By IDs](#)
- [Search By Dates](#)
- [Search By Order Status](#)
- [Search By Custom Headers](#)



Search By IDs

Procedure

1. From the **Select Type** dropdown list, choose **Order ID**, **Order Ref**, **Customer ID**, or

Subscriber ID as per your requirement.

2. Enter the details in the **Provide IDs** field.

 **Note:** You can add multiple IDs by clicking add icon .

3. Click the **APPLY FILTERS** button.

 **Note:** When you use **Search By IDs** criteria, all other criteria are disabled.


Result

Orders are displayed with the applied filters on the **Find orders** window.

Search By Dates

Procedure

1. On the **Select Start and End date**, choose **Today, Yesterday, This Week, Last Week**, or any other available option there as per your requirement.
2. Click the **APPLY FILTERS** button.

 **Note:** You can use **Search By Dates** and **Filter By Order Status** criteria in a combination and all other criteria are disabled.

Result

Orders are displayed with the applied filters on the **Find orders** window.

Filter By Order Status

Procedure

1. On the **Select order status**, select the following checkboxes as per your requirement:
 - OPD

- OPD Error
- Execution
- Completed
- Suspending
- Suspended
- Canceled
- Pre qualification failed
- Feasibility
- Blocked

2. Click the **APPLY FILTERS** button.

i **Note:** You can use **Search By Dates** and **Filter By Order Status** criteria in a combination and all other criteria are disabled. You can also add multiple statuses in one filter criteria.


Result

Orders are displayed with the applied filters on the **Find orders** window.

Search By Custom Headers

Procedure

1. On the **Order Headers** or **Order Lines**, enter the details in the **Name** and **Value** fields.

i **Note:** You can use either **Order Headers** or **Order Lines** at a time. You can add new rows of the headers by clicking the **ADD**  icon.

2. Click the **APPLY FILTERS** button.

i Note: You can use **Search By Dates** and **Filter By Order Status** criteria in a combination and all other criteria are disabled.

Result

Orders are displayed with the applied filters on the **Find orders** window.

Plan Level Search Criteria

The plans can be searched by using the following criteria:

- [Search By IDs](#)
- [Search By Dates](#)
- [Filter By Status](#)

Search By IDs

Procedure

1. From the **Select Type** dropdown list, choose **Plan ID, Order ID, Order Ref, Process component ID, or Process component Name** as per your requirement.
2. Enter the details in the **Provide IDs** field.

i Note: You can add multiple IDs separated by comma in this field.

3. Click the **APPLY FILTERS** button.

i Note: When you use **Search By IDs** criteria, all other criteria are disabled.

Result

Orders are displayed with the applied filters on the **Find orders** window.

Search By Dates

Procedure

1. On the **Select Start and End date**, choose **Today, Yesterday, This Week, Last Week**, or any other available option there as per your requirement.
2. Click the **APPLY FILTERS** button.

i Note: You can use **Search By Dates** and **Filter By Status** criteria in a combination and all other criteria are disabled.

Result

Orders are displayed with the applied filters on the **Find orders** window.

Filter By Status

Procedure

1. Select the following **Plan** status or **Plan item** status checkboxes as per your requirement:

i Note: You can use either **Plan** status or **Plan item** status at a time.

Plan Status

- Execution
- Completed
- Suspending
- Suspended
- Canceled
- Withdraw

Plan Item Status

- Pending

- Execution
- Completed
- Suspended
- Canceled
- Error Handler
- Error

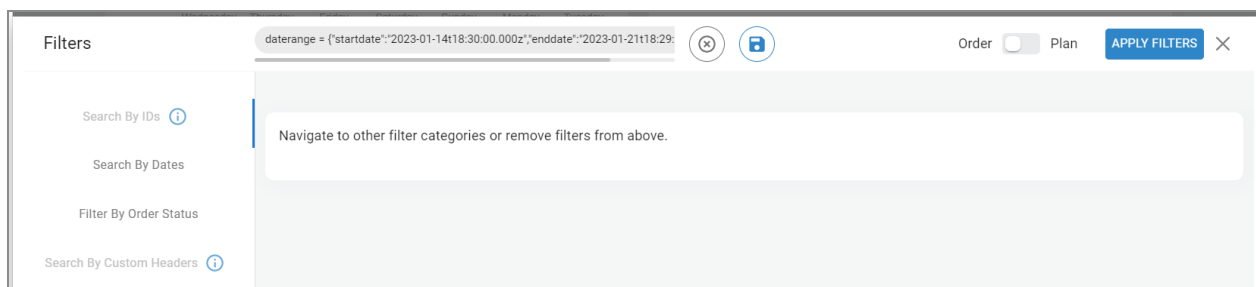
2. Click the **APPLY FILTERS** button.

Note: You can use **Search By Dates** and **Filter By Status** criteria in a combination and all other criteria are disabled. You can also add multiple statuses in the search criteria.

Result

Orders are displayed with the applied filters on the **Find orders** window.

Clearing and Saving a Search

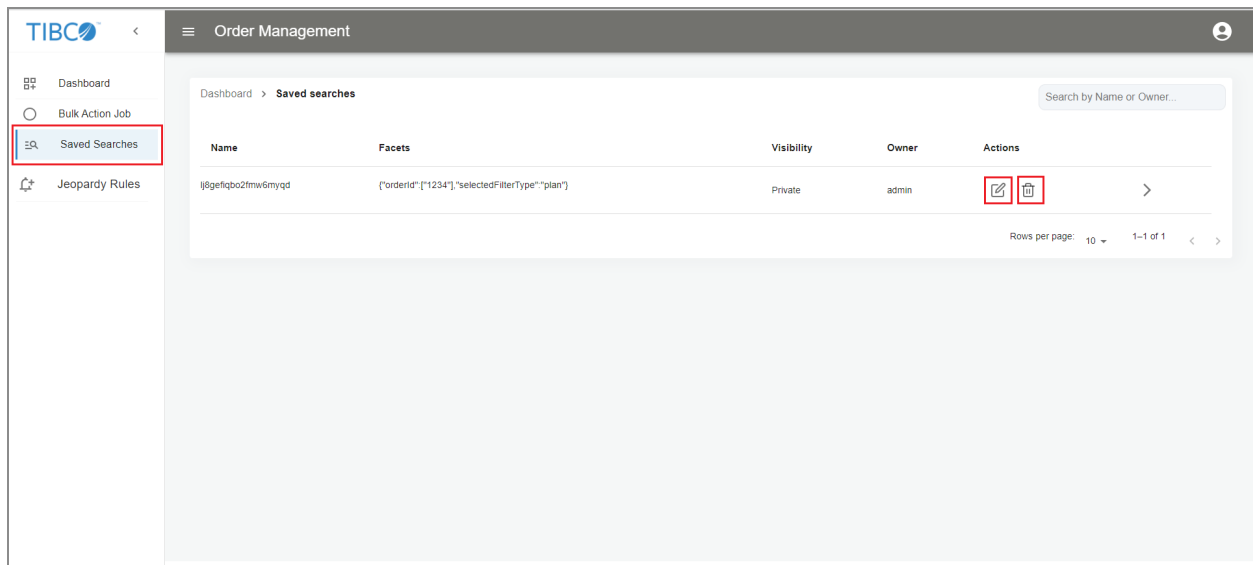


To clear a search criteria, click the  **clear filters** icon.



To save a search criteria, click the  **save search** icon.

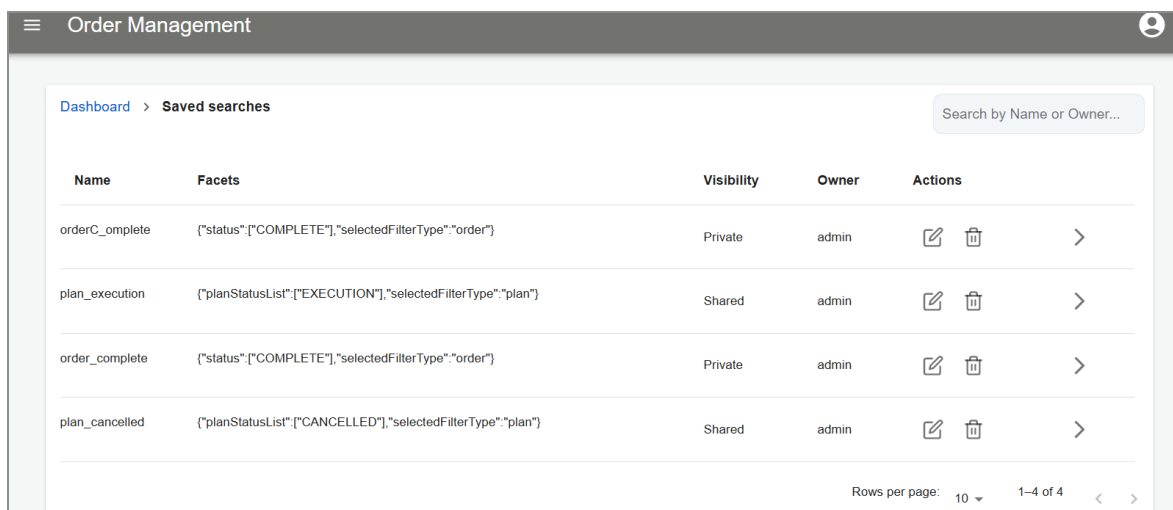
Note: While saving a search criteria, you can select the **Private** or **Shared** radio button. If you select **Private**, the search criteria are not visible to other users. If you select **Shared**, the search criteria are visible to other users.

Editing and Deleting Saved Search



Procedure

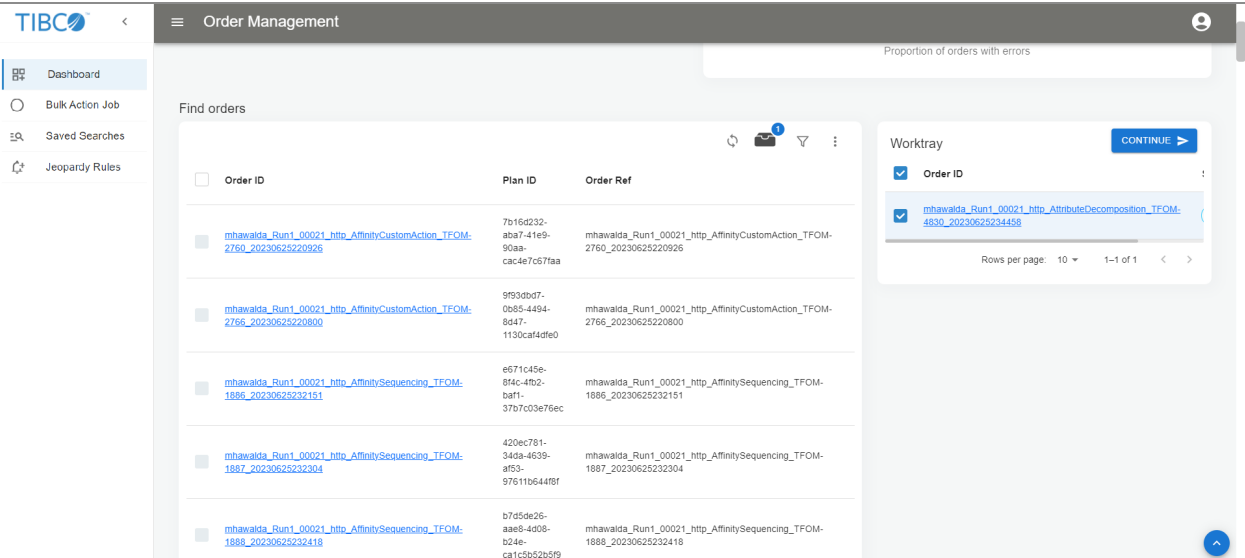
1. Click the **Saved Searches** tab on the left panel. A window opens showing all the saved search criteria.
2. Click the  edit icon to modify or the  delete icon to remove a search criteria.



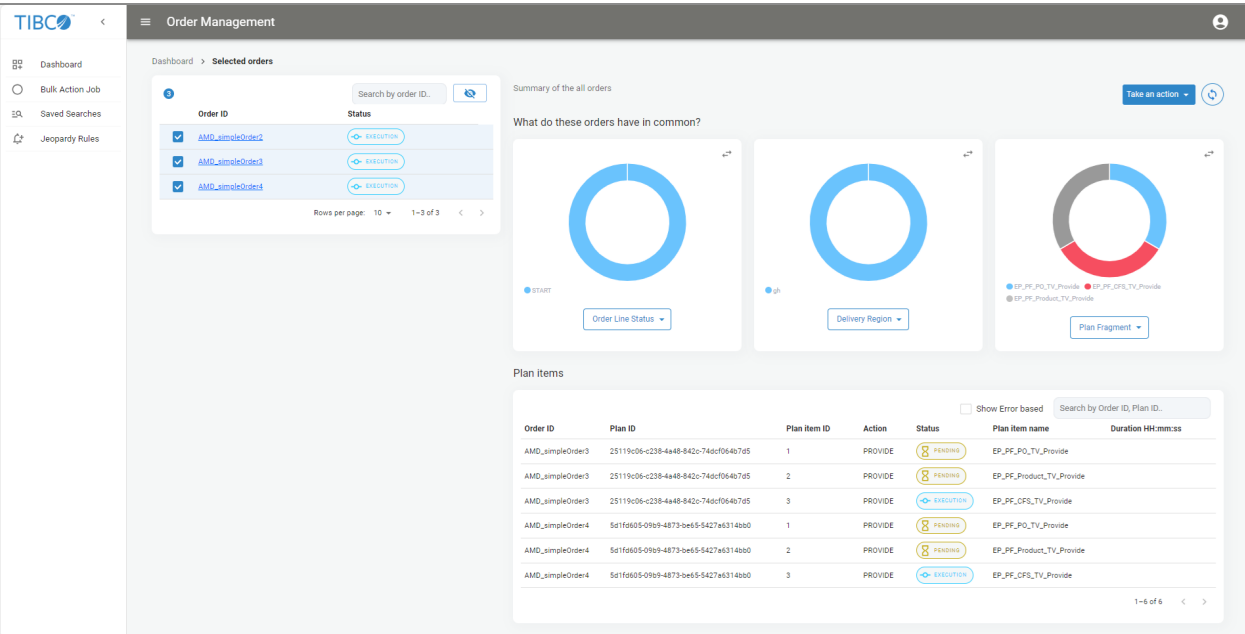
Note: Only the owner of the created criteria can edit or delete a search criteria.

Work Tray

Whenever you select a non-final state order from the **Find orders** window, the order is displayed on **Worktray**. This indicates that you are working on that particular order. Similarly, you can clear an order from **Worktray**.




On the **Worktray** window, click **CONTINUE**. The orders are displayed in a new window.



The orders chosen in the **Selected orders** window are reflected in the comparison Donut charts on the right.

The **Take an action** button has dropdown options as **Cancel**, **Resume**, **Suspend**, **Withdraw**, and **Retry**. Expand the **Take an Action** dropdown menu from the top-right corner and select the desired bulk action.

Note: In the \$OM_ HOME/seed-data/app-properties/ConfigValues_OMSUI.JSON file, you can set the operation.roles.amendOrder property, which is used to allow the types of user to perform amendment on Order Management System UI.

Click the refresh icon  to synchronize the selected orders.

Once an action is applied, the action details can be viewed on the **In progress jobs** tab. The **Job ID**, **Action**, **Job Details**, and **Orders** fields are shown here.

In progress jobs

Job ID	Action
606cfac2-f203-4259-90a0-cc13ee9f916c	SUSPEND

Rows per page: 10 ▾
1–1 of 1
<
>

Job Details

Job ID	606cfac2-f203-4259-90a0-cc13ee9f916c
Created Date	Tue Jan 31 2023 14:54:14 GMT+0530 (India Standard Time)
Action	SUSPEND
Requested by	admin
Tenant ID	TIBCO
Processed orders	0
Processing orders	2
Total orders	2

Order IDs

ord_33
ord_34

From the **In progress jobs** tab, you can view order details of a particular order by clicking the **Order IDs** hyperlink.

Note: Select two or more orders to view the comparison Donut charts and to take actions on order or plan items.

Orders

When you click the **Order ID** hyperlink, a new window opens with all the details about that particular order.



Note: The hyperlinks are disabled for the orders which are in final state. The hyperlinks work only for active orders. Withdrawn orders do not appear on the OMSUI.

The screenshot displays the TIBCO Order Management System User Interface. The top navigation bar includes the TIBCO logo and a hamburger menu. The main header is labeled "Order Management". On the left, a sidebar contains navigation options: Dashboard, Bulk Action Job, Saved Searches, and Jeopardy Rules. The main content area shows the "Selected orders" section with a list of orders. The selected order is "mhawalda_Run1_00021_http_AttributeDecomposition_TFOM-4830_20230625234458". Above this order, there are buttons for "Take an Action" and "Remove From Worktray". Below the order list, there are tabs for "Order Homepage", "Order Lines", "Plan Items", "Plan Items Dependencies", "Order Composition", "Plan Timeline", "Activity Log", and "Pending Tasks". The "Order Homepage" tab is active, showing "Basic info" for the selected order. This section is divided into two columns: "About this order" and "About this plan".

About this order		About this plan	
Order Ref ID	mhawalda_Run1_00021_http_AttributeDecomposition_TFOM-4830_20230625234458	Plan ID	08ae7d70-34c6-4fd3-aa6c-9c4450c52859
Order ID	mhawalda_Run1_00021_http_AttributeDecomposition_TFOM-4830_20230625234458	Risk region	There is no response from Jeopardy service!
Subscriber ID		Plan description	SingleUse
Customer ID	CUSTOMER		
Priority			
SLA			
Order description	SingleUse		

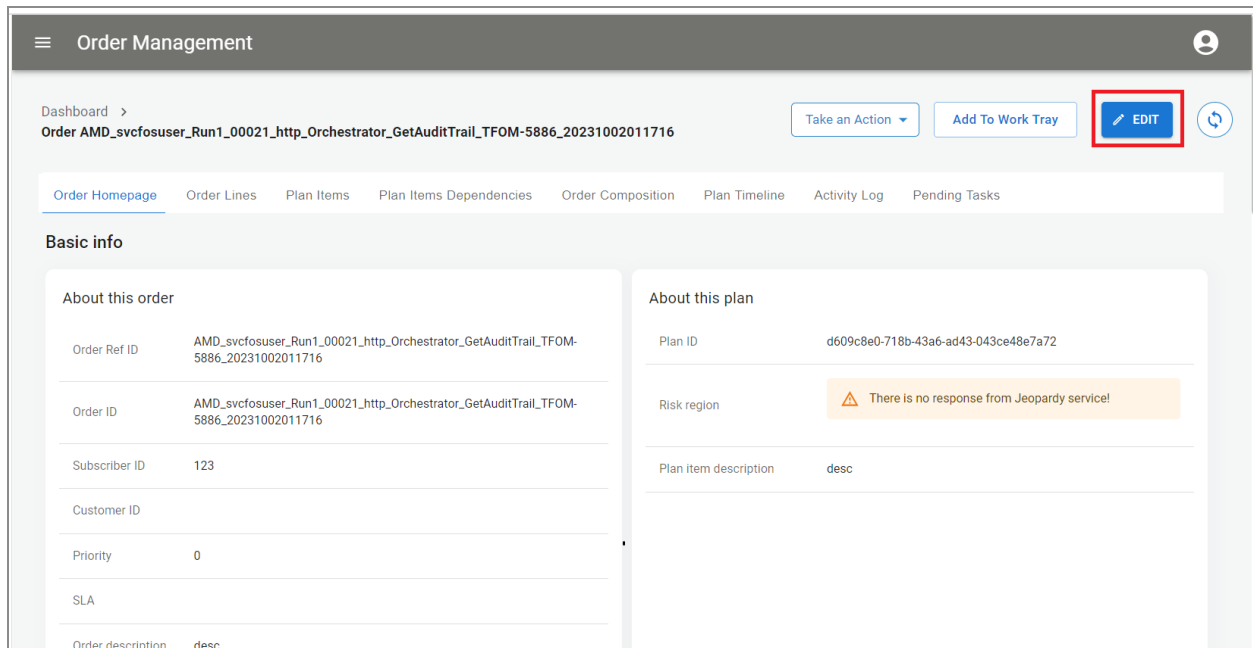
Order Homepage

The following information is available on the **Order Homepage** tab:

- Basic info (order and plan)
- Status (order and plan)
- Location (invoice address and delivery address)
- Dates (order and plan)
- Custom headers
- Plan
- Amendments (all amendments with clickable links to navigate to the amended order)

You can edit a suspended order to apply various amendments to the same.

Note: Only the user roles those are set for the `operation.roles.amendOrder` property under the "Application Security" category in the `$OM_HOME/seed-data/app-properties/ConfigValues_OMSUI.json` file, can amend orders on the OMS UI.



When you open a suspended order, an **EDIT** edit button is shown on the top-right corner for the **Order Homepage** and **Order Lines** tabs. By clicking the **EDIT** edit button, you can perform various amendments to the orders.

On the **Order Homepage** tab, edit the **Required by** field on the **Dates** section and information on the **Custom headers** section. You can edit any one out of these two sections at a time and need to restore the same before doing any edit on the other section.

- **Add a new order line:**

On the **Order Lines** tab, click the **EDIT** edit button on the top-right corner. Click the **+ ADD** add button, and then click the **+ CREATE NEW** create button to create an order line or click the **COPY** copy button to copy an existing order line.

Order Management

Dashboard > Order_AMD_11Oct06

PREVIEW & SUBMIT CANCEL

Order Homepage Order Lines Plan Items Plan Items Dependencies Order Composition Plan Timeline Activity Log Pending Tasks

1 Order Line

+ ADD

+ CREATE NEW

OR

Copy order line from below

Search

1

COPY

Provide

Quantity 1

Action Mode

Link ID

Subscriber ID mona

Unit of measure uom

Notes

Service Level Agreement (SLA)

Require by mm/dd/yyyy -- --

Require on

When you create an order line, initially a random line number is given to the order line. Fill all the required values in the **Order line Details**, **Custom headers**, and **Location** sections.

When you copy an order line, the values in the **Order line Details**, **Custom headers**, and **Location** sections are pre-populated from the parent one. You can edit the fields with the required values.

Order Management

1 Order Line

+ ADD

Order line Details

Line Number Value Inkhy220o8xs

Product ID Value VALIDATE

Product Version Value

Inventory ID Value

Action

Quantity Value 0

Action Mode Value

Link ID Value

Subscriber ID Value

Unit of measure Value

Notes Value

Service Level Agreement (SLA) Value

Require by 10/10/2023 04:06:03 PM

Require on 10/10/2023 04:06:03 PM

In the **Order line Details** section, you must provide a valid Product ID. You can click the **VALIDATE** button to check the validity of the product ID.

In the **Location** section, select the **Same as delivery address** checkbox if both the

delivery and invoice addresses are the same.

After filling out all the required details, click the **PREVIEW & SUBMIT** button. If there are some errors, those are shown as pop-up messages on the UI, and you can fix those.

The screenshot shows a 'Preview Amendment' window with two side-by-side panels. Each panel contains a table with the following structure:

description	IM																		
customerID	MILESTONE_CUSTOMER																		
InvoiceAddress	<table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>line1</td> <td>PUNE</td> </tr> <tr> <td>line2</td> <td>PUNE</td> </tr> <tr> <td>line3</td> <td>PUNE</td> </tr> <tr> <td>locality</td> <td>PUNE</td> </tr> <tr> <td>region</td> <td>PUNE</td> </tr> <tr> <td>country</td> <td>IN</td> </tr> <tr> <td>postCode</td> <td>1234</td> </tr> <tr> <td>supplementaryLocation</td> <td>PUNE</td> </tr> </tbody> </table>	Name	Value	line1	PUNE	line2	PUNE	line3	PUNE	locality	PUNE	region	PUNE	country	IN	postCode	1234	supplementaryLocation	PUNE
	Name	Value																	
	line1	PUNE																	
	line2	PUNE																	
	line3	PUNE																	
	locality	PUNE																	
	region	PUNE																	
	country	IN																	
postCode	1234																		
supplementaryLocation	PUNE																		
<table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>line1</td> <td>PUNE</td> </tr> </tbody> </table>	Name	Value	line1	PUNE															
Name	Value																		
line1	PUNE																		

At the bottom right of the window, there are 'CLOSE' and 'SAVE' buttons.

If there are no errors, the **Preview Amendment** window shows the old and new values. If there are no further changes, click the **SAVE** button and the new order line is added.

If you do not want a particular order line, you can remove the same by clicking the **Delete** icon.

- **Edit an existing order line:**

On the **Order Lines** tab, click the **EDIT** button on the top-right corner. Click an existing order line that you want to edit.

You can edit any one section from **Order line Details** or **Custom headers** at a time and need to restore the same before doing any edit on the other section.

On the **Order line Details** section, you can update the **Action** and **Require by** fields only.

After filling out all the required details, click the **PREVIEW & SUBMIT** button. If there are some errors, those are shown as pop-up messages on the UI, and you can fix those.

Preview Amendment

OLD		NEW																	
Name	Value	Name	Value																
orderID	AMD_mhavalda_Run1_00001_http_Crossfunctionality_IM_with_Amendment_TFOM-4874_20231227144744	orderID	AMD_mhavalda_Run1_00001_http_Crossfunctionality_IM_with_Amendment_TFOM-4874_20231227144744																
orderRef	AMD_mhavalda_Run1_00001_http_Crossfunctionality_IM_with_Amendment_TFOM-4874_20231227144744	orderRef	AMD_mhavalda_Run1_00001_http_Crossfunctionality_IM_with_Amendment_TFOM-4874_20231227144744																
^		^																	
submittedDate	1703668707.805	submittedDate	1703668707.805																
description	IM	description	IM																
customerID	MAILESTONE_CUSTOMER	customerID	MAILESTONE_CUSTOMER																
<table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>line1</td> <td>PUNE</td> </tr> <tr> <td>line2</td> <td>PUNE</td> </tr> <tr> <td>line3</td> <td>PUNE</td> </tr> </tbody> </table>		Name	Value	line1	PUNE	line2	PUNE	line3	PUNE	<table border="1"> <thead> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>line1</td> <td>PUNE</td> </tr> <tr> <td>line2</td> <td>PUNE</td> </tr> <tr> <td>line3</td> <td>PUNE</td> </tr> </tbody> </table>		Name	Value	line1	PUNE	line2	PUNE	line3	PUNE
Name	Value																		
line1	PUNE																		
line2	PUNE																		
line3	PUNE																		
Name	Value																		
line1	PUNE																		
line2	PUNE																		
line3	PUNE																		
invoiceAddress	<table border="1"> <tbody> <tr> <td>locality</td> <td>PUNE</td> </tr> <tr> <td>region</td> <td>PUNE</td> </tr> </tbody> </table>	locality	PUNE	region	PUNE	invoiceAddress	<table border="1"> <tbody> <tr> <td>locality</td> <td>PUNE</td> </tr> <tr> <td>region</td> <td>PUNE</td> </tr> </tbody> </table>	locality	PUNE	region	PUNE								
locality	PUNE																		
region	PUNE																		
locality	PUNE																		
region	PUNE																		

[CLOSE](#) [SAVE](#)

If there are no errors, the **Preview Amendment** window shows the old and new values. If there are no further changes, click the **SAVE** button and the changes are updated.

Order Lines

The following information is available on the **Order Lines** tab:

- Order Line
- Order Line Details
- Custom headers
- Location (invoice address and delivery address)

Dashboard > **Order Order_22June07**

Order Homepage **Order Lines** Plan Items Plan Items Dependencies Order Composition Plan Timeline Activity Log Pending Tasks

1 Order Line

Search by Line, Product ID or Status...

Line	Product ID	Action	Status
1	CFS_TV001_TFOM-5928	PROVIDE	COMPLETE

Rows per page: 10 1-1 of 1 < >


Order line Details

Line Number	1
Status	COMPLETE
Product ID	CFS_TV001_TFOM-5928
Product Version	1
Inventory ID	
Action	PROVIDE
Quantity	1
Action Mode	
Link ID	

Subscriber ID	
Unit of measure	
Notes	
Service Level Agreement (SLA)	
Require by	GMT+0530 (India Standard Time)
Require on	

Settings

- Items Table
- ☐ Pin Order Lines Table
- [Column Picker](#)

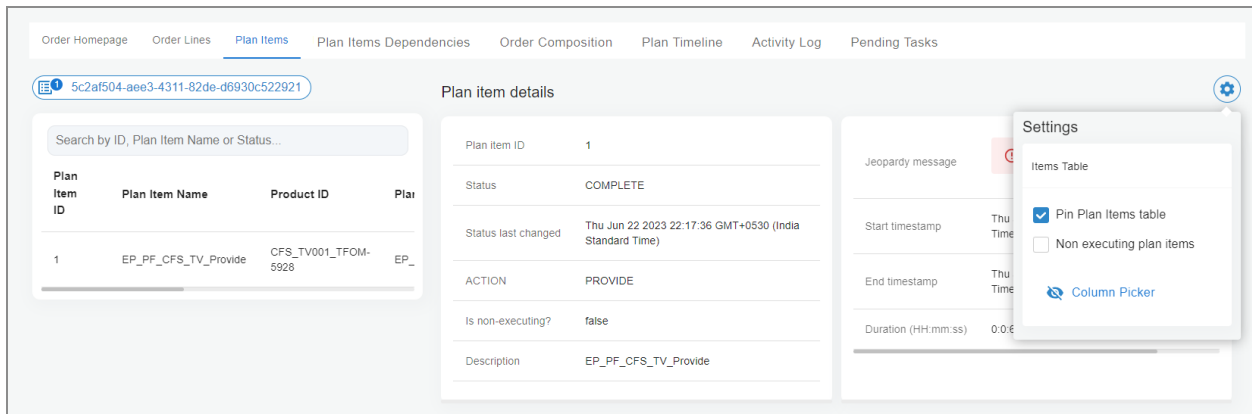
Click the settings icon  and select the **Pin Order Lines Table** checkbox to make the order lines scrollable. By default, the order lines are shown across the pages. You can hide or show the desired columns from the **Column Picker** option.

You can search an order line by **Line**, **Product ID**, or **Status** on the search box.


Plan Items

The following information is available on the **Plan Items** tab:

- Plan ID
- Plan items Details
- Process component information
- Custom headers
- Milestones (shows the dependencies if present)



The screenshot displays the 'Plan Items' tab in the Order Management System. The top navigation bar includes links for Order Homepage, Order Lines, Plan Items (active), Plan Items Dependencies, Order Composition, Plan Timeline, Activity Log, and Pending Tasks. A search bar with the ID '5c2af504-aea3-4311-82de-d6930c522921' is visible. The main content area is divided into two sections: 'Plan item details' and a table of plan items. The 'Plan item details' section shows information for Plan item ID 1, including Status (COMPLETE), Status last changed (Thu Jun 22 2023 22:17:36 GMT+0530 (India Standard Time)), ACTION (PROVIDE), Is non-executing? (false), and Description (EP_PF_CFS_TV_Provide). The table of plan items has columns for Plan Item ID, Plan Item Name, Product ID, and Plan Item Status. A settings overlay is visible on the right, showing options for 'Items Table' (Pin Plan Items table checked, Non executing plan items unchecked) and a 'Column Picker' link.

Click the settings icon  and select the **Pin Plan Items table** checkbox to make the plan items scrollable. By default, the plan items are shown across the pages. Similarly, you can select the **Non executing plan items** checkbox to show or hide the non-executing plan items. You can hide or show the desired columns from the **Column Picker** option.

You can search a plan item by **ID**, **Plan Item Name**, or **Status** on the search box.

You can take the **Retry**, **Resume**, or **Complete** action on the plan item with an error state.

Dashboard > Order AMD_09Oct04

Take an Action Add To Work Tray

Order Homepage Order Lines **Plan Items** Plan Items Dependencies Order Composition Plan Timeline Activity Log Pending Tasks

6b16ded4-b134-46c2-9e39-c7a390d5a649

Plan item details

Search by ID, Plan Item Name or Status...

Plan Item Name	Action	Status	Plan Item ID
NON_EXECUTING	PROVIDE	COMPLETE	NO
EP_PF_PO_TV_Provide	PROVIDE	ERROR	EP
EP_PF_Product_TV_Provide	PROVIDE	COMPLETE	EP
NON_EXECUTING	PROVIDE	COMPLETE	NO
EP_PF_PO_TV_Provide	PROVIDE	PENDING	EP

Plan Item ID: 1

Status: Error

Status last changed: 23 16:44:10 GMT+0530 (India S)

ACTION

Is non-executing?: false

Description: EP_PF_PO_TV_Provide

Jeopardy message: There is no response from Jeopardy se

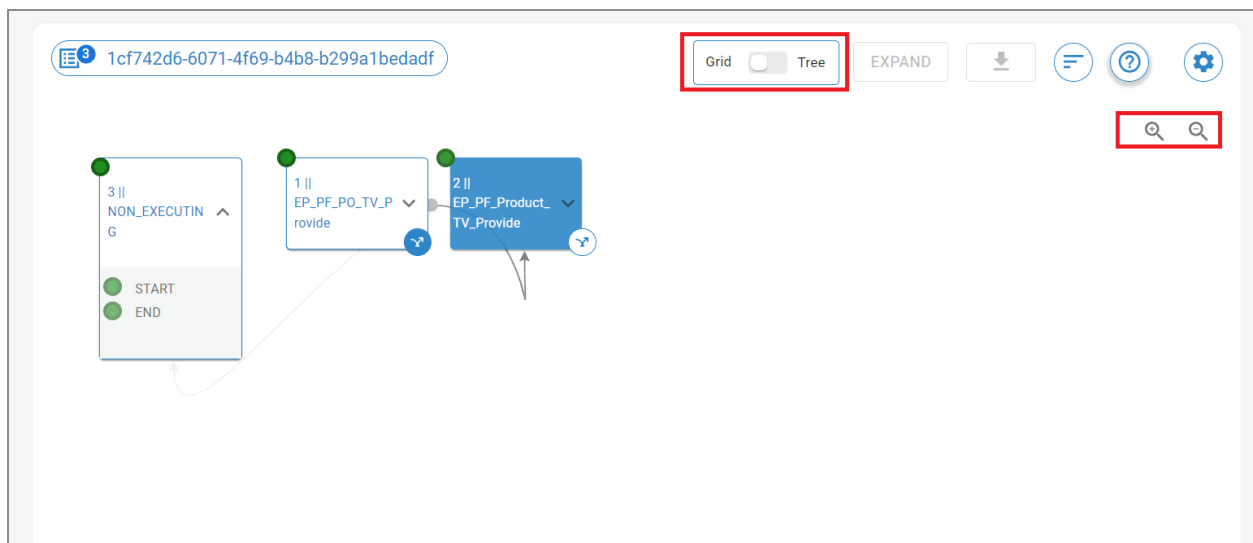
Start timestamp: Mon Oct 09 2023 16:39:36 GMT+0530 (India S)

End timestamp


Duration (HH:mm:ss)

Plan Items Dependencies

The **Plan Items Dependencies** tab shows the dependencies between milestones in a graphical representation.




You can choose the **Grid** view or **Tree** view by selecting the radio button on the top-right side. The Grid view is very helpful to show the dependencies efficiently if there are a large number of plan items.

You can click the link  icon to see the dependent plan item. When you click a plan item tile, it expands and shows the milestones. Right-click on a plan item tile to open the **Plan item details** window, where it shows in-depth details.

Click the zoom-in or zoom-out button on the right side to view the page as per your requirements.

Note: The zooming functionality works best with the Google Chrome browser and there might be some issues with other browser.

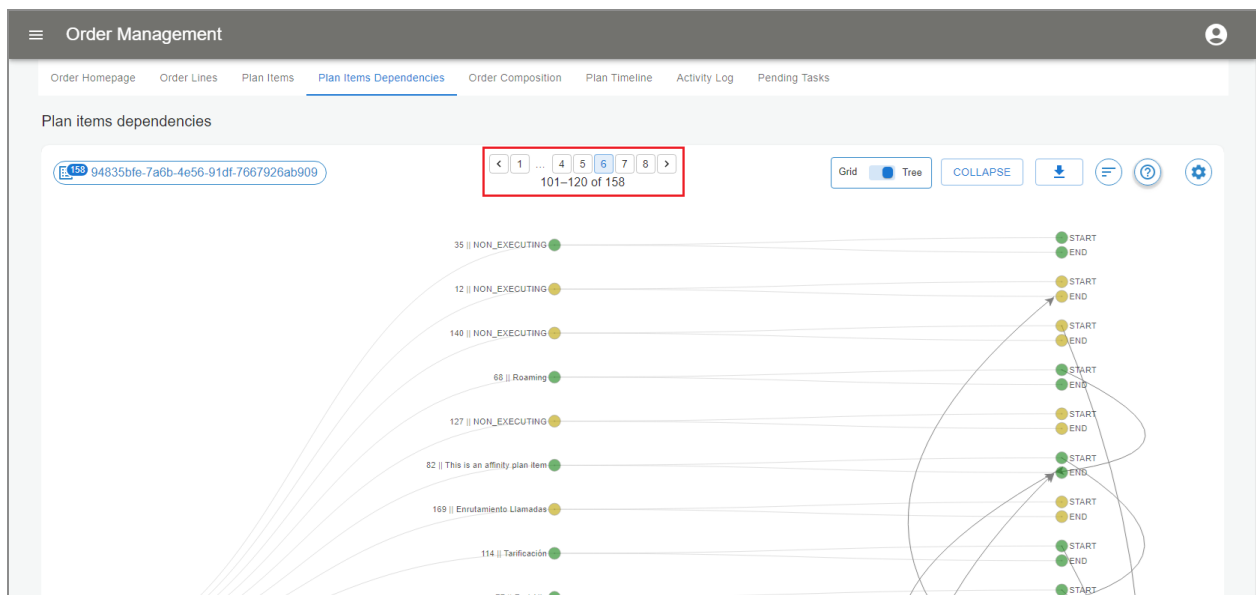
Click the sort icon  to sort the plan-items based on dependencies.

Click the help icon  to view the legend for the milestones (for example, START, PENDING, EXECUTION) with color coding.

Right-click on a node to view the complete details about the item.

Click the download icon  to save the dependency view as an image.



Click the settings icon  to show or hide various options as per your requirements.



You can choose the page you need from the top. This makes it easier to manage large amounts of data.

Order Composition

The **Order Composition** tab shows how a particular order is decomposed into various items in a flow chart.

When there are multiple levels of order lines in a catalog hierarchy, it delays significantly to load all levels of order lines. To overcome this issue, only the first levels of order lines are loaded initially on the UI. To see the next level of order lines, click the  expand button at the bottom of the first-level order lines. Similarly, you can continue expanding for all the levels of order lines. You can collapse a level of order line by clicking the  collapse button at the top of that level order line.

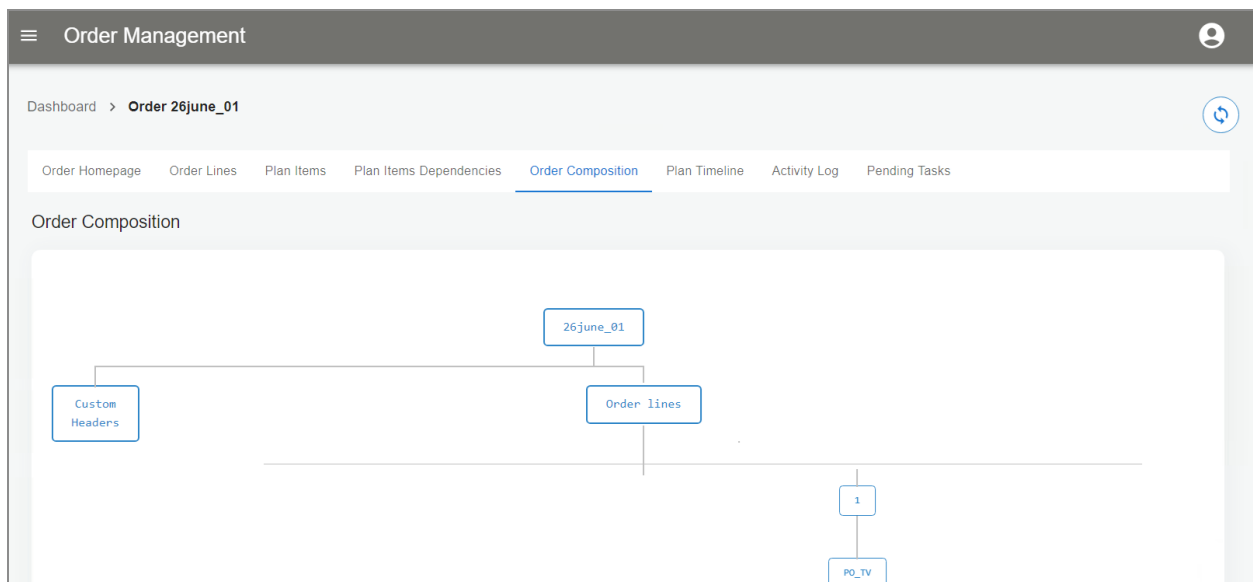
The order planId is sent from the OMS UI to the AOPD server.

In AOPD services, from planId, the services fetch the plans and generate a hierarchical response for the particular order. The response is then used to create the order composition chart on the UI.

To Order Composition view uses the following endpoint of AOPD:

```
/v1/plan/product/hierarchy
```

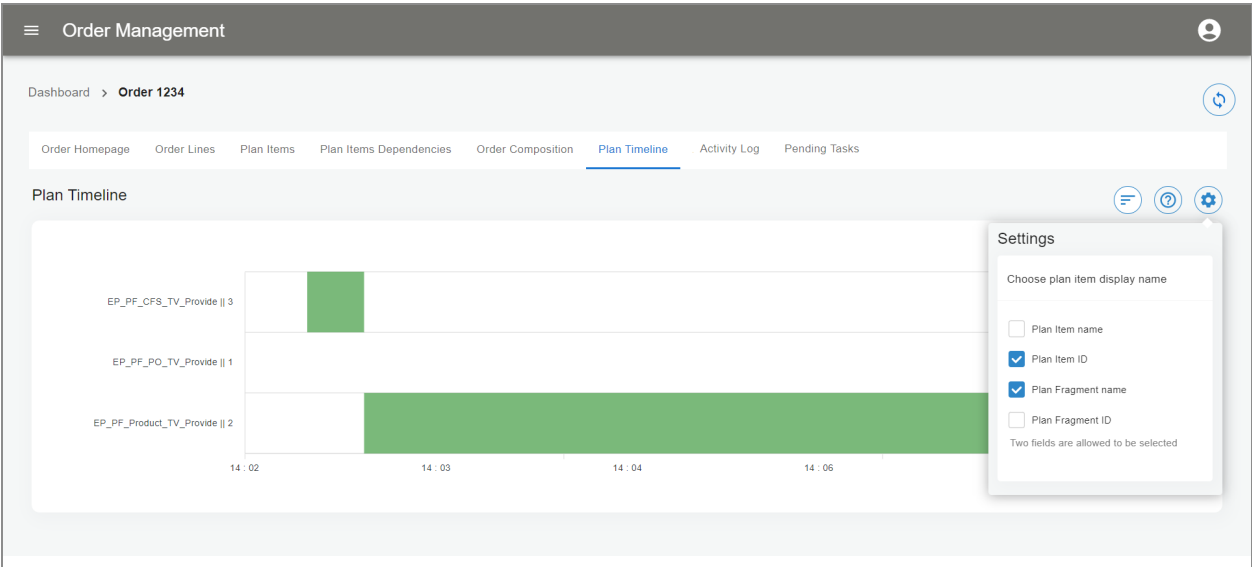
HTTP POST method is used for this endpoint. In request body, it takes AOPD plan and generates a JSON response that has parent-child relationship. The output of this endpoint is used to draw the Order Composition tree.



You can click an item to view more details.

Plan Timeline

The **Plan Timeline** tab shows the timeline of a plan in a chart. You can choose the plan item display names from the settings icon.



Activity Log

The **Activity Log** tab shows various transitions of the order.

Activity Logs

☒ Order ☒ OrderLine ☒ Plan ☒ PlanItem ☒ OrderAmendment ☒ Generic

Date & Time	Ref. ID	Type	Origin	Message
Tue Jun 25 2024 11:15:47...	25june_01...	ORDER	ORCHESTRATOR	ORDER status changed from FEASIBILITY to OPD
Tue Jun 25 2024 11:15:47...	25june_01...	ORDER	ORCHESTRATOR	ORDER status changed from PENDING to FEASIBILITY
Tue Jun 25 2024 11:15:52...	25june_01...	ORDER	ORCHESTRATOR	ORDER status changed from OPD to EXECUTION
Tue Jun 25 2024 11:15:52...	ce2d8cc0-10a9-4b6c-9951-f343b5529d7d	PLAN	ORCHESTRATOR	PLAN status changed from PENDING to EXECUTION
Tue Jun 25 2024 11:15:52...	1	ORDERLINE	ORCHESTRATOR	ORDERLINE status changed from START to PENDING
Tue Jun 25 2024 11:15:52...	3	PLANITEM	ORCHESTRATOR	PLANITEM status changed from PENDING to COMPLETE
Tue Jun 25 2024 11:15:52...	2	PLANITEM	ORCHESTRATOR	PLANITEM status changed from PENDING to EXECUTION

Rows per page: 10 1-7 of 7 < >

You can select or clear the checkboxes for **Order**, **Orderline**, **Plan**, **Planitem**, **OrderAmendment**, and **Generic** to view the related logs that you want. The log shows details such as **Date & Time**, **Ref. ID**, **Type**, **Origin**, and **Message**.

Pending Tasks

You can re-trigger any pending task for a non-final state orders through the **Pending Tasks** tab.

All the pending tasks are shown on this tab. Before re-triggering a pending task, you can select the required state from the **Reply** dropdown options.

Depending on the **Pending Task**, the **Retrigger** function and **Reply** dropdown options varies. The following table shows such pending tasks and their available options.

Pending Task	Retrigger	Reply	Result
PLAN_ITEM_EXECUTE_REQUEST	RE-TRIGGER	Complete Error / Error Handler	In case of re-trigger option, the application sends planItemExecuteRequest again to the southbound system. In case of reply option, plan item goes to complete state or error / error_handler state based on the selected

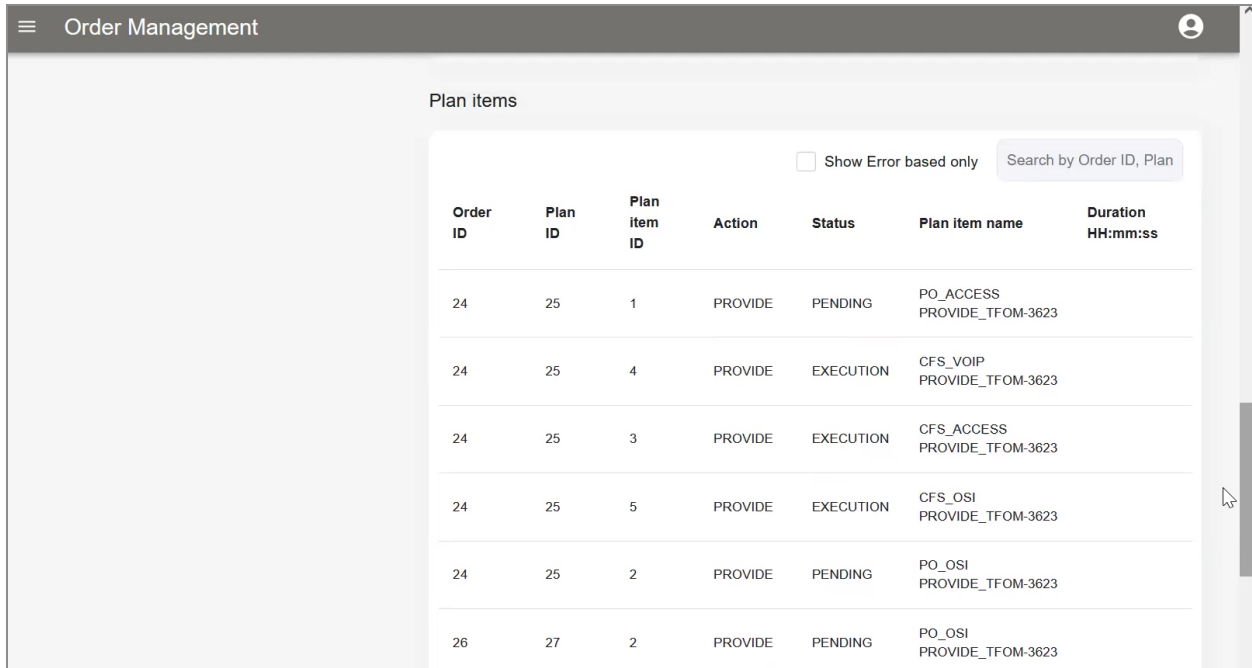
Pending Task	Retrigger	Reply	Result
			option.
PLAN_ITEM_ SUSPEND_REQUEST	SUSPEND	-	Plan item goes to the suspended state.
PLAN_ITEM_ ACTIVATE_REQUEST	RE-TRIGGER	Complete Error / Error Handler	In case of re-trigger option, application sends planItemExecuteRequest again to the southbound system. In case of reply option, plan item goes to complete state or error / error_handler state based on the selected option.
PLAN_ITEM_FAILED_ REQUEST	RE-TRIGGER	-	In case of re-trigger, the application sends PLAN_ITEM_FAILED_REQUEST again.
ORDER_FEASIBILITY	RE-TRIGGER	-	In case of re-trigger, the application sends ORDER_FEASIBILITY again.
AMEND_ORDER_ FEASIBILITY	RE-TRIGGER	-	In case of re-trigger, the application sends AMEND_ORDER_FEASIBILITY again.
MILESTONE_ RELEASE_REQUEST	RE-TRIGGER	-	In case of re-trigger, the application sends MILESTONE_RELEASE_REQUEST again.
PRE_QUALIFICATION_ FAILED	RE-TRIGGER	-	In case of re-trigger, the application sends PRE_QUALIFICATION_FAILED again.

You can hide or show the desired columns from the **Column Picker** option.

Plan items

Add non-final state orders in the **Worktray** from **Find orders table** and click **CONTINUE**.

On this window, all the plan items are displayed that are chosen on the **Selected orders** window.



Order ID	Plan ID	Plan item ID	Action	Status	Plan item name	Duration HH:mm:ss
24	25	1	PROVIDE	PENDING	PO_ACCESS PROVIDE_TFOM-3623	
24	25	4	PROVIDE	EXECUTION	CFS_VOIP PROVIDE_TFOM-3623	
24	25	3	PROVIDE	EXECUTION	CFS_ACCESS PROVIDE_TFOM-3623	
24	25	5	PROVIDE	EXECUTION	CFS_OSI PROVIDE_TFOM-3623	
24	25	2	PROVIDE	PENDING	PO_OSI PROVIDE_TFOM-3623	
26	27	2	PROVIDE	PENDING	PO_OSI PROVIDE_TFOM-3623	

Also, plan items can be searched with their order ID or plan ID on the search box.

You can select the checkboxes against the **Show error based only** component to view the plan items with error state. When plan items with error states are filtered out, the **Take an action** button is enabled with the dropdown options as **Retry**, **Resume**, and **Complete**. You can take an action as required.

Data Access Interfaces

TIBCO Order Management provides the options to access or update the user-defined field data during order fulfillment.

The following are the interfaces:

1. **GetPlan** - to get the User Defined Fields data of the execution plan associated with an order. This can also include the User Defined Fields data of all comprising plan items.
2. **GetPlanItems** - to get the User Defined Fields data of the specific plan items passed in the request.
3. **SetPlan** - to update or replace the User Defined Fields data of the execution plan associated with an order. This can also include the User Defined Fields data of one or more comprising plan items.
4. **SetPlanItem** - to update/replace the User Defined Fields data of a specific plan item passed in the request.

Get Plan

The **GetPlan** interface can be used by the process components to retrieve the plan corresponding to an order from the TIBCO Order Management system.

If an exception occurs during the **GetPlan** operation, it is logged into the Data Service log. The details of the exception are returned in the response.

Get Plan Request

The following properties must be passed in the request message header:

Request Endpoint: /v1/plan/get

Element	Type	Cardinality	Description
businessTransactionID	String	Optional	The unique identifier for tracing purposes across function calls.
planID	String	Required	The internal unique identifier for the plan to retrieve.
correlationID	String	Optional	The unique identifier for tracing purposes across a single function call. This is generally used by the calling application to correlate requests and responses.
idsOnly	Boolean	Optional	If <code>true</code> , only returns the IDs of elements rather than all plan data. If <code>false</code> , returns all plan data.
includeItems	Boolean	Optional	If <code>true</code> returns all plan items with the plan. If <code>false</code> , only the plan details are returned.
matchingUdfNameAllPi	String	Optional	The value of this field is in the format: UDF name=UDF value. When this value is set, the output is a plan with the plan items, which have the matching user-defined field name and value

Element	Type	Cardinality	Description
			<p>as specified in the header value together with all the User Defined Fields for the corresponding matched plan items.</p> <p>If the format for this user-defined field is not in the form UDF name=UDF value, an error is returned.</p>
ALL_PI_SINGLE_UDF_NAME	String	Optional	<p>The value of this is a user-defined field name. The output is a plan with all the plan items and the plan items have only the user-defined field specified in the header if present.</p> <p>If both MATCHING_PI_UDF_NAME_VAL and ALL_PI_SINGLE_UDF_NAME are present in the request, an error is returned.</p>
matchingUdfNameAllPiIgnoreEmpty	Boolean	Optional	<p>This is intended to be used in conjunction with ALL_PI_SINGLE_UDF_NAME. The default value of false indicates that all plan items are present in</p>

Element	Type	Cardinality	Description
			the response even if they do not have a user-defined field matching the user-defined field specified in ALL_PI_SINGLE_UDF_NAME. A true value indicates that plan items with no matching User Defined Fields are removed from the response.
OrderID	String	Required	<p>The internal unique identifier for the order.</p> <p>Note: The value of orderId must not contain ":"</p>

i Note: Any one of the orderID, orderRef, or planID is mandatory in the REST request calls.

Get Plan Response

The response message contains the following header properties:

Element	Type	Cardinality	Description
success	Boolean	Required	The flag indicating if the call was successful.

Element	Type	Cardinality	Description
messageCode	String	Required	The result message code.
message	String	Required	The result message.
planID	String	Required	The internal unique identifier for the plan specified in the request.
found	Boolean	Required	The flag indicating if the plan was found.

The following table lists the details of the response elements.

Element	Type	Cardinality	Description
resultStatus	Type	Required	The result status type. See Schema References for the specification of this type.
plan	Type	Optional	The plan type. If the plan is not found, this is omitted.
plan/planID	String	Required	The internal unique identifier for the plan.
plan/orderID	String	Required	The internal unique identifier for the order for the plan. Note: The value of orderId must not contain ":"
plan/orderRef	String	Required	External unique identifier for the order for the plan.

Element	Type	Cardinality	Description
			Note: The value of orderRef must not contain ":"
plan/udf	Type	0-M	user-defined field type.
plan/udf/type	String	Optional	Type of the user defined field.
plan/udf/flavor	String	Optional	Flavor of the user-defined field. The valid values are one of the following three: <ul style="list-style-type: none"> • config - For user-defined field corresponding to a characteristic in the product model or a system user-defined field generated by Automated Order Plan Development. • input - For user-defined field passed in the order. • output - For user-defined field set by the process component.
plan/udf/name	String	Required	Field name.

Element	Type	Cardinality	Description
plan/udf/value	String	Optional	Field value.
plan/udf/originalValue	String	Optional	Original field value at the time of plan creation.
plan/udf/lastModified	DateTime	Optional	Timestamp when the user-defined field was last modified.
plan/planItem	Type	0-M	Plan item type.
plan/planItem/planItemID	String	Required	Internal unique identifier for the plan item.
plan/planItem/planItemName	String	Optional	Name of the process component.
plan/planItem/udf	Type	0-M	user-defined field type.
plan/planItem/udf/type	String	Optional	Type of the user defined field.
plan/planItem/udf/flavor	String	Optional	<p>Flavor of the user-defined field. The valid values are one of the following three:</p> <ul style="list-style-type: none"> • config - For user-defined field corresponding to a characteristic in the product model or a system user-defined field generated by Automated Order

Element	Type	Cardinality	Description
			Plan Development. <ul style="list-style-type: none"> • input - For user-defined field passed in the order. • output - For user-defined field set by the process component.
plan/planItem/udf/name	String	Required	Field name.
plan/planItem/udf/value	String	Optional	Field value.
plan/planItem/udf/originalValue	String	Optional	Original field value at the time of plan creation.
plan/planItem/udf/lastModified	DateTime	Optional	Timestamp when the user-defined field was last modified.
plan/planItem/parentID	String	0-M	IDs of the plan items, which depend on the current plan item.
plan/planItem/childID	String	0-M	IDs of the plan items on, which the current plan item depends.
plan/planItem/siblingID	String	0-M	IDs of the plan items corresponding to SIBLING_PRODUCT_* of the product fulfilled by current plan item.

Element	Type	Cardinality	Description
plan/planItem/dependentID	String	0-M	IDs of the plan items corresponding to DEPENDENT_PRODUCT_* of the product fulfilled by current plan item.
plan/status	String	Required	Status of the plan from a data perspective: active or complete.
plan/planDescription	String	Optional	Plan description.

Get Plan Messages and Message Codes

The error codes and their respective error messages for the Get Plan interface are as follows:

Get Plan Messages and Message Codes

Message Code in Response Header and Result Status	Message in Response Header and Result Status	Scenario
FOM-DATA-ACCESS-SUCCESS-0000	Successfully processed GetPlanRequestEvent	Plan is found and data is mapped in the response successfully.
FOM-DATA-ACCESS-PLAN-NOT-FOUND-9999	Plan not found for planID <planID value>	Plan is not found against the planID specified in the request header.
FOM-DATA-ACCESS-ORDER-NOT-FOUND-9999	Order not found for orderID <orderID value>	Order is not found against the orderID specified in the request header.

Get Plan Items

The `GetPlanItems` interface can be used by the process components to retrieve the data of one or many plan items in the plan corresponding to an order from the TIBCO Order Management system.

If an exception occurs during `GetPlanItems` operation, then it is logged into the Data Service log. The details of the exception are returned in the response.

Get Plan Items Request

Request Endpoint: `/v1/planitems/get`

The following properties must be passed in the request message header:

Element	Type	Cardinality	Description
<code>businessTransactionID</code>	String	Optional	The unique identifier for tracing purposes across function calls.
<code>planID</code>	String	Required	The internal unique identifier for the plan to retrieve.
<code>correlationID</code>	String	Optional	The unique identifier for tracing purposes across a single function call. This is generally used by the calling application to correlate requests and responses.
<code>idsOnly</code>	Boolean	Optional	If <code>true</code> , returns the IDs of elements rather than all plan data. If <code>false</code> , returns all plan data.
<code>includeRelatedPlanItems</code>	Boolean	Optional	The value of this header is <code>true</code> or <code>false</code> , and the absence of this header denotes a <code>false</code> value. When this header is set in the

Element	Type	Cardinality	Description
			request, the output ID all the plan items, which belong to the same order line as the input plan item.
			When this header is set in the request, only a single plan item can be present in the getPlanItem request. If multiple plan items are present in the request, an error is returned.

The following table lists the details of the getPlanItemsRequest elements.

Element	Type	Cardinality	Description
businessTransactionID	String	Optional	Unique identifier for tracing purposes across function calls.
planID	String	Required	Internal unique identifier for the plan to retrieve.
idsOnly	String	Optional	If true only returns the IDs of elements rather than all plan data. Otherwise, if false returns all plan data.
planItem	Type	0-M	Plan item type.
planItem/planItemID	String	Required	Internal unique identifier for the plan item to retrieve.



Note: Any one of the orderID, orderRef, or planID is mandatory in the REST request calls.

Get Plan Items Response

The response message contains the following header properties:

Element	Type	Cardinality	Description
Success	Boolean	Required	Flag indicating if the call was successful.
messageCode	String	Required	Result message code.
Message	String	Required	Result message.
planID	String	Required	Internal unique identifier for the plan specified in the request.
Found	Boolean	Required	Flag indicating if the plan was found.

The following table lists the details of the response elements.

Element	Type	Cardinality	Description
resultStatus	Type	See Image	Result status type. See Schema References for the specification of this type.
planItem	Type	0-M	Plan item type.
planItem/planItemID	String	Required	Internal unique identifier for the plan item.
planItem/planItemName	String	Optional	Name of the process component.
planItem/udf	Type	0-M	user-defined field type.
planItem/udf/type	String	Optional	Type of the user defined field.
planItem/udf/flavour	String	Optional	Flavor of the user-defined field. The valid values are one

Element	Type	Cardinality	Description
			<p>of the following three:</p> <ul style="list-style-type: none"> • config - For user-defined field corresponding to a characteristic in the product model or a system user-defined field generated by Automated Order Plan Development. • input - For user-defined field passed in the order. • output - For user-defined field set by the process component.
planItem/udf/name	String	Required	Field name.
planItem/udf/value	String	Optional	Field value.
planItem/udf/originalValue	String	Optional	Original field value at the time of plan creation.
planItem/udf/lastModified	DateTime	Optional	Timestamp when the user-defined field was last modified.
planItem/parentID	String	0-M	IDs of the plan items, which depends on the current plan item.
planItem/childID	String	0-M	IDs of the plan items on which the current plan item depends.

Element	Type	Cardinality	Description
planItem/siblingID	String	0-M	IDs of the plan items corresponding to SIBLING_PRODUCT_* of the product fulfilled by current plan item.
planItem/dependentID	String	0-M	IDs of the plan items corresponding to DEPENDENT_PRODUCT_* of the product fulfilled by current plan item.

Get Plan Items Messages and Message Codes

The error codes and their respective error messages for the Get Plan Items interface are as follows:

Get Plan Items Messages and Message Codes

Message Code in Response Header and Result Status	Message in Response Header and Result Status	Scenario
FOM-DATA-ACCESS-SUCCESS-0000	Successfully processed GetPlanItemsRequestEvent	PlanItem or PlanItems are found and data is mapped in the response successfully.
FOM-DATA-ACCESS-PLAN-NOT-FOUND-9999	Plan not found for planID <planID value>	The plan corresponding to the planID specified in the request header is not found.
FOM-DATA-ACCESS-ORDER-NOT-FOUND-9999	Order not found for orderID <orderID value>	The order corresponding to the orderID specified in the request header is not found.
FOM-DATA-ACCESS-PLANITEM-NOT-FOUND-9999	PlanItem not found for planID <planID value> and	The planItem corresponding to the planID and planItemID specified in

Message Code in Response Header and Result Status	Message in Response Header and Result Status	Scenario
	planItemID <planItemID value>	the request header is not found.

Set Plan

The SetPlan interface can be used by the process components to add a new user-defined field or update the value of an existing user-defined field of plan or any of the containing plan items in the TIBCO Order Management system. However, it is suggested to use this interface for plan level User Defined Fields only since there is a separate interface for plan items as explained further in this guide.

If an exception occurs during SetPlan operation, then it is logged into the Data Service log. The details of the exception are returned in the response.

Set Plan Request

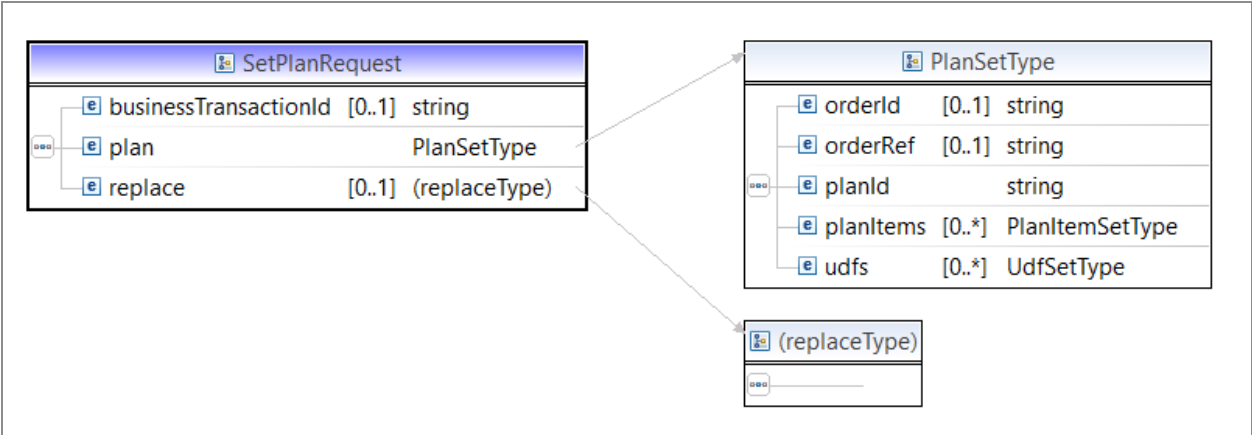
Request Endpoint: /v1/plan

The following properties must be passed in the request message header:

Element	Type	Cardinality	Description
businessTransactionID	String	Optional	Unique identifier for tracing purposes across function calls.
planID	String	Required	Internal unique identifier for the plan to retrieve.
OrderID	String	Required	Internal unique identifier for the order related to the plan to update.

Element	Type	Cardinality	Description
			Note: The value of orderId must not contain ":"
correlationID	String	Optional	Unique identifier for tracing purposes across a single function call. This is generally used by the calling application to correlate requests and responses.
replace	Boolean	Required	<p>If set to true:</p> <p>All the existing User Defined Fields is replaced by the User Defined Fields that are present in the request.</p> <p>If set to false:</p> <p>The User Defined Fields passed in the request is merged with the existing User Defined Fields.</p> <p>In any of the above case, the uniqueness of a user-defined field is maintained based on the 'name' and 'flavor' combination in the user-defined field. A user-defined field having exactly same 'name' and 'flavor' is not duplicated, if the flag EnableUniqueUDFNames is set to true in Order Management Server configurations. In case of multiple User Defined Fields with exactly same name and flavor in the request, the value from the last encountered user-defined field is considered.</p>

SetPlanRequest



The following table lists the details of the elements.

Element	Type	Cardinality	Description
businessTransactionID	String	Optional	Unique identifier for tracing purposes across function calls.
Plan	Type	Required	Plan type.
plan/planID	String	Required	Internal unique identifier for the plan to update.
plan/orderID	String	Required	Internal unique identifier for the order related to the plan to update. <div>Note: The value of orderId must not contain ":"</div>
plan/orderRef	String	Required	External unique identifier for the order related to the plan to update. <div>Note: The value of orderRef must not contain ":"</div>

plan/udf	Type	0-M	
plan/udf/type	String	Optional	Type of the user defined field.
plan/udf/flavor	String	Optional	Flavor of the user-defined field. Must be set as <i>output</i> .
plan /udf/name	String	Required	Field name.
plan/udf/value	String	Required	Field value.
plan/planItem	Type	0-M	Plan item type.
plan/planItem/planItemID	String	Required	Internal unique identifier for the plan item to update.
plan/planItem/planItemName	String	Optional	Process component name.
plan/planItem/udf	Type	0-M	user-defined field type.
plan/planItem/udf/type	String	Optional	Type of the user defined field.
plan/planItem/udf/flavor	String	Optional	Flavor of the user-defined field. Must be set as <i>output</i> .
plan/planItem/udf/name	String	Required	Field name.
plan/planItem/udf/value	String	Required	Field value.
replace	Any	Optional	If true it completely replaces the plan item, otherwise merges the user-defined field data.

Set Plan Response

The response message contains the following header properties:

Element	Type	Cardinality	Description
success	Boolean	Required	Flag indicating if the call was successful.
messageCode	String	Required	Result message code.
message	String	Required	Result message.
planID	String	Required	Internal unique identifier for the plan specified in the request.

There is nobody (payload) associated with the response message.

Set Plan Messages and Message Codes

The error codes and their respective error messages for the Set Plan interface are as follows:

Set Plan Messages and Message Codes

Message Code in Response Header and Result Status	Message in Response Header and Result Status	Scenario
FOM-DATA-ACCESS-SUCCESS-0000	Successfully processed SetPlanRequestEvent	Plan is found and user-defined field data specified in the request in plan header and plan items is updated successfully.
FOM-DATA-ACCESS-PLANITEM-NOT-FOUND-9999	PlanItem not found for planID <planID value> and planItemID <planItemID value>	PlanItem is not found against the planID and planItemID specified in the request header.
FOM-DATA-ACCESS-ORDER-NOT-FOUND-9999	Order not found for orderID <orderID value>	Order is not found against the orderID specified in the request header.
FOM-DATA-ACCESS-PLAN-NOT-FOUND-9999	Plan not found for planID <planID value>	Plan is not found against the planID specified in the request.

Set Plan Item

Overview

The SetPlanItem interface can be used by the process components to add a new user-defined field or update the value of an existing user-defined field of the plan items in a plan in the TIBCO Order Management System.

If an exception occurs during SetPlanItem operation, then it is logged into the Data Service log. The details of the exception are returned in the response.

Set Plan Item Request

Request EndPoint: /v1/planitems

The following properties must be passed in the request message header:

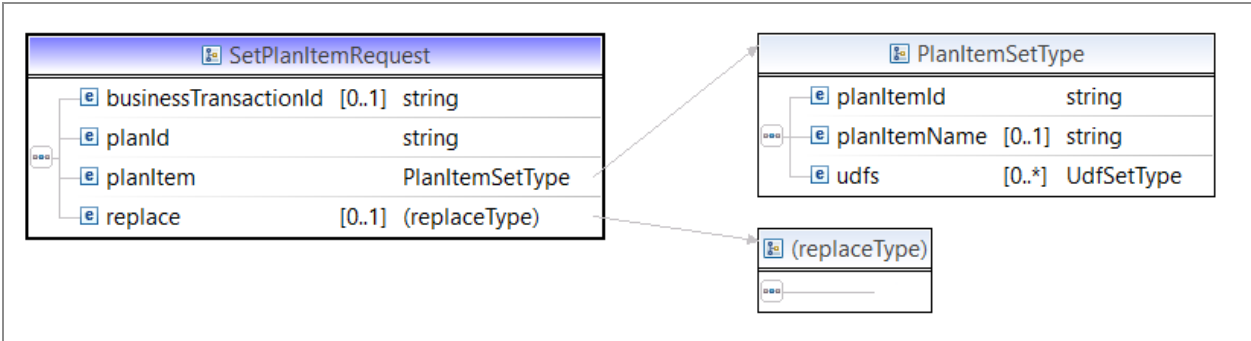
Element	Type	Cardinality	Description
businessTransactionID	String	Optional	Unique identifier for tracing purposes across function calls.
planID	String	Required	Internal unique identifier for the plan to retrieve.
correlationID	String	Optional	Unique identifier for tracing purposes across a single function call. This is generally used by the calling application to correlate requests and responses.
replace	Boolean	Required	If set to true: All the existing User Defined Fields is replaced by the User Defined Fields that are present in the request. If set to false:

The User Defined Fields passed in the request is merged with the existing User Defined Fields.

In any of the earlier mentioned cases, the uniqueness of a user-defined field is maintained based on the 'name' and 'flavor' combination in the user-defined field. A user-defined field having the same 'name' and 'flavor' do not get duplicated, if the flag EnableUniqueUDFNames is set to true in Order Management Server configurations. In case of multiple User Defined Fields with the same name and flavor in the request, the value from the last encountered user-defined field is considered.

orderId	string	Required	Internal unique identifier for the plan. <div>Note: The value of orderId must not contain ":"</div>
planItemId	String	Required	Internal unique identifier for the plan item to be updated.

SetPlanItemRequest



The following table lists the details of the elements.

Element	Type	Cardinality	Description
businessTransactionID	String	Optional	Unique identifier for tracing purposes across function calls.
planID	String	Required	Internal unique identifier for the plan to update.
planItem	Type	Required	Plan item type. Only user-defined field Name and Value are updated. If the Unique User Defined Fields are enabled for the engine an update occurs if disabled the entire current user-defined field payload is dropped and replaced with the new payload.
planItem/planItemID	String	Required	Internal unique identifier for the plan item to update.
planItem/planItemName	String	Optional	Process component name.
planItem/udf	Type	0-M	user-defined field type.
planItem/udf/type	String	Optional	Type of the user defined field.
planItem/udf/flavor	String	Optional	Flavor of the user-defined field. Must be set as output.
planItem/udf/name	String	Required	Field name.
planItem/udf/value	String	Required	Field value.
replace	Any	Optional	If true it completely replaces the plan item, otherwise merges the user-defined field data.

Set Plan Item Response

The response message contains the following header properties:

Element	Type	Cardinality	Description
success	Boolean	Required	Flag indicating if the call was successful.
messageCode	String	Required	Result message code.
message	String	Required	Result message.
planID	String	Required	Internal unique identifier for the plan specified in the request.

There is nobody (payload) associated with the response message.

Set Plan Item Messages and Message Codes

The error codes and their respective error messages for the Set Plan Items interface are as follows:

Set Plan Items Messages and Message Codes

Message Code in Response Header and Result Status	Message in Response Header and Result Status	Scenario
FOM-DATA-ACCESS-SUCCESS-0000	Successfully processed SetPlanItemRequestEvent	PlanItem is found and user-defined field data specified in the request is updated successfully.
FOM-DATA-ACCESS-PLAN-NOT-FOUND-9999	Plan not found for planID <planID value>	Plan is not found against the planID specified in the request header.
FOM-DATA-ACCESS-ORDER-NOT-FOUND-9999	Order not found for orderID <orderID value>	Order is not found against the orderID specified in the request

Message Code in Response Header and Result Status	Message in Response Header and Result Status	Scenario
		header.
FOM-DATA-ACCESS-PLANITEM-NOT-FOUND-9999	PlanItem not found for planID <planID value> and planItemID <planItemID value>	PlanItem is not found against the planID and planItemID specified in the request header.

Best Practices for TIBCO Order Management

This section covers the best practices that can serve as guidelines for building a fulfillment solution by using TIBCO Order Management.

Exception Handling Guidelines

Exception Handling Guidelines provides information about guidelines that can be followed for handling the exceptional conditions in process components.

General Approach

TIBCO Order Management does not include any out-of-the-box approach for error handling. The product architecture does account for exception handlers, and provides the necessary hooks, where it can be integrated with an existing exception or fallout management system, or to which a custom solution can be connected.

The product capabilities for supporting error handling are fully described in the product documentation, and it is assumed that the reader is familiar with that document. The basics are not covered here.

Plan Item Failed Handler or no Plan Item Failed Handler

A key question is whether to handle exceptions within the process component itself, or whether to manage exception handling through the orchestrator and the Plan Item Failed (PIF) handler. In the first case, process components must only return a success result to the orchestrator, and no Plan Item Failed handler is required.

In the second case, it is necessary to develop a Plan Item Failed handler that receives `PlanItemFailedRequest` from the orchestrator for direction on how to proceed once an error is encountered. The Plan Item Failed handler must respond to the orchestrator telling it whether to retry the plan item, or Continue (that is, mark the plan item as completed and continue with the plan).

A consideration here is the type of process component. If a process component makes use of a workflow engine for its implementation, which already includes manual tasks and GUI interaction, it makes sense for errors in the flow to be managed within the workflow engine, rather than have them passed back to the orchestrator. If the process component is BW or BE, the Plan Item Failed handler might be a better option.

Functional Exception against Technical Exception

Any consideration of exceptions handling must consider the different types of exception that can occur, which typically fall into two broad categories, functional exceptions, and technical exceptions. For the purposes of this discussion, we define these as follows:

- A *Functional Exception* occurs when a back-end system returns an error code to the process component, indicating a problem with the request. Functional exceptions always occur in the context of a process component. An example can be a request to allocate a phone number that is already in use. Receiving a functional exception is expected to occur under normal circumstances, and the system is built to handle these events.
- A *Technical Exception* occurs when something goes wrong, so that the system is not designed to handle under normal circumstances. They can occur in process components and also TIBCO Order Management components such as orchestrator and Order Management Server. They are typically caused by conditions in the external environment, such as running out of memory, failure in Enterprise Message Service, and hardware failure, but can also be caused by defects.

Different strategies might be considered for each of these types. For instance, as functional exceptions occur within the context of a process component, and typically require an operator to review and decide on remedial action, it makes sense for these to be handled through the orchestrator and a Plan Item Failed handler, which might defer to an external GUI for a resolution.

Technical exception handling is more difficult, as they can be caused by almost anything. In this case, even if a Technical exception occurs in a plan item, it might make more sense to simply log it and stop execution of the plan item.

Example Approach

This topic describes an approach for implementing exception handling, where order fallout is managed externally to the TIBCO Order Management implementation. This is a good approach where the customer site has an existing order fallout management system,

providing GUI Windows, and so on. whose functionality can be applied. In the rest of this topic we refer to this external error handling system as Exception Management, or EM. Please note this is an example only, and might not be applicable for your particular circumstance.

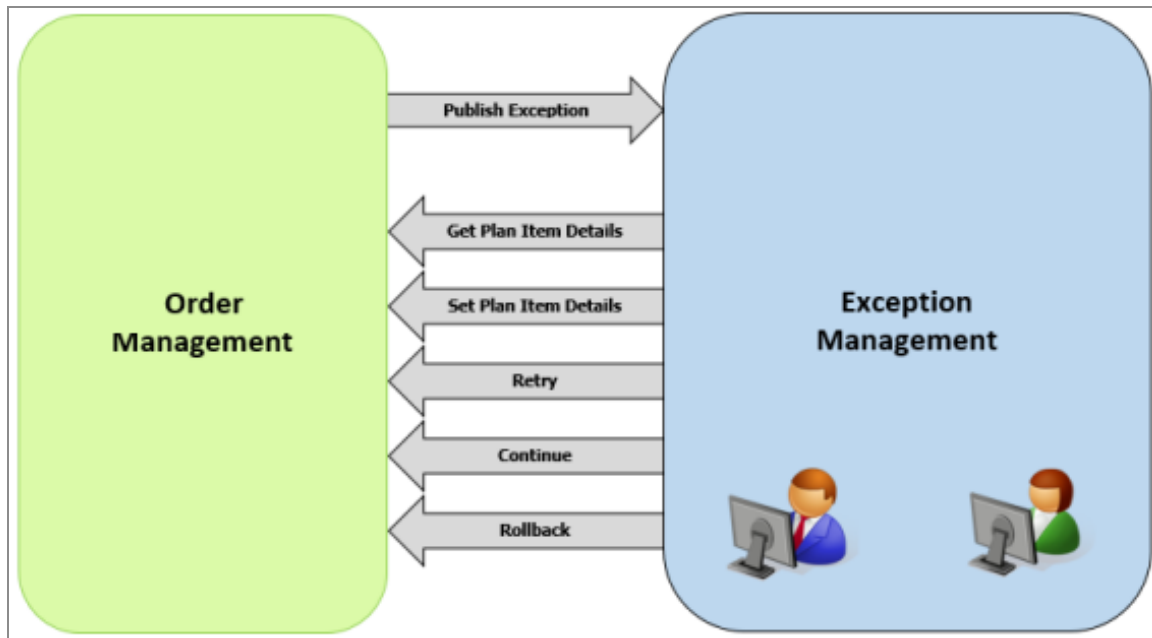
In this case, Functional exceptions are managed via the Plan Item Failed handler, and Technical exceptions via a separate mechanism.

For Functional exceptions the requirements are to forward all to Exception Management, for an operator to analyze. The possible resolutions are:

- Retry the Plan Item step, with the possibility to edit input values for the downstream call that failed. Note this is different to retrying the plan item from the beginning. Some process components can internally be orchestrating multiple steps.
- Continue the Plan Item, with the possibility to edit output values from the downstream call that failed (note this might not be quite the same as the Complete Plan Item Failed handler response, which instructs the Orchestrator to complete the plan item. There might be the activity that we require the process component to complete after the downstream call but before it completes).
- Roll back the entire order, performing compensating actions if required.

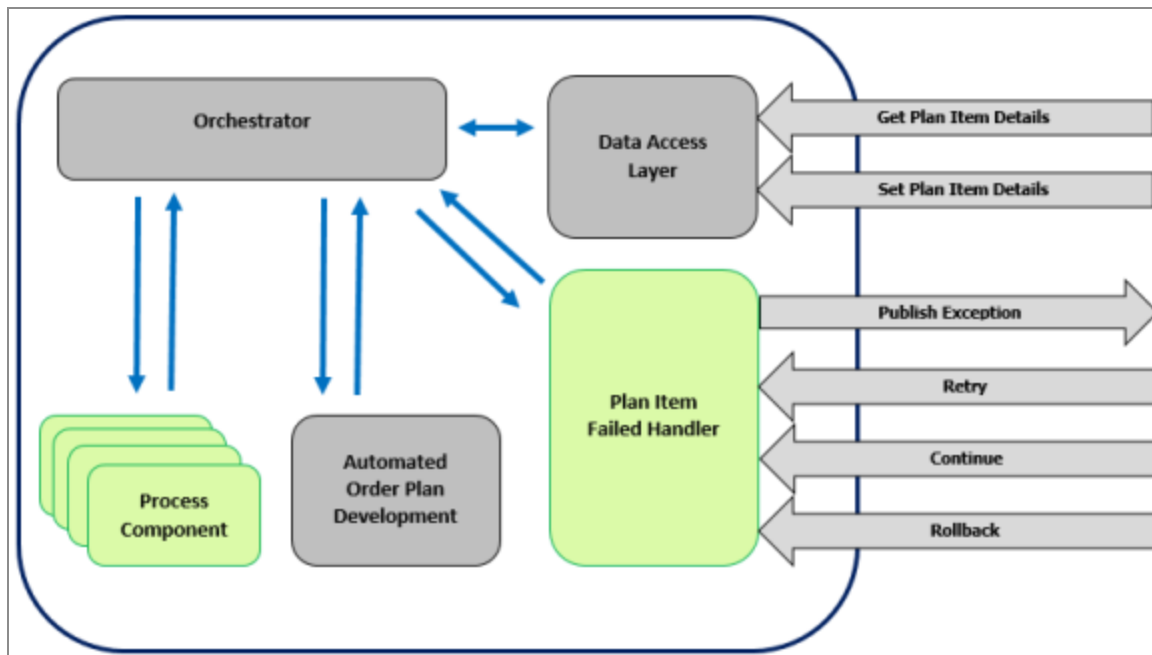
The architectural approach here is to define a clear separation of concerns, between what TIBCO Order Management does, and what Exception Management does. The following diagram shows the approach in the case of Functional exceptions. Also, the data access GetPlanItem and SetPlanItem calls are used to support the functionality of editing input/output values.

Example Functional Exception Handling Overview Architecture



The following image shows an approach for how this can be implemented within the application:

Example Functional Exception Handling TIBCO Order Management Components



Plan Item Failed Handler

In this example, the Plan Item Failed (PIF) Handler is built as a pass-through component. It performs the following:

- On receipt of a `PlanItemFailedRequest` message from the Orchestrator, publishes a message (to Exception Management).
- On receipt of a “Retry” or “Continue” resolution type from Exception Management, creates a `PlanItemFailedResponse` message and sends to the Orchestrator with an appropriate flag that is either retry, resume, or complete.
- On receipt of a “Rollback” resolution type from Exception Management, send a message to Order Management Server to cancel the entire order. No `PlanItemFailedResponse` message is sent to the Orchestrator in this case.

Process Component Considerations

When mapping the selected resolution type to a `PlanItemFailedResponse` to send to the Orchestrator, there are some considerations regarding this and the nature of the process component implementation, that is, whether it runs multiple steps, or is atomic.

For process components that implement multiple steps (example: a BE process component):

- A retry action means that the entire process component is re-run. If what is required is just the current step to be retried, a Resume action must be specified, not retry.
- A complete action means that the process component is not invoked again in any way, and the plan item is marked as complete.
- A distinction needs to be made between a resume where the current step needs to be retried, or skipped. There is no way to do this on the `PlanItemFailedResponse` message, so this needs to be managed another way, example: by setting a user-defined field on the plan item to indicate, which is required.

Pre-Qualification Failed Handler

Like the Plan Item Failed handler, there is no default implementation of the Pre-Qualification Failed handler provided with the product.

Be aware that the pre-qualification failed handler deals with errors raised not only in Feasibility, but also, in Automated Order Plan Development. Even if in your architecture you don't have a Feasibility step, you have Automated Order Plan Development, and if Automated Order Plan Development raises exceptions, the orchestrator publishes an event to the Pre-Qualification Failed handler and waits for a response. If there is no Pre-Qualification Failed handler implemented, nothing further happens for that order and it is "stuck".

Even if Automated Order Plan Development exceptions are expected to be rare for your application (i.e. you validate the order thoroughly prior to Automated Order Plan Development), consider at the very least, implementing monitoring on the Pre-Qualification Failed request queue, so that operations is aware that Automated Order Plan Development has failed for an order, and some action needs to be taken to move the order on.

You might want to consider making the Pre-Qualification Failed handler "just another" source of Technical Exceptions. In this way, a framework for dealing with automating Technical Exception handling, can be used to also deal with Pre-Qualification Failed requests. This is the approach that is described in the next section.

Technical Exception Handling

For technical exception handling, it is difficult to define a pattern that can always apply, given the diverse range of possible exceptions that can be raised. Such exceptions can be raised from anywhere – TIBCO Order Management components, process components, and any other code that might be developed as part of a total fulfillment solution.

It is of course always good general software engineering practice to build components as resilient as possible to errors. It might make sense, depending on the context, to build in mechanisms such as retry, when events such as timeouts occur. Of course, this needs to be weighed up against the additional complexity this introduces into the solution, and needs to consider the idempotency of interactions. Complex, built-in "self-healing" type mechanisms themselves increase the risk of coding defects, increase the complexity of testing, and cannot catch all types of errors.

The underlying principle here is that, as with functional exceptions, technical exceptions require manual inspection to determine what to do. The default approach is that all technical exceptions are also dealt with manually. This can mean messages being manually copied from one queue to another, database entries being manually edited, and so on.

Nevertheless, it is highly desirable, in some common circumstances, for a fulfillment system that can resolve technical exceptions in an automated way. No system can be built

to automatically resolve all exceptions, however one approach is to build a mechanism that can support the incremental addition of automated technical exception resolutions, as the system evolves and experience is gained into the types of exceptions that can occur, and how best to deal with them. This section outlines a possible approach to building such a mechanism.

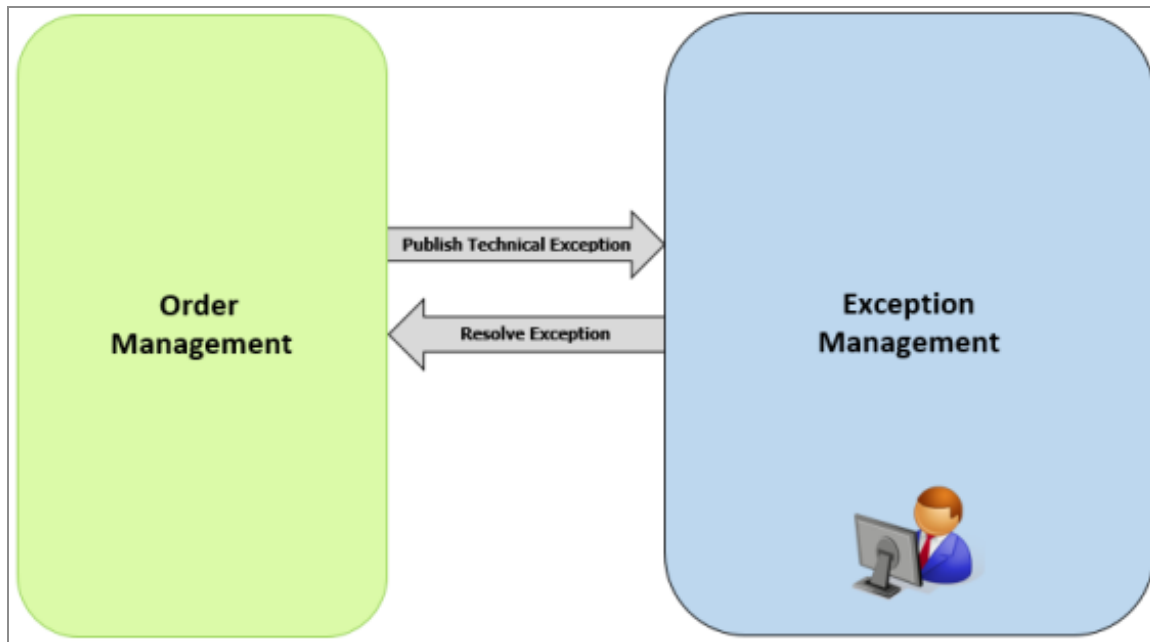
As with the handling of functional exceptions, it is important to define a clear architectural separation between the fulfillment system and the system that determines what to do with exceptions. Again, we term this body Exception Management, see [Technical Exception Handling Architecture Overview](#).

To simplify the interface, we define here a single “Resolve Exception” service, which is used to resolve all technical exceptions. It expects an argument “Resolution Type”, which is a string that defines what specific resolution behavior is required.

It is good practice when building custom components of a fulfillment solution, such as the process components, and database adapters, and enrichment processes, to ensure that technical exceptions are caught and logged in a consistent way. We define a single “Publish Technical Exception” service for TIBCO Order Management to use when publishing a technical exception. This same service is invoked regardless of the source of the technical exception, which can be custom code, TIBCO Order Management internal components, or even a Pre-Qualified Failed error.

When publishing an exception to Enterprise Message Service, TIBCO Order Management needs to publish along with the exception, all the information that it has to handle the resolution.

Technical Exception Handling Architecture Overview



Types of Technical Exception

We identify the following types of technical exceptions as candidates for automation via this approach. These are of course not the only types of Technical exception that can occur.

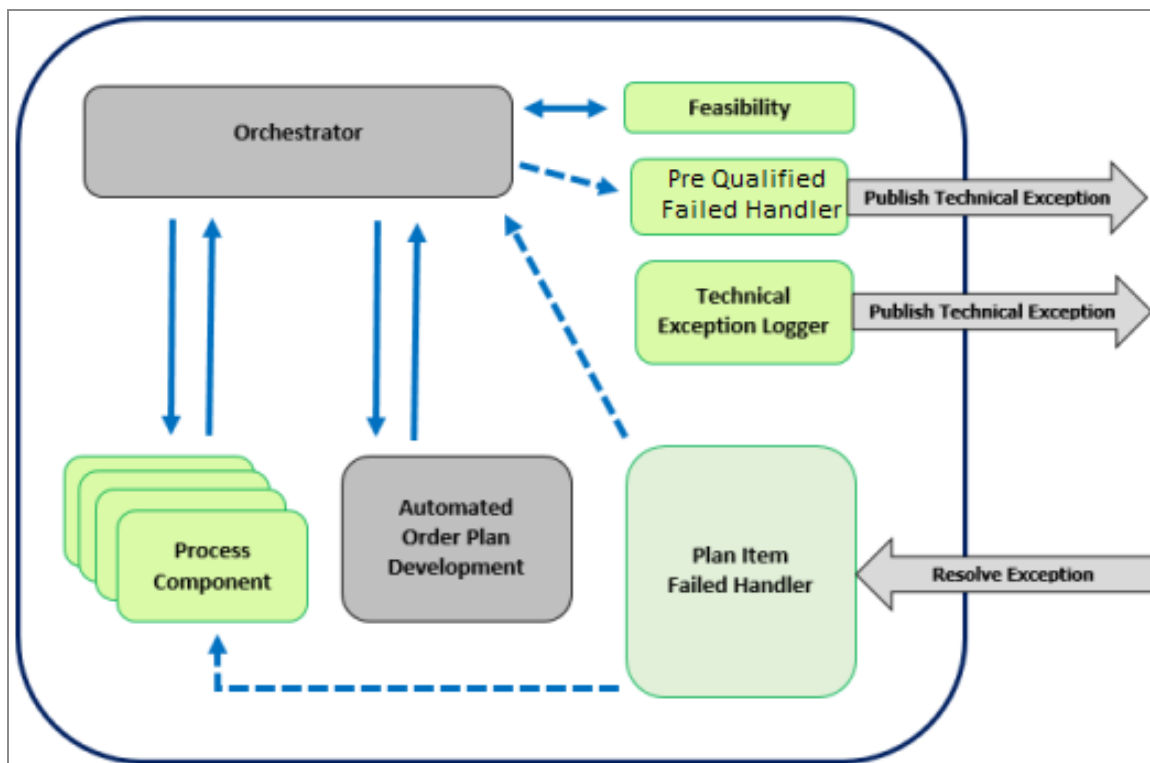
1. Pre order submit (i.e. an error that happens during any order pre-processing or enrichment)
 - a. Resubmit order only action possible – but first state needs to be cleaned from any database tables etc.
 - b. Relatively easy to automate.
2. Pre-qualification Failed Handler
 - a. If an error occurs in plan development (or Feasibility, if present).
3. Process Component
 - a. Most technical exceptions likely to be this type.
 - b. The Process Component can potentially be restarted (retried), continued or completed, depending on how far it has progressed.

TIBCO Order Management Components for Technical Exception Handling

The [Components involved in Technical Exception Handling](#) outlines the components within TIBCO Order Management that is involved in handling technical exceptions. Note that the other components not directly involved in the solution for technical exception handling are not shown.

It must be noted however that any component within the TIBCO Order Management can generate a technical exception. This includes those shown below, and the core components, such as Orchestrator, data access interfaces, and Automated Order Plan Development.

Components involved in Technical Exception Handling



The following outlines the required behavior of the components that have to be built, in the context of Technical Exception Handling:

Process Component

Technical exceptions occurring during the execution of process components logs a technical exception directly to Exception Management, via the Technical Exception Logger,

and stop execution. Orchestrator is not notified when a Technical exception occurs, and considers the Process Component to be in “Execution” state. The process component can be retried or continued by the Technical Exception handler, or the Technical Exception handler can notify Orchestrator directly that the Process Component is complete.

Feasibility

The Feasibility step is invoked by Orchestrator after it has received and stored the order, but before it invokes Automated Order Plan Development to get the plan. Like Automated Order Plan Development, the feasibility component can return an error to the Orchestrator, if Feasibility fails. Also like Automated Order Plan Development, in the context of the example, a Feasibility error is regarded as a Technical exception, as Feasibility must always pass under normal circumstances (this might not typically be true though, for instance if order validation is performed at Feasibility).

Technical Exception Logger

This component publishes Technical exceptions to Exception Management. It must capture all Technical exceptions generated from custom components, and publish them in a standard way to Exception Management. A standard set of exception fields must be published, which must include order ids (if available), the component and service that generated the error, and an error code. The message being processed at that time might also be logged. The key requirement is that there must be enough information logged to enable Exception Management to choose a resolution type to be applied to resolve the exception, and enough information to be passed back to the Technical Exception Handler for it can resolve the exception.

Pre-Qualification Failed (PQF) Handler

Its role is to receive notifications that Orchestrator publishes when it receives an error from Feasibility or Automated Order Plan Development, and publish them to Exception Management.

Technical Exception Handler

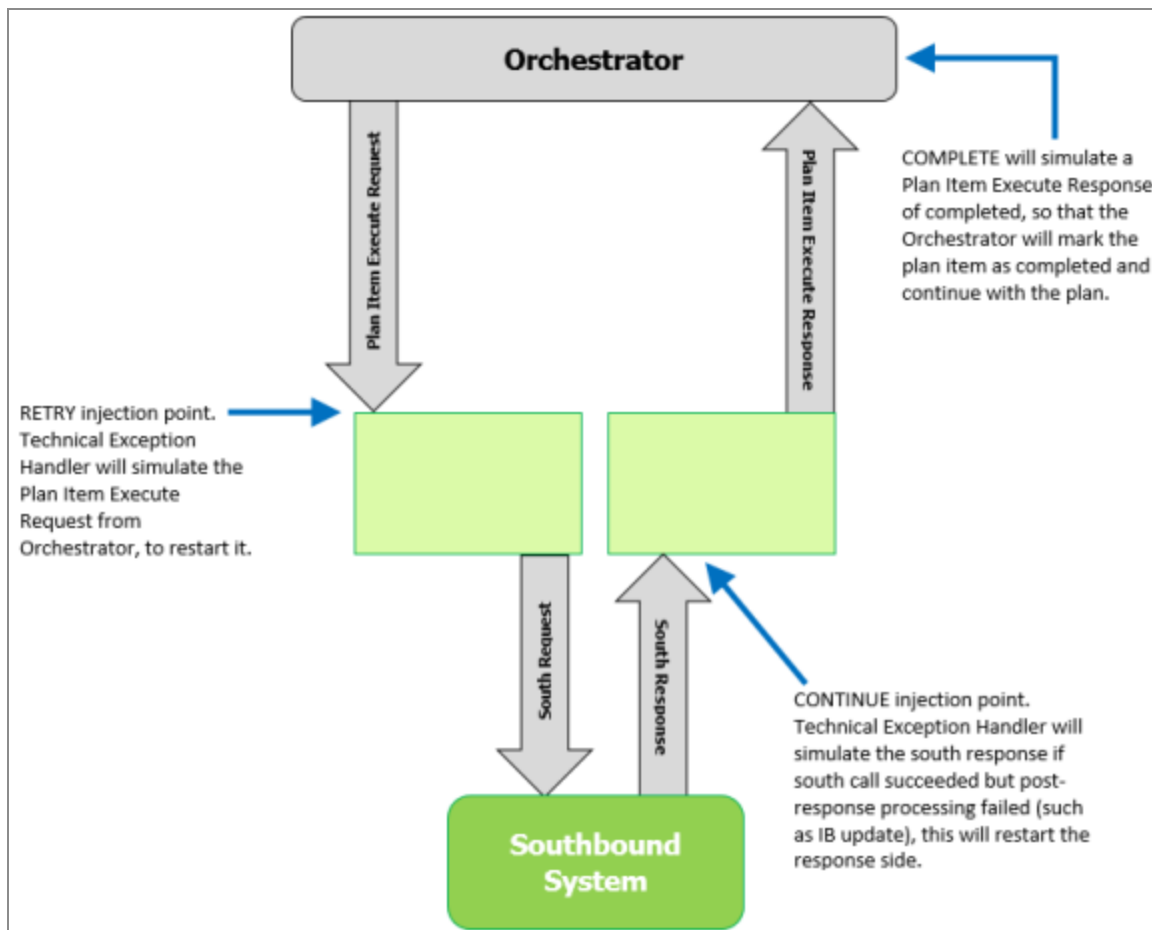
Its role is to expose the service for resolving Technical Exceptions for Exception Management to call, and to implement the logic for performing the resolution. This might involve sending messages to a process component, or to perform some custom action (such as clean up a database table and resubmit an order). It might also communicate

directly with Orchestrator, for instance to send a Pre-Qualification Failed Response message.

The number of resolution types this component can expose, might grow over time as new resolutions are added.

The [Process Component Technical Exception Services Overview](#) outlines at a high level, how the Retry, Continue and Complete services can potentially be implemented.

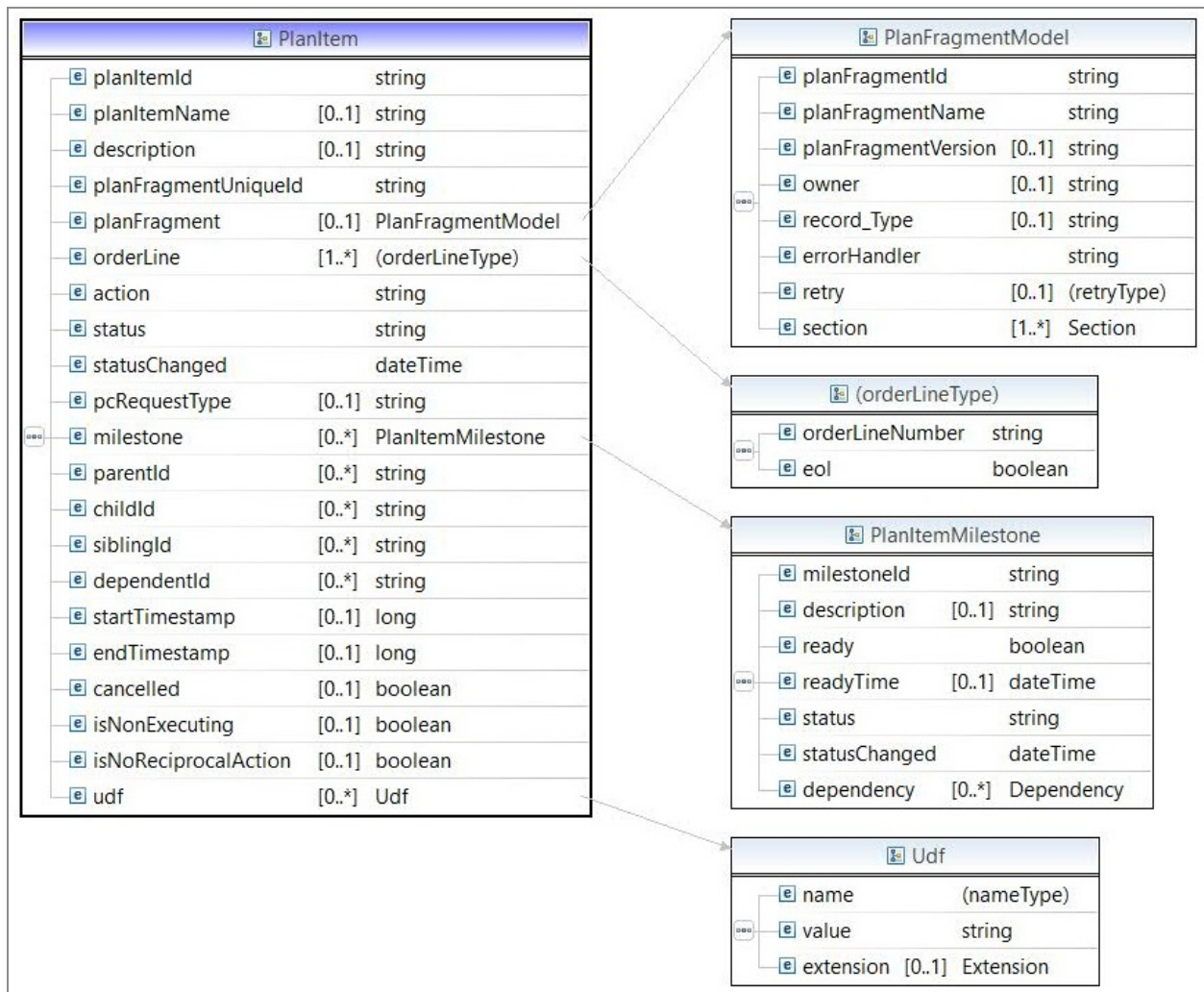
Process Component Technical Exception Services Overview



Schema References

Plan Item

Plan Item



Element

Type

Cardinality

Description

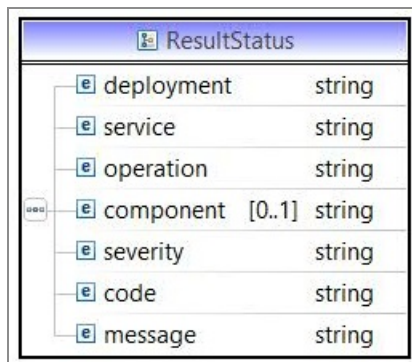
planItem/planItemID	String	Required	A unique identifier for the plan item within the plan to be executed.
planItem/description	String	Optional	Description for the plan item to be executed.
planItem/processComponentID	String	Required	A unique identifier for the process component to be executed.
planItem/processComponentName	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.
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.
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.

ResultStatus

Result Status



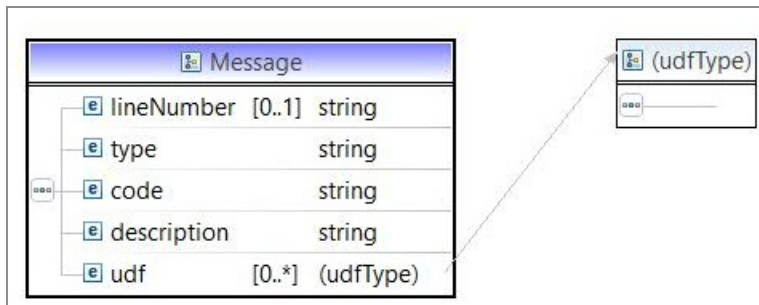
Element	Type	Cardinality	Description
deployment	String	Required	Engine deployment that returned this result.
service	String	Required	Service name that returned this result
operation	String	Required	Operation within the service that returned this result.
component	String	Optional	Component within the operation and service

that returned this result.

severity	String	Required	Severity result. Valid values are: <ul style="list-style-type: none"> • S - Success • W - Warning • E - Error
code	String	Required	Message code for this result.
message	String	Required	Message details for this result.

Message

Message

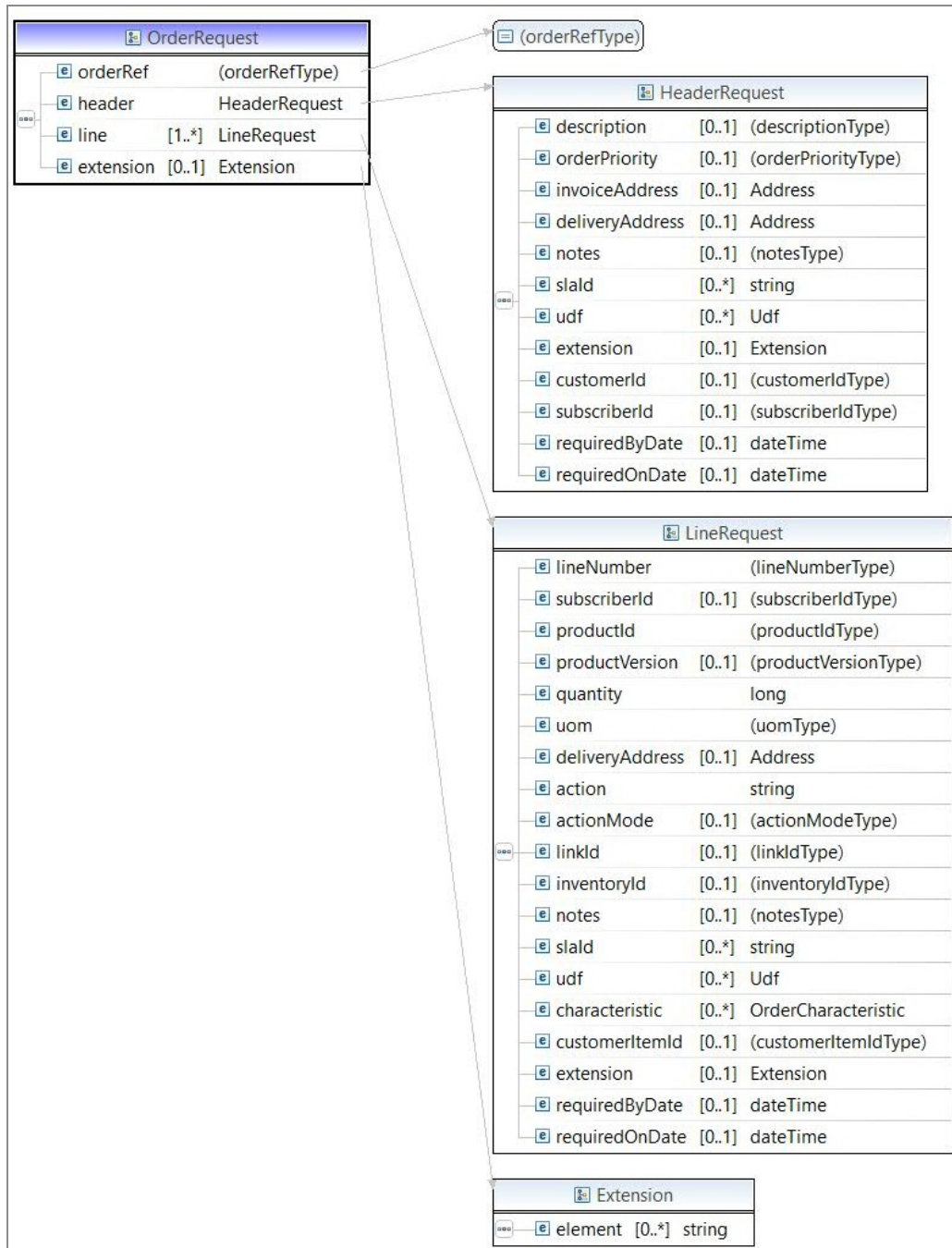


Element	Type	Cardinality	Description
lineNumber	String	Optional	Order line number that this message refers to.
type	String	Required	Message type. Valid values are: <ol style="list-style-type: none"> 1. Information 2. Warning 3. Error
code	String	Required	Message code for this message.

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. Note: The value of orderRef must not contain ":"
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

Samples

Sample Order XML

The sample order XML is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<Order Id="544">
  <orderID>81</orderID>
  <sessionID>CORRELATION-3baee8b0-b483-47aa-89b2-
bf7b03d0c41f</sessionID>
  <orderlines Id="545">
    <lineID>1</lineID>
    <productID>CFS TV</productID>
    <action>PROVIDE</action>
    <quantity>1.0</quantity>
    <requiredByDate>2011-04-30T23:50:00+05:30</requiredByDate>
    <LineUsed>>false</LineUsed>
    <OrderlinesUDF Id="546">
      <name>OrderRef</name>
      <value>OrderRefID</value>
      <flavor>input</flavor>
    </OrderlinesUDF>
  </orderlines>
  <orderlines Id="547">
    <lineID>2</lineID>
    <productID>CFS Live Box</productID>
    <action>PROVIDE</action>
    <quantity>1.0</quantity>
    <requiredByDate>2011-04-30T23:50:00+05:30</requiredByDate>
    <LineUsed>>false</LineUsed>
    <OrderlinesUDF Id="548">
      <name>OrderRef</name>
      <value>OrderRefID</value>
      <flavor>input</flavor>
    </OrderlinesUDF>
  </orderlines>
  <orderlines Id="549">
    <lineID>3</lineID>
```

```

    <productID>CFS_VOIP</productID>
    <action>PROVIDE</action>
    <quantity>1.0</quantity>
    <requiredByDate>2011-04-30T23:50:00+05:30</requiredByDate>
    <LineUsed>>false</LineUsed>
    <OrderlinesUDF Id="550">
      <name>OrderRef</name>
      <value>OrderRefID</value>
      <flavor>input</flavor>
    </OrderlinesUDF>
  </orderlines>
  <status>NewOrder</status>
  <currentTime>2012-07-18T10:19:03+05:30</currentTime>
  <TineDelay>0</TineDelay>
  <customerref>Apple</customerref>
  <OrderHeaderUDF Id="551">
    <name>Company</name>
    <value>Orange</value>
    <flavor>input</flavor>
  </OrderHeaderUDF>
  <Originator>Orchestrator</Originator>
  <OrderRef>OrderRefID</OrderRef>

  <businessTransactionID>a7eb1e1de1fa45c993f65589dba70648</businessTransac
tionID>
</Order>

```

Sample Plan Item XML

The sample plan item is as follows:

```

<?xml version="1.0" encoding="UTF-8"?>
<PlanItem Id="2169">
  <planID>PF1</planID>
  <parentID>CORRELATION-1b1260e6-9cdd-4903-a184-
d473aa11b622</parentID>
  <lineID>2</lineID>
  <dependentOn>PF10.7747556</dependentOn>
  <planDesc> PROVIDE</planDesc>
  <planItemID>PF10.8786092</planItemID>
  <EOL>N</EOL>
  <TimeDelay>0</TimeDelay>
  <status>addDependency</status>

```

```

<singleUse>>false</singleUse>
<sequence>0</sequence>
<sequenceName>leaf</sequenceName>
<action>PROVIDE</action>
<productID>GSMDDataService</productID>
<itemMark4Del>>false</itemMark4Del>
<mustComplete>>true</mustComplete>
<affinityType>ConditionalAffinity</affinityType>
<affintyPlanID>PF1</affintyPlanID>
<affintyPlanDesc> AFFINITY PROVIDE</affintyPlanDesc>
<udfs Id="2170">
  <name>TASKID</name>
  <value>PF10.8786092</value>
  <flavor>config</flavor>
</udfs>
<udfs Id="2172">
  <name>PRODUCTID</name>
  <value>GSMDDataService</value>
  <flavor>config</flavor>
</udfs>
<udfs Id="2173">
  <name>RECORD_TYPE</name>
  <value>SERVICE</value>
  <flavor>config</flavor>
</udfs>
<udfs Id="2174">
  <name>MSISDN</name>
  <value>123</value>
  <flavor>input</flavor>
</udfs>
<Ancestors>PF10.7747556</Ancestors>
<cancelUsed>>false</cancelUsed>
<M_EP_UDFS Id="2171">
  <name>M_EP_UDFS</name>
  <value>PF10.8786092</value>
</M_EP_UDFS>
<pI_Used>>false</pI_Used>
<isLeaf>>false</isLeaf>
<counter>0</counter>
<LinkID>1</LinkID>
<affinityCorrelation>$var/PlanItem[productID='GSMLine']/
udfs[name='MSISDN']/value/text()</affinityCorrelation>
<affinityParentGroup>>false</affinityParentGroup>
<affinityActionGroup>>false</affinityActionGroup>
<isDynamic>>false</isDynamic>
</PlanItem>

```


Sample XPATHs

```
<ns0:affinityCondition>exists($var/Order/OrderHeaderUDF  
[name="SubscriberProduct"and value="Product BB Network Access"])  
</ns0:affinityCondition>
```

```
<ns0:affinityCorrelation>exists($var/Order/OrderHeaderUDF[name=  
"SubscriberProduct"and value="Product BB Network  
Access"])</ns0:affinityCorrelation>
```

```
<ns0:affinityActionValue>$var/Order/orderlines[productID='CFS  
STB']/action/text()</ns0:affinityActionValue>
```

```
<affinityCorrelation>$var/PlanItem[productID='GSMLine']/udfs  
[name='MSISDN']/value/  
text()</affinityCorrelation>
```

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.