TIBCO® Fulfillment Order Management Concepts and Architecture
Software Release 3.0.0
July 2015
SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

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

This document contains confidential information that 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 TIBCO Software Inc.

TIBCO, Two-Second Advantage, TIBCO ActiveMatrix BusinessWorks, TIBCO Runtime Agent, TIBCO Administrator, and TIBCO Enterprise Message Service, are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

EJB, Java EE, J2EE, and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and 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.

THIS 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 THIS SOFTWARE VERSION 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. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

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 “READ ME” FILES.

Copyright © 2010-2015 TIBCO Software Inc. ALL RIGHTS RESERVED.

TIBCO Software Inc. Confidential Information.
# Contents

**Preface**.........................................................................................................................7
  - Related Documentation..............................................................................................8
  - Typographical Conventions......................................................................................9
  - Connecting with TIBCO Resources..........................................................................10

**Chapter 1  Introduction**..............................................................................................11
  - About TIBCO® Fulfillment Order Management........................................................12
  - TIBCO® Fulfillment Orchestration Suite Overview..................................................13
    - TIBCO Fulfillment Orchestration Components.......................................................14
    - User Interface Integration.......................................................................................14
  - TIBCO® Fulfillment Order Management Overview................................................17

**Chapter 2  Concepts**....................................................................................................19
  - Order..........................................................................................................................20
  - Characteristics............................................................................................................22
  - Product........................................................................................................................23
  - Plan.............................................................................................................................24
    - Plan Item..................................................................................................................24
    - Milestone................................................................................................................24
    - Dependency............................................................................................................24
  - Plan Fragment..........................................................................................................27
    - Error Handling.......................................................................................................27
    - SLA Notification....................................................................................................28
  - Plan Development.....................................................................................................30
  - Lifecycle....................................................................................................................31
    - Order.......................................................................................................................31
    - Order Line...............................................................................................................32
    - Order Amendment..................................................................................................33
    - Plan..........................................................................................................................34
    - Plan Item..................................................................................................................35
    - Milestone................................................................................................................36
    - Dependency............................................................................................................36
  - Sequences..................................................................................................................38
    - Standard Order......................................................................................................38
    - Amend Order Fulfillment.........................................................................................48
    - Cancel Order..........................................................................................................53
# Chapter 3  Architecture

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>56</td>
</tr>
<tr>
<td>Order Management Server</td>
<td>58</td>
</tr>
<tr>
<td>Orchestrator</td>
<td>59</td>
</tr>
<tr>
<td>Automated Order Plan Development</td>
<td>62</td>
</tr>
<tr>
<td>Manual Order Plan Development</td>
<td>65</td>
</tr>
<tr>
<td>Order Capture System</td>
<td>66</td>
</tr>
<tr>
<td>Offer and Price Engine</td>
<td>69</td>
</tr>
<tr>
<td>Process Components</td>
<td>70</td>
</tr>
<tr>
<td>Feasibility Provider</td>
<td>71</td>
</tr>
<tr>
<td>Jeopardy Management</td>
<td>72</td>
</tr>
<tr>
<td>Router</td>
<td>76</td>
</tr>
<tr>
<td>Key Functionality</td>
<td>77</td>
</tr>
</tbody>
</table>
Preface

The preface contains information about documentation related to the current document, typographical conventions, and information on how to contact TIBCO support.
Related Documentation

This section lists documentation resources you may find useful.

- **TIBCO Fulfillment Order Management Concepts and Architecture** This manual describes terminology and concepts of TIBCO® Fulfillment Order Management.
- **TIBCO Fulfillment Order Management Installation and Configuration** Read this manual for instructions on installation, and configuration.
- **TIBCO Fulfillment Order Management Administration** Read this manual for instructions on administration tasks.
- **TIBCO Fulfillment Order Management User’s Guide** This manual describes the features as well as all the screens.
- **TIBCO Fulfillment Order Management Web Services** Read this manual for information about the web services.
- **TIBCO Fulfillment Order Management Release Notes** Read the release notes for a list of features. This document also contains the list of known issues for this release.
Typographical Conventions

The following typographical conventions are used in this manual:

Table 1: General Typographical Conventions

<table>
<thead>
<tr>
<th>Convention</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIBCO_HOME</td>
<td>Many TIBCO products are installed within the same home directory. This directory is referenced in documentation as TIBCO_HOME. The value of TIBCO_HOME depends on the operating system. For example, on Unix systems the default value is $HOME/tibco.</td>
</tr>
<tr>
<td>TRA_HOME</td>
<td>TIBCO® Runtime Agent installs into a directory inside ENV_HOME. This directory is referenced in documentation as TRA_HOME. The value of TRA_HOME depends on the operating system. For example, on Unix systems the default value is $TIBCO_HOME/tra.</td>
</tr>
<tr>
<td>AF_HOME</td>
<td>TIBCO® Fulfillment Order Management installs into a directory inside ENV_HOME. This directory is referenced in documentation as AF_HOME. The value of AF_HOME depends on the operating system. For example, on Unix systems the default value is $TIBCO_HOME/af/3.0.</td>
</tr>
</tbody>
</table>

*Code font* identifies commands, code examples, filenames, pathnames, and output displayed in a command window. For example:

Use *MyCommand* to start the foo process.

*Bold code font* is used in the following ways:

- In procedures, to indicate what a user types. For example: Type *admin*.
- In large code samples, to indicate the parts of the sample that are of particular interest.
- In command syntax, to indicate the default parameter for a command. For example, if no parameter is specified, MyCommand is enabled:

```
MyCommand [enable | disable]
```

*Italic font* is used in the following ways:

- To indicate a document title. For example: See TIBCO® BusinessWorks Concepts.
- To introduce new terms. For example: A portal page may contain several portlets. Portlets are mini-applications that run in a portal.
- To indicate a variable in a command or code syntax that you must replace. For example: *MyCommand pathname*

The note icon indicates information that is of special interest or importance, for example, an additional action required only in certain circumstances.

The tip icon indicates an idea that could be useful, for example, a way to apply the information provided in the current section to achieve a specific result.

The warning icon indicates the potential for a damaging situation, for example, data loss or corruption if certain steps are taken or not taken.
Connecting with TIBCO Resources

How to Join TIBCOMmunity

TIBCOMmunity is an online destination for TIBCO customers, partners, and resident experts—a place to share and access the collective experience of the TIBCO community. TIBCOMmunity offers forums, blogs, and access to a variety of resources. To register, go to http://www.tibcommunity.com.

How to Access All TIBCO Documentation

After you join TIBCOMmunity, you can access the documentation for all supported product versions here: https://docs.tibco.com.

How to Contact TIBCO Support

For comments or problems with this manual or the software it addresses, please contact TIBCO Support as follows:

• For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:
  http://www.tibco.com/services/support
• If you already have a valid maintenance or support contract, visit this site:
  https://support.tibco.com

  Entry to this site requires a username and password. If you do not have a username, you can request one.
Chapter

1

Introduction

This chapter gives an overview of TIBCO® Fulfillment Orchestration Suite and its internal component TIBCO® Fulfillment Order Management and its infrastructure software. This includes information regarding all of the TIBCO Fulfillment Orchestration Suite components, user interface integration for the suite components, and TIBCO Fulfillment Order Management components.

Topics

• About TIBCO® Fulfillment Order Management
• TIBCO Fulfillment Orchestration Suite Overview
• TIBCO® Fulfillment Order Management Overview
About TIBCO® Fulfillment Order Management

TIBCO® Fulfillment Order Management (FOM) is a meta data driven order management and fulfillment system. FOM is a component of the TIBCO® Fulfillment Orchestration Suite (FOS).
New technologies and network architectures enable communications service providers (CSP) to create innovative converged products and services offerings which are introduced faster and have shorter life cycles than previous service offerings to address a very changing and competitive market.

In view of the rapid pace of change in the technology, the industry is evolving to become a contributor and not remain a mere consumer of technology. In this environment, communications service providers face the challenge of defining, managing, and delivering numerous complex products and variations to the market in a most effective way to differentiate themselves. TIBCO has concentrated and structured its services around the following points:

• New product offerings are designed and rolled out in a few weeks including implementation in all the fulfillment chain.
• Customer orders are instantly fulfilled and provisioned in the network to maximize customer experience.
• Customer orders come from a large variety of order entries such as customer self-care portals, customer sales representative desks or even network elements detecting service access to leverage fulfillment chain investment and support hardware rationalization.

TIBCO provides CSPs with a comprehensive and integrated solution ready for complete end-to-end fulfillment automation. The TIBCO® Fulfillment Orchestration Suite defines new product and service offerings, associated fulfillment rules and processes, and automates the delivery from order capture down to the service activation in the network.

![Fulfillment Orchestration Architecture](image)

**Figure 1: Fulfillment Orchestration Architecture**
TIBCO Fulfillment Orchestration Components

The TIBCO® Fulfillment Orchestration Suite solution is capable of supporting end-to-end order fulfillment with Order Management, Provisioning, and Catalog capabilities.

The following are the constituents of the TIBCO® Fulfillment Orchestration Suite:

- **TIBCO® Fulfillment Catalog**: A Catalog function that defines and manages life cycles of commercial and technical offerings.

- **TIBCO® Fulfillment Order Management**: TIBCO® Fulfillment Order Management is a meta data driven order management and fulfillment system which allows development of fulfillment plans based on meta-data specified in product catalogs. Order fulfillment and service provisioning is no longer a simple single-service or product workflow. The dynamic bundled offerings along with the explosion of devices, applications, real-time inventory management and third-party content providers require a complex order fulfillment system which can adapt to the changes in processes, meta data, and inventory. Traditional OSS/BSS approach with the data silos fail to provide a dynamic and agile solution.

- **Fulfillment Provisioning**: A Provisioning function that automates the activation of the underlying network services and allocation of all the network resources. The Fulfillment Orchestration Suite has a Provisioning element that implements Service Order Management & Resource Order Management TAM applications and aligns to Service & Resource Activator OSS/J Order Management components and eTOM Service Configuration & Activation and Resource Provisioning processes and functions.

Figure 2: Provisioning in the Orchestration Suite

To enable the TIBCO® Fulfillment Orchestration Suite to provide a truly unified and cohesive solution suite, different components of the suite have been integrated. For instance, the Suite provides pre-defined inter-connectivity between Fulfillment Provisioning (FP) and Fulfillment Order Management/Fulfillment Catalog, catalog concept alignment between Fulfillment Catalog and Fulfillment Provisioning through data synchronization process, and a GUI integration for a similar look-and-feel. See **TIBCO Fulfillment Order Management Concepts and Architecture** for details.

User Interface Integration

TIBCO® Fulfillment Order Management (FOM) and TIBCO® Fulfillment Catalog (FC) interact with each other to provide complete fulfillment solution. TIBCO® Fulfillment Provisioning (FP), another optional component, enhances the overall fulfillment capability by providing a network provisioning system capable to connect directly to network elements.
The interaction among the three components is fully configurable through the user interface to maintain their anonymity and to ensure that they can work independently.

![Figure 3: Component Integration](image)

**Figure 3: Component Integration**

FP configuration is done through editable files including script files, such as data mappers, cartridge configuration files, and routing services. The catalog configuration allows to define all the product catalog related concepts, service catalog related concepts and both depending on what catalog components are deployed. Service catalog determines services and orders associated with a rendering process.

The BPMN standard is used for process modeling. It provides a single and standard process modeling interface for both TIBCO® Fulfillment Order Management and TIBCO® Fulfillment Provisioning.

The following diagram shows the different integration components:

![Figure 4: UI Integration](image)

**Figure 4: UI Integration**

**FC and FP Integration**

FC and FP component integration helps synchronize FP Service Orders data into FC with a defined ownership. For details, see TIBCO® Fulfillment Order Management User’s Guide.

**FOM and FP Integration**

FP deployment and configuration with FOM adds another distinct feature to the FOM fulfillment capability. The fulfillment request generated by FOM (Orchestrator) are routed to FP or a user defined process components. The routing is automated by using the message owner information. Based on the owner name, a fulfillment request is sent to a particular component. For example, a fulfillment request is sent to the FP component for all the orders with the owner name 'FP'.

TIBCO® Fulfillment Order Management Concepts and Architecture
Figure 5: FC-FP-FOM Integration

The user interface invokes the following TIBCO® Fulfillment Provisioning features:

- Service Order Tracking
- Service Catalog Editor
- Interface Management

See the TIBCO® Fulfillment Provisioning User’s Guide for more details.
TIBCO® Fulfillment Order Management Overview

TIBCO® Fulfillment Order Management is a meta data driven order management and fulfillment system which allows development of fulfillment plans based on meta-data specified in product catalogs. Order fulfillment and service provisioning is no longer a simple single-service or product workflow. The dynamic bundled offerings along with the explosion of devices, applications, real time inventory management and third-party content providers requires a complex order fulfillment system which can adapt to the changes in processes, meta data and inventory. Traditional OSS/BSS approach with their data silos fail to provide a dynamic and agile solution. An end-to-end order management system based on product and service catalogs is a key differentiator of Fulfillment Order Management.

TIBCO® Fulfillment Order Management is a comprehensive software solution to design, deploy and maintain high-performance scalable enterprise level business processes for advanced and dynamic order fulfillment. TIBCO® Fulfillment Order Management enables companies to quickly introduce new product offerings and in most cases requiring little or no change to fulfillment processes. The product bundles are decomposed into existing products to automatically generate a plan specific to the order.

TIBCO® Fulfillment Order Management also enables companies to efficiently manage changes to the existing business process to meet rapidly changing business environment.

Product model can be defined following SID 9 guidelines using TIBCO® Fulfillment Catalog or any other catalog management system and imported into TIBCO® Fulfillment Order Management.
In order to understand how TIBCO® Fulfillment Order Management works and how to use it, it is important to understand the key concepts. Those concepts are generic, and used extensively through the documentation.

**Topics**

- Order
- Characteristics
- Product
- Plan
- Plan Fragment
- Plan Development
- Lifecycle
- Sequences
Order

In Order Management, the order is the key concept. The Fulfillment Order Management fulfills orders. Typically, an order lists products or services that need to be fulfill. External systems may submit updates to the order as amendments, but the components within FOM may not change an order.

In the context of TIBCO® Fulfillment Order Management (FOM) an order is composed of one or more order line(s). Each order line corresponds to a requested product.

Fulfillment Order Management creates an execution plan for each order received. The plan is computed using a product model that is stored in a Product Catalog. The plan is composed of one or more plan item(s).

Figure 6: Object Model

Plan fragment, plan item and Process Component are all inter-related concepts. To clarify:

- Plan Item is one step in a plan that must be executed to reach the goal of fulfilling an order. The plan item is configured with the name of the Process Component that must be invoked to fulfill a product. The name of the Process Component is provided by AOPD during plan development and gathered from the Product Catalog using the name of the product.

- Plan Fragment is the model definition of a Process Component that will fulfill a particular product. Products are linked to plan fragments in the Product Catalogue. The name of a plan fragment will be the same as the name of the Process Component that it describes.

- Process Component is the physical implementation of the tasks required to fulfill a product. It is described by a plan fragment and invoked as a plan item step in a plan.

The logical relationship between order and order lines is shown in the following diagram:
Characteristics

Characteristics may be of the following common pre-defined types:

- **Feature** – A distinct feature or capability of a product. In general, features distinguish a product from other products of the same class. For example features of a mobile device might include: SMS, Voice, MMS, 4G, Stereo Wireless Headset, Keyboard, etc. Features could also be chargeable or non-chargeable, e.g. For billing purposes a device that provides SMS capability could mean it may need a SMS capable billing plan.

- **Instance** – Instance characteristics are similar to Features, the feature in question has measurable quantity that is defined for each related product. An example would be a discrete “Free 500 SMS Package” product could have an “Instance” characteristic called “Free SMS”. This characteristic would have a relationship value = 500. Another similar product could be created called “Free 1000 SMS Package”. It would have the same “Free SMS” characteristic associated with it but have a relationship value = 1000.

- **Input** – These characteristics represent information values that need to be captured and associated with the product at time of order/order fulfillment. They generally represent information that needs to be propagated to other systems OR will impact the fulfillment process. Input characteristics generally have no values until the order is placed/fulfilled. An example of an Input characteristic could be an MSISDN (phone number) allocated to a mobile device, or a “Contact Address” captured for a business internet product at time of order.

- **Shared** - Indicates that the attribute is shared.

Identifying Common Characteristics

The common characteristics/udf in a plan item can be identified in two ways. You can also switch between these two syntaxes using global variable `EnableAffinityUDFParent` in the AOPD component.

1. One way to identify the common characteristics/udf in a plan item is explained as follows:

   For example, consider a scenario where there is an affinity between the two plan items with UDFs sharing the same name but different values and they have plan items decomposed from the same orderline. It was difficult to identify which plan item the characteristic belongs to with the previous syntax. In the new syntax, the value at the end of the last colon (:) is always comma-separated line numbers and it becomes easier to correlate the characteristic on the basis of `affinityProductID` and `parentProductID`. The changed UDF syntax is as follows:

   `<name:parentProductID:affinityProductID:linenumber>`

   Where:
   - `name` = Name of the UDF.
   - `parentProductID` = Parent of the affinity product ID.
   - `affinityProductID` = Affinity product ID.
   - `linenumber` = Comma-separated line numbers to denote from which order lines these characteristics appeared.

   This functionality can be enabled by setting the value of global variable `EnableAffinityUDFParent` to `true`. By default, this behavior is disabled.

2. The other way to identify the common characteristics/udf in a plan item is:

   `<UDF>:<OrderLineNumber>`

   Where:
   - `UDF` = Name of UDF
   - `OrderLineNumber` = Comma-separated line numbers to denote from which order lines these characteristics appeared.
**Product**

A product is modeled in a fulfillment catalog. Orders are composed of order lines, with each order line corresponding to a particular product that is requested by a customer.

For each product that a customer orders, a series of plan items must be completed in order for that product to be provided. The link between product and plan item is maintained in the fulfillment catalog. The rules defining how different products depend on one another is also maintained in the fulfillment catalog. This then translates into dependencies between plan items in the overall execution plan.
Plan

A plan represents the tasks to be completed to reach the fulfillment goal. An order only ever has one plan associated with it, and one plan is only ever associated with a single order.

To fulfill an order, a series of tasks must be executed in a defined sequence and to a defined schedule. Sequencing and scheduling is modelled by dependencies between tasks. Once all dependencies for a task have been satisfied, then that task may be executed, with the net result being the order is fulfilled by following the required steps in the correct order.

Within FOM the series of tasks is represented by a plan object. The plan defines how to fulfill an order as a series of tasks, or plan items. Plan items are the smallest units of work recognized within FOM however they may be composed of a series of sub-tasks that actually implement the work required. This can take the form of automated back-end system invocations, manual tasks, or any other unit of work that may be required.

Plan Item

As an order consists of order lines, a plan consists of plan items. Each plan item represents a set of work that must be performed to fulfill an order. Order lines may map onto plan items, but not necessarily. However, plan items always map onto at least one order line. One order line may require multiple plan items to be fulfilled, and likewise multiple order lines may be fulfilled by a single plan item. An order line that doesn’t require the completion of any physical tasks is classified as non-executing and does not require a plan item. The abstraction and logic required to map order lines into plan items is implemented in AOPD as part of core FOM.

Milestone

Plan items are composed of a series of milestones, which represent critical points of execution. All plan items have start and end milestones, and may have zero to many intermediate milestones. This is shown in the following diagram:

Figure 8: Plan Item Milestones

In this example, plan item EP_B1 has only start and end milestones. This represents the basic set of milestones that all plan items contain. Plan item EP_B2 has start and end milestones, as well as two intermediate milestones MILE1 and MILE2.

Milestones represent the critical points during execution of a plan item and represent the points where Orchestrator can control execution of the plan item. Milestones are points where dependencies may be attached to a plan item. Dependencies may be attached only to start and intermediate milestones and may not be attached to any other point on a plan item.

Dependency

Dependencies are conditions that must be satisfied before a milestone can be considered ready. If a milestone is not yet ready, then execution may not proceed past the milestone. In the case of a start milestone, Orchestrator
will not request execution of the associated plan item until all attached dependencies are satisfied. In the case of an intermediate milestone, the Process Component must halt execution at the milestone point within its internal process model until notified by Orchestrator that a milestone is ready to fire. At that point the Process Component may continue execution. This notification may occur while the Process Component is waiting at the milestone, or at any point before execution reaches the milestone.

There are three different dependency types:
- **External** – satisfied when an external event is received by Orchestrator from an external system.
- **Time** – satisfied when a certain time period has elapsed, or a certain absolute date and time has been reached.
- **Point** – this dependency is satisfied by some milestone in another plan item having made ready.

### External
External dependencies rely on an outside event occurring before it is satisfied. This event is identified by the following parameters:
- **Event Name** – name identifying the type of event that must occur
- **Event ID** – unique identifier for a given Event Name that identifies an external dependency as being satisfied.

### Time
Time dependencies may take the form of an absolute date time, or a relative time delta. If an absolute date and time is specified then this is translated into a time delta from the point where plan execution begins. The time delta is specified in milliseconds. Once the time delta period has passed, then the dependency is considered satisfied.

### Point
Point dependencies rely on the execution sequence of other plan items to be satisfied. When a parent plan item reaches a certain intermediate or end milestone, then the dependency is satisfied. Point dependencies may use parent plan items in the current plan or in a completely different plan. If a milestone is not specified, then it is assumed that the end milestone must be made ready for the dependency to be satisfied.

Some common point dependency scenarios are shown in the following diagram:
A milestone may have zero to many dependencies attached and dependency types may be mixed between external, time, and point. A milestone may have multiple external and point dependencies, but only one time dependency is permitted. If a milestone does not have any dependencies then it is made ready immediately. Otherwise the milestone is only made ready once all dependencies are satisfied.

**Figure 9: Point Dependencies**

EP_B1 begins execution immediately.
EP_B2 may only begin execution once EP_B1 completes.

EP_B1 begins execution immediately.
EP_B1 begins execution immediately but may only proceed past MILE1 once EP_B1 completes.

EP_B1 begins execution immediately.
EP_B2 may only begin execution once MILE9 on EP_B1 completes.

EP_B1 begins execution immediately but may not proceed past MILE9 until EP_B2 has completed MILE2.
EP_B2 may only begin execution once MILE8 on EP_B1 completes.
**Plan Fragment**

A *plan fragment* is an abstraction of a Process Component that contains configuration information that Orchestrator requires in order to handle errors and SLA notifications. Plan fragments are optional. If no plan fragment is defined for a particular Process Component then Orchestrator will use engine defaults to handle errors and no SLA notifications will occur.

During the evolution of a system it may be necessary to deploy multiple versions of a Process Component simultaneously. To support this plan fragments may be versioned. The relationship between plan fragments, versions, and process components is shown in the following diagram:

![Diagram showing the relationship between plan fragments, versions, and process components](image)

**Figure 10: Plan Fragment and Process Component Logical Components**

In this example plan fragment PC_1000 describes Process Component PC_1000. This Process Component may be invoked by a plan item using different versions. Therefore plan fragment PC_1000 has Version 1 which maps the Process Component of the same version. The base plan fragment always defines the currently active Process Component version.

The jeopardy detection and rules for consequential action are not applied for any execution containing process component mapping with no valid plan fragment model.

**Error Handling**

In the event that a Process Component returns a failed or incomplete execution response Orchestrator handles the error using engine default configurations. Standard error handling functionality is to retry the plan item for a defined number of times, with a defined delay interval between invocations before referring it to the Plan Item Error Handler with the name of a default error handler for manual intervention. If a plan fragment is defined for a Process Component, these error handling properties may be overridden.

If the retry override flag is set to true, then the plan fragment configuration for retry will be used instead of the engine configuration. The following parameters may be set:

- **Retry Count** – the number of times to retry the plan item on failure before referring it to the Plan Item Error Handler.
- **Retry Delay** – the delay in msec to wait between plan item retries.

When invoking the Plan Item Error Handler Orchestrator specifies the error handling module that is relevant for the plan item being submitted. Generally this is a default error handler, but the name of the error handler may be overridden by specifying it in the plan fragment.
SLA Notification

SLA notifications are sent out by Orchestrator when plan items exceed expected typical, threshold [specific percentage of maximum duration] and maximum execution durations. These typical and maximum durations are modelled on plan fragment sections, which represent the part of a plan item that executes between two milestones. For example:

**Figure 11: Plan Fragment Sections**

In this example plan item EP_B1 has an associated plan fragment with a single plan fragment section that defines the typical and maximum execution duration between the start and end milestones.

Plan item EP_B2 has an associated plan fragment also with a plan fragment section that defines the typical and maximum execution duration between the start and end milestones. But it also has 3 other sections that define typical and maximum execution durations between start and mile1, mile1 and mile2, and mile2 and end.

Plan fragment sections are also versioned, so they exist within the plan fragment hierarchy as follows:
Figure 12: Plan Fragment Logical Components
Plan Development

During order plan development, Orchestrator calls out to AOPD to design the plan of action to fulfill an order. AOPD will analyze the order and the Product Catalog and determine what plan items must be executed and in what order in order to fulfill an order.

AOPD will look at the product on an order line and determine one of the following scenarios:

- The product is non-executing, in which case no plan item is required and the order line is marked as complete.
- The product requires a single plan item. When this plan item completes then the order line is marked as complete.
- The product requires multiple plan items to be executed in a defined sequence. The final plan item in the sequence will be flagged as end-of-line (EOL). When this plan item completes then the order line is marked as complete.
- The product has previously been provisioned. The plan item or items will be created as above, but the status set immediately to complete. The order line is also marked as complete.
- Plan item dependencies are determined by rules and the configuration of the product model hierarchy

The logical relationship between order, order lines, plan, and plan items is shown in the following diagram:

![Order and Plan Logical Components](image)

**Figure 13: Order and Plan Logical Components**

In this example, AOPD has analyzed the order and determined the following:

1. Order Line 1 orders Product A. This product is fulfilled by Process Component EP_B1. This component has no dependencies so it begins execution immediately.
2. Order Line 2 orders Product B. This is a non-executing product so there is no associated plan item and no execution occurs.
4. Order Line 4 orders Product D. This product is fulfilled by Process Component EP_B4. This component may only begin execution once EP_B1 completes.
5. Order Line 5 orders Product E. This product is fulfilled by Process Component EP_B5. This component may only begin execution once both EP_B3 and EP_B4 complete.
Lifecycle

During fulfillment the different entities managed by Orchestrator go through a defined status lifecycle.

Order

The order lifecycle is shown in the following diagram:

![Order Lifecycle Diagram]

Figure 14: Order Lifecycle

The lifecycle steps are summarized below:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submitted</td>
<td>The order has been submitted to Orchestrator and successfully stored in the database. Processing has started.</td>
</tr>
<tr>
<td>Feasibility</td>
<td>The order is currently being evaluated for feasibility. If feasibility has been enabled, Orchestrator has requested a feasibility analysis from the Feasibility Provider. If the Feasibility Provider deems the order not feasible then the next status is Pre-Qualification Failed if feasibility error handling is enabled, or Withdrawn if error handling is not enabled. Otherwise the next status is Plan Development. Otherwise the order is automatically deemed feasible and the next status is Plan Development.</td>
</tr>
<tr>
<td>Plan Development</td>
<td>The order has been deemed feasible and is currently being analyzed for plan development. Orchestrator has sent a request to AOPD to design a plan. If AOPD is unable to design a plan then the next status is Pre-Qualification Failed if OPD error handling is enabled, or Withdrawn if error handling is not enabled. Otherwise the next status is Execution.</td>
</tr>
<tr>
<td>Execution</td>
<td>The order is feasible and a plan has been created and linked to the order. The plan has been successfully stored in cache and is progressing through the plan lifecycle. All order lines are Pending. The order is being fulfilled by Orchestrator.</td>
</tr>
<tr>
<td>Step</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Complete</td>
<td>The order has completed fulfillment successfully. The plan and all plan items in the plan are Complete, and all order lines are Complete. The order status in the database has been updated to completed.</td>
</tr>
<tr>
<td>Pre-Qualification Failed</td>
<td>The order does not meet the criteria for feasibility or for plan development and has been referred to a Pre-Qualification Failed Handler for further analysis before proceeding.</td>
</tr>
<tr>
<td></td>
<td>If the Pre-Qualification Failed Handler returns Order Request then that order will be submitted as an amendment and the next order status will be Submitted.</td>
</tr>
<tr>
<td></td>
<td>If the Pre-Qualification Failed Handler returns Retry OPD then the next order status will be OPD.</td>
</tr>
<tr>
<td></td>
<td>If the Pre-Qualification Failed Handler returns Retry Feasibility then the next order status will be Feasibility. Likewise if it returns Retry Plan Development then the next order status will be Plan Development.</td>
</tr>
<tr>
<td></td>
<td>If the Pre-Qualification Failed Handler returns Withdraw then the next order status will be Withdrawn.</td>
</tr>
<tr>
<td>Suspended</td>
<td>The order has stopped fulfillment and is awaiting further instructions. A suspended order may be returned to Execution either with or without an amendment applied.</td>
</tr>
<tr>
<td>Cancelled</td>
<td>The order has cancelled fulfillment successfully. The plan and all plan items in the plan are Cancelled, and all order lines are Cancelled. The order status in the database has been updated to cancelled.</td>
</tr>
<tr>
<td></td>
<td>Note that the order will only go to this status in the event of a full order cancel. If a partial order cancel occurs then the order will go to Complete status.</td>
</tr>
<tr>
<td>Withdrawn</td>
<td>The order has been withdrawn and deleted from the database.</td>
</tr>
</tbody>
</table>

**Order Line**

The order line lifecycle is shown in the following diagram:
The lifecycle steps are summarized below:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pending</td>
<td>The order has been successfully submitted and the plan has been developed. The associated plan items for the order line either are ready to execute or currently executing.</td>
</tr>
<tr>
<td>Complete</td>
<td>The order line has been successfully completed. The plan item that has specified this order line as end-of-line (EOL) has successfully completed.</td>
</tr>
<tr>
<td>Cancelled</td>
<td>The order line has been successfully cancelled. The plan item that has specified this order line as end-of-line (EOL) has successfully cancelled.</td>
</tr>
</tbody>
</table>

**Order Amendment**

The order amendment lifecycle only occurs for orders that are currently executing. For amendments that occur before execution, then any previous plan is simply discarded and recreated using the new order.

For amendments during execution the order amendment lifecycle is shown in the following diagram:

The lifecycle steps are summarized below:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submitted</td>
<td>The order amendment has been submitted to Orchestrator and successfully stored in cache. Processing has started.</td>
</tr>
<tr>
<td>Feasibility</td>
<td>The order amendment is currently being evaluated for feasibility.</td>
</tr>
</tbody>
</table>
If feasibility has been enabled, Orchestrator has requested a feasibility analysis from the Feasibility Provider. If the Feasibility Provider deems the order amendment not feasible then the next status is Pre-Qualification Failed if feasibility error handling is enabled, or Withdrawn if error handling is not enabled. Otherwise the next status is Plan Development.

Otherwise the order amendment is automatically deemed feasible and the next status is Plan Development.

The order amendment has been deemed feasible and is currently being analyzed for plan development. Orchestrator has sent a request to AOPD to design a plan.

If AOPD is unable to design a plan then the next status is Pre-Qualification Failed if AOPD error handling is enabled, or Withdrawn if error handling is not enabled. Otherwise the next status is Execution.

The order amendment is being processed by Orchestrator and the plan updated.

The order amendment has completed successfully. The plan has been updated.

The order amendment does not meet the criteria for feasibility or for plan development and has been referred to a Pre-Qualification Failed Handler for further analysis before proceeding.

If the Pre-Qualification Failed Handler returns New Order then that order amendment will be submitted as a new amendment and the current amendment will end.

If the Pre-Qualification Failed Handler returns Retry Feasibility then the next status will be Feasibility. Likewise if it returns Retry Plan Development then the next status will be Plan Development.

If the Pre-Qualification Failed Handler returns Withdraw then the next status will be Withdrawn.

The order amendment has been withdrawn and deleted from cache.

### Plan

The overall plan lifecycle is shown in the following diagram:
Figure 17: Plan Lifecycle

The lifecycle steps are summarized below:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pending</td>
<td>The plan has been successfully created and is awaiting execution.</td>
</tr>
<tr>
<td>Execution</td>
<td>The plan is currently being executed. Plan item requests are being sent to the appropriate Process Components in the correct sequence.</td>
</tr>
<tr>
<td></td>
<td>If the plan completes successfully then the next status is Complete. This includes any order amendment except for a full plan cancel. If the plan</td>
</tr>
<tr>
<td></td>
<td>has been cancelled completely then the next status is Cancelled.</td>
</tr>
<tr>
<td></td>
<td>If a suspend plan request is made then the next status will be Suspended.</td>
</tr>
<tr>
<td>Complete</td>
<td>The plan has completed.</td>
</tr>
<tr>
<td>Suspended</td>
<td>The plan has stopped fulfillment and is awaiting further instructions. A plan order may be returned to Execution either with or without an amendment applied.</td>
</tr>
<tr>
<td>Cancelled</td>
<td>The plan has been successfully cancelled.</td>
</tr>
</tbody>
</table>

Plan Item

The overall plan item lifecycle is show in the following diagram:

Figure 18: Plan Item Lifecycle

The lifecycle steps are summarized below:
### Step

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pending</td>
<td>The plan item has been successfully created and is awaiting execution. This will be waiting for required dependencies to be satisfied before proceeding.</td>
</tr>
</tbody>
</table>
| Execution | The plan item is currently being executed. A request has been issued to the Process Component to execute.  
If the Process Component returns a result that it completed successfully then the next status is Complete. If the Process Component returns a result that it cancelled successfully then the next status is Cancelled.  
If a suspend plan item request is made and the Process Component returns a result indicating that the suspend was successful then the next status will be Suspended. |
| Complete | The plan item has completed. The Process Component has returned a response and indicated that it completed successfully. |
| Suspended | The plan item has stopped fulfillment and is awaiting further instructions. A request was sent to the Process Component to suspend execution, and it returned a successful suspend response. |
| Cancelled | The plan item has been successfully cancelled. |

### Milestone

The overall milestone lifecycle is shown in the following diagram:

![Diagram of Milestone Lifecycle](image)

**Figure 19: Milestone Lifecycle**

The lifecycle steps are summarized below:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pending</td>
<td>The milestone is waiting for all its dependencies to be satisfied.</td>
</tr>
<tr>
<td>Complete</td>
<td>All dependencies have been satisfied for the milestone and execution has continued.</td>
</tr>
</tbody>
</table>

### Dependency

The overall dependency lifecycle is shown in the following diagram:
The lifecycle steps are summarized below:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pending</td>
<td>The dependency is waiting to be satisfied.</td>
</tr>
<tr>
<td>Complete</td>
<td>The dependency has been satisfied.</td>
</tr>
</tbody>
</table>
Sequences

Standard Order

This is the normal flow of events through standard order fulfillment with no exceptions. The sequence is shown in the following diagram:

Figure 21: Standard Order Fulfillment – Successful Completion Sequence

1. The order is submitted from Order Entry through the Orchestrator Submit Order interface. Note that this may use intermediate service layers. The order now has Start status.
2. Orchestrator sends a request to cache to store the order. The order now has Submitted status.
3. Cache saves the order and returns a response to Orchestrator. The order now has Feasibility status.
4. If feasibility is enabled, Orchestrator sends a request to the Feasibility Provider to perform feasibility checking on the order.
a. Feasibility Provider may delegate the feasibility checking call to Validation as well as perform some internal checking.
b. Validation returns the result of the order validation back to Feasibility Provider.
c. Feasibility Provider aggregates all order feasibility checks and concludes that the order is feasible and sends a response back to Orchestrator.

5. If feasibility is not enabled or the Feasibility Provider has returned the result then the order status is now Plan Development.
6. Orchestrator sends a request to AOPD to analyze the order and design an execution plan.
7. AOPD sends a response back to Orchestrator with the execution plan definition. Orchestrator then generates a plan based on this definition. The order now has Execution status and the plan now has Start status.
8. Orchestrator sends a request to cache to store the plan. The plan now has Submitted status.
9. Cache saves the plan and returns a response to Orchestrator. The plan now has Pending status.
10. Orchestrator changes the plan status to Execution and begins invoking Process Components in the correct sequence. This is repeated for each plan item.
11. Process Component returns a response to Orchestrator for each invocation.
12. Once all plan items have completed, the plan is set to Complete status. Orchestrator sends a request to cache to change the order status to complete for archiving. The order status then goes to Complete in Orchestrator.

Successful Completion

This is the normal flow of events through standard order fulfillment with no exceptions. The sequence is shown in the following diagram:
Figure 22: Standard Order Fulfillment – Successful Completion Sequence

1. The order is submitted from Order Entry through the Orchestrator Submit Order interface. Note that this may use intermediate service layers. The order now has Start status.

2. Orchestrator sends a request to Transient Data Store to store the order. The order now has Submitted status.

3. Transient Data Store saves the order and returns a response to Orchestrator. The order now has Feasibility status.

4. If feasibility is enabled, Orchestrator sends a request to the Feasibility Provider to perform feasibility checking on the order.
   a. Feasibility Provider may delegate the feasibility checking call to Validation as well as perform some internal checking.
   b. Validation returns the result of the order validation back to Feasibility Provider.
   c. Feasibility Provider aggregates all order feasibility checks and concludes that the order is feasible and sends a response back to Orchestrator.
5. If feasibility is not enabled or the Feasibility Provider has returned the result then the order status is now Plan Development.

6. Orchestrator sends a request to AOPD to analyze the order and design an execution plan.

7. AOPD sends a response back to Orchestrator with the execution plan definition. Orchestrator then generates a plan based on this definition. The order now has Execution status and the plan now has Start status.

8. Orchestrator sends a request to Transient Data Store to store the plan. The plan now has Submitted status.

9. Transient Data Store saves the plan and returns a response to Orchestrator. The plan now has Pending status.

10. Orchestrator changes the plan status to Execution and begins invoking Process Components in the correct sequence. This is repeated for each plan item.

11. Process Component returns a response to Orchestrator for each invocation.

12. Once all plan items have completed, the plan is set to Complete status. Orchestrator sends a request to Transient Data Store to change the order status to complete for archiving. The order status then goes to Complete in Orchestrator.

**Feasibility Failed**

This is the flow of events through standard order fulfillment with the Feasibility Provider indicating that the order is not feasible. The sequence is shown in the following diagram:
1. The order is submitted from Order Entry through the Orchestrator Submit Order interface. Note that this may use intermediate service layers. The order now has Start status.

2. Orchestrator sends a request to Transient Data Store to store the order. The order now has Submitted status.

3. Cache saves the order and returns a response to Orchestrator. The order now has Feasibility status.

4. Orchestrator sends a request to the Feasibility Provider to perform feasibility checking on the order.

5. Feasibility Provider may delegate the feasibility checking call to AFF Validation as well as perform some internal checking.

6. Validation returns the result of the order validation back to Feasibility Provider.

7. Feasibility Provider aggregates all order feasibility checks and concludes that the order is not feasible and sends a response back to Orchestrator. The order status is now Pre-Qualification Failed.

8. Orchestrator sends the order and the validation messages from the Feasibility Provider to the Pre-Qualification Failed Handler for manual intervention.

9. Pre-Qualification Failed Handler sends a response back to Orchestrator with one of two possible actions:
The order is submitted from Order Entry through the Orchestrator Submit Order interface. Note that this may use intermediate service layers. The order now has Start status.

2. Orchestrator sends a request to Transient Data Store to store the order. The order now has Submitted status.

3. Transient Data Store saves the order and returns a response to Orchestrator. The order now has Feasibility status.

4. If feasibility is enabled, Orchestrator sends a request to the Feasibility Provider to perform feasibility checking on the order.
a. Feasibility Provider may delegate the feasibility checking call to Validation as well as perform some internal checking.

b. Validation returns the result of the order validation back to Feasibility Provider.

c. Feasibility Provider aggregates all order feasibility checks and concludes that the order is feasible and sends a response back to Orchestrator.

5. If feasibility is not enabled or the Feasibility Provider has returned the result then the order status is now Plan Development.

6. Orchestrator sends a request to AOPD to analyze the order and design an execution plan.

7. AOPD sends a response back to Orchestrator indicating the order is not valid for plan design. The order status is now Pre-Qualification Failed.

8. Orchestrator sends the order and the validation messages from AOPD to the Pre-Qualification Failed Handler for manual intervention.

9. Pre-Qualification Failed Handler sends a response back to Orchestrator with one of two possible actions:
   a. Withdraw the order, which will terminate execution and the order status is now Withdrawn.
   b. Resubmit the order with an updated version. This flow continues as a normal order amendment flow.

**Plan Item Execution Failed**

This is the flow of events through standard order fulfillment with one or many Process Components indicating that execution of a plan item has failed. The sequence is shown in the following diagram:
Figure 25: Standard Order Fulfillment – Plan Item Execution Failed Sequence

1. The order is submitted from Order Entry through the Orchestrator Submit Order interface. Note that this may use intermediate service layers. The order now has Start status.

2. Orchestrator sends a request to Transient Data Store to store the order. The order now has Submitted status.

3. Transient Data Store saves the order and returns a response to Orchestrator. The order now has Feasibility status.

4. If feasibility is enabled, Orchestrator sends a request to the Feasibility Provider to perform feasibility checking on the order.
   a. Feasibility Provider may delegate the feasibility checking call to Validation as well as perform some internal checking.
   b. Validation returns the result of the order validation back to Feasibility Provider.
   c. Feasibility Provider aggregates all order feasibility checks and concludes that the order is feasible and sends a response back to Orchestrator.
5. If feasibility is not enabled or the Feasibility Provider has returned the result then the order status is now Plan Development.

6. Orchestrator sends a request to AOPD to analyze the order and design an execution plan.

7. AOPD sends a response back to Orchestrator with the execution plan definition. Orchestrator then generates a plan based on this definition. The order now has Execution status and the plan now has Start status.

8. Orchestrator sends a request to Transient Data Store to store the plan. The plan now has Submitted status.

9. Transient Data Store saves the plan and returns a response to Orchestrator. The plan now has Pending status.

10. Orchestrator changes the plan status to Execution and begins invoking Process Components in the correct sequence. This is repeated for each plan item.

11. Process Component returns a response to Orchestrator for each invocation.

12. If a Process Component response indicates that the invocation was incomplete or resulted in an error, then Orchestrator sends a request to the Plan Item Error Handler with the details of the plan item failure for manual intervention.

13. The Plan Item Error Handler sends a response back to Orchestrator with one of three possible actions:
   a. Complete the plan item and continue execution as normal.
   b. Resume the plan item from the point the error occurred and continue execution as normal.
   c. Retry the plan item from the beginning and continue execution as normal.

**Processing New Order with MOPD Enabled**

The sequence diagram for processing a new order with MOPD enables is as follows:

![Sequence Diagram for Processing New Order with MOPD Enabled](image)

**Figure 26: Sequence Diagram for Processing New Order with MOPD Enabled**

For a newly submitted order, a template plan is generated which can be edited by the user from FOM UI. This template plan can be generated using following template implementation:

- No Template
- Default Template
• Custom Template
• JMS Template
• Using AOPD

The following table provides the implementation details for generating template plan. Some of these implementations are supported by FOM application and user customization of implementation is also supported.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Property Value</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>com.tibco.fom.orch.mopd.templatePlan</td>
<td>No Template Implementation will be used.</td>
</tr>
<tr>
<td></td>
<td>Default</td>
<td>Default Template Implementation will be used.</td>
</tr>
<tr>
<td></td>
<td>Custom</td>
<td>Custom Template Implementation will be used. Fully qualified class name of custom implementation will be used by specifying property com.tibco.fom.orch.mopd.customTemplatePlanCreator.</td>
</tr>
<tr>
<td></td>
<td>JMS</td>
<td>JMS Template Implementation will be used</td>
</tr>
<tr>
<td></td>
<td>AOPD</td>
<td>AOPD will be used to generate template plan</td>
</tr>
</tbody>
</table>

No Template Implementation

No Template Implementation is supported by the application. The template plan generated will only have a plan wrapper without any plan item details. Configure the property using the following code:

```xml
<ConfValue description="Template for Mopd plan generation" isHotDeployable="true"
    name="Template for Mopd plan generation" propname="com.tibco.fom.orch.mopd.templatePlan"
    readonly="false" sinceVersion="3.0" visibility="Basic">
  <ConfString default="AOPD" value="No"/>
</ConfValue>
```

Default Template Implementation

Default Template Implementation is also supported by the application. The template plan generated will have plan details with one plan item details for each order line mentioned in the order details. Configure the property using the following code:

```xml
<ConfValue description="Template for Mopd plan generation" isHotDeployable="true"
    name="Template for Mopd plan generation" propname="com.tibco.fom.orch.mopd.templatePlan"
    readonly="false" sinceVersion="3.0" visibility="Basic">
  <ConfString default="AOPD" value="Default"/>
</ConfValue>
```

Custom Template Implementation

Custom Template Implementation can be implemented by the user. The details for implementing custom template to generate plan are as follows:

- Create a new java project in IDE with omsCommon.jar in build path.
- Implement an interface with name com.tibco.aff.oms.server.jms.orch.mopd.MOPDTemplate.
- Build the code and include the .class/jar file in WEB-INF/classes(.class) or WEB-INF/lib(jar) folder of omsServer.war.
- Configure the following property through the Configurator:

```xml
<ConfValue description="Template for Mopd plan generation" isHotDeployable="true"
    name="Template for Mopd plan generation" propname="com.tibco.fom.orch.mopd.templatePlan"
    readonly="false" sinceVersion="3.0" visibility="Basic">
  <ConfString default="AOPD" value="Custom"/>
</ConfValue>
```
- Configure the implemented class name through Configurator:

```xml
<ConfValue description="Fully qualified class name of custom Mopd plan creator"
    isHotDeployable="true" name="Custom Mopd Plan Creator" propname="com.tibco.fom.orch.mopd.customTemplatePlanCreator"
    readonly="false"/>
```
- Redeploy the OMS Server.
- A sample do-nothing implementation is included with distribution in `omsCommon.jar` with the name `com.tibco.aff.oms.server.jms.mopd.custom.CustomMOPDTemplateImpl`. This is only for testing and will not generate a valid plan to proceed further. If enabled, the following logs will be generated: Generate template plan through {} for orderID {}.
- By default, blank value is present in the Configurator that indicates no plan generation by custom implementation.

### JMS Template Implementation

JMS Template Implementation is also supported by application. The request to generate plan will be published to a JMS destination and the generated plan will be published back to the JMS destination. Configure the property using the following code:

```xml
<ConfValue description="Template for Mopd plan generation" isHotDeployable="true" name="Template for Mopd plan generation" propname="com.tibco.fom.orch.mopd.templatePlan" readonly="false" sinceVersion="3.0" visibility="Basic">
  <ConfString default="AOPD" value="Jms" />
</ConfValue>
```

You can configure the JMS destination for publishing request payload to generate plan using the following property:

```xml
<ConfValue description="Queue for sending order request to generate template plan for MOPD" propname="com.tibco.fom.oms.orch.mopd.template.orderrequest.sender.queue" readonly="false" sinceVersion="3.0" visibility="Basic">
  <ConfString default="" value="" />
</ConfValue>
```

You can configure JMS destination for receiving the template plan payload using the following property:

```xml
<ConfValue description="Queue to receive response of generated template plan for MOPD" propname="com.tibco.fom.oms.orch.mopd.template.response.queue" readonly="false" sinceVersion="3.0" visibility="Basic">
  <ConfString default="" value="" />
</ConfValue>
```

The default value is always blank for the configuration.

### Using AOPD

The plan will be generated by AOPD. Configure the property using the following code:

```xml
<ConfValue description="Template for Mopd plan generation" isHotDeployable="true" name="Template for Mopd plan generation" propname="com.tibco.fom.orch.mopd.templatePlan" readonly="false" sinceVersion="3.0" visibility="Basic">
  <ConfString default="AOPD" value="AOPD" />
</ConfValue>
```

This is the default configuration to generate template plan.

### Amend Order Fulfillment

#### Before Plan Creation

Order amendment is the process of modifying an order after submission. An amendment before plan development occurs prior to the creation of a plan within Orchestrator. The sequence is shown in the following diagram. Note that Validation has been removed for simplicity.
Figure 27: Amend Order Fulfillment – Before Plan Creation Sequence

In this sequence the amendment is shown as occurring after the response from AOPD. Note that the amendment may occur at any point before this event. The key factor is that the plan has not yet been created in Orchestrator.
1. The order is submitted from Order Entry through the Orchestrator Submit Order interface. Note that this may use intermediate service layers. The order now has Start status.

2. Orchestrator sends a request to Transient Data Store to store the order. The order now has Submitted status.

3. Transient Data Store saves the order and returns a response to Orchestrator. The order now has Feasibility status.

4. Orchestrator sends a request to the Feasibility Provider to perform feasibility checking on the order.

5. Feasibility Provider aggregates all order feasibility checks and concludes that the order is feasible and sends a response back to Orchestrator. The order status is now Plan Development.

6. Orchestrator sends a request to AOPD to analyze the order and design an execution plan.

7. AOPD sends a response back to Orchestrator with the execution plan definition. Orchestrator then generates a plan based on this definition. The order now has Execution status and the plan now has Start status.

8. The order amendment is submitted from Order Entry through the Orchestrator Submit Order interface.

9. Orchestrator sends a request to Transient Data Store to store the amended order. The order now has Submitted status.

10. Transient Data Store saves the amended order and returns a response to Orchestrator. The order now has Feasibility status.

Processing now continues as in the normal Standard Order Fulfillment case.

**Amend Order Fulfillment**

Order amendment is the process of modifying an order after submission. An amendment after plan development occurs after the creation of a plan within Orchestrator. Note that Validation has been removed for simplicity.
Figure 28: Amend Order Fulfillment – After Plan Creation Sequence

In this sequence the amendment is shown as occurring during execution.

1. The order is submitted from Order Entry through the Orchestrator Submit Order interface. Note that this may use intermediate service layers. The order now has Start status.

2. The sequence of events follows the Standard Order Fulfillment sequence until the order amendment is submitted.

3. The order amendment is submitted from Order Entry through the Orchestrator Submit Order interface.

4. Orchestrator sends a suspend request to all executing plan items. The plan and order status are now both Suspended.

5. Each executing plan item returns a response indicating it suspended successfully or completed.

6. Orchestrator sends a request to Transient Data Store to store the amended order. The order amendment now has Submitted status. The order and plan are still in Suspended status.

7. Transient Data Store saves the order amendment and returns a response to Orchestrator. The order amendment now has Feasibility status. The order and plan are still in Suspended status.
8. If feasibility is enabled, Orchestrator sends a request to the Feasibility Provider to perform feasibility checking on the order amendment.

9. Feasibility Provider aggregates all order amendment feasibility checks and concludes that the order amendment is feasible and sends a response back to Orchestrator. The order amendment status is now Plan Development. The order and plan are still in Suspended status.

10. Orchestrator sends a request to AOPD to analyze the order amendment and design an amended execution plan.

11. AOPD sends a response back to Orchestrator with the amended execution plan definition. Orchestrator then updates the plan based on this definition. The order amendment status is now Amending. The order and plan are still in Suspended status.

12. Orchestrator sends a request to Transient Data Store to store the updated plan.

13. Transient Data Store saves the updated plan and returns a response to Orchestrator. The order amendment status is now complete. The order and plan are still in Suspended status.

14. Orchestrator changes the order and plan status to Execution. Orchestrator then sends an activate request to all suspended plan items. The activate message will instruct the Process Component to do one of the following:
   a. Resume execution from the point of suspension
   b. Cancel execution but do not rollback any previously completed tasks
   c. Cancel execution and rollback any previously completed tasks

15. Process Components return a response to Orchestrator for each activation.

16. Orchestrator continues invoking Process Components in the correct sequence. This is repeated for each plan item.

Processing now continues as in the normal Standard Order Fulfillment case.

Processing Amended Order with MOPD Enabled

The sequence diagram for processing an amended order with MOPD enables is as follows:

Figure 29: Sequence Diagram for Processing Amended Order with MOPD Enabled
The following steps provide a high level flow for manual order submission in case of amendment:
1. Identify an order which needs to be amended.
2. Suspend the identified order. Order is suspended and database is updated.
3. Submit an amendment request.
4. Orchestrator sends the amend order Request to AOPD, JMS, or third party for draft plan generation.
5. Orchestrator receives the draft plan and stops the further execution until the final plan is not received.
6. Search for the manual order which needs to be edited manually.
7. Retrieve the order details for manual order and gets order details in OMSUI.
8. Traverse to the draft plan which was saved earlier.
9. Draft plan in UI is visible.
10. Provide Instructions to get draft plan in UI. This provides an indication to the system that you want to edit the plan.
11. Edit the plan by adding, modifying, or deleting plan item, milestones, and dependencies in the plan developed by AOPD.
12. Edit the draft plan through FOM UI and provide instructions to save the plan in database.
13. Plan is saved in database.
14. Provide instructions to execute this plan.
15. Plan is validated using the validation framework provided by the server.
16. If plan is valid then the orchestrator is notified with the final plan and it starts execution of the plan.
17. If plan is not valid then application returns back specifying that plan is not valid and the plan can be corrected.

For amended order, a template plan will be generated, which you can edit from FOM-UI. This template plan can be generated in following ways:
1. Using the existing plan
2. New plan generated by the AOPD

The following property provides an indication whether the plan will be generated using AOPD or not:

```
<ConfValue description="Template for Mopd amendment plan generation" isHotDeployable="true"
    name="Template for Mopd amendment plan generation" propname="com.tibco.fom.orch.mopd.amendment.templatePlan"
    readonly="false" sinceVersion="3.0" visibility="Basic">
    <ConfString default="AOPD" value="AOPD"/>
</ConfValue>
```

A property with value FALSE indicates that the existing plan will not be used and a plan will be generated using AOPD for amended order. A property with value TRUE indicates that existing plan will be used as template plan for editing plan by user.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Value</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.tibco.fom.orch.mopd.amendment.templatePlan</td>
<td>AOPD</td>
<td>Template plan will be generated using AOPD for editing by user</td>
</tr>
<tr>
<td>existingPlan</td>
<td>Use existing old plan as template plan for this order</td>
<td></td>
</tr>
</tbody>
</table>

**Cancel Order**

Cancel Order Fulfillment is a special case of Amend Order Fulfillment. The same sequence of events is followed, except at the end of the process the overall order and plan status goes to Cancelled rather than Completed.
Chapter 3

Architecture

Topics

• Overview
• Order Management Server
• Orchestrator
• Automated Order Plan Development
• Manual Order Plan Development
• Order Capture System
• Offer and Price Engine
• Process Components
• Feasibility Provider
• Jeopardy Management
• Router
• Key Functionality
Overview

TIBCO® Fulfillment Order Management is made of several components. Each component has a particular role.

The major components of TIBCO® Fulfillment Order Management include:

- **Order Manager Server (OMS)** - exposes SOAP Web services that can be used by external systems to submit orders to the Fulfillment Order Management.

- **Orchestrator** - Takes the order plan from AOPD and executes those tasks till completion. It invokes micro-level process plan fragments to initiate tasks within the operator’s operations ecosystem, enabling appropriate actions to take place in a variety of back-end systems (for example, billing systems, network systems, scheduling systems). Orchestrator keeps track of status and manages exceptions.

- **Automatic Order Plan Development (AOPD)** - Valid orders accepted by the OMS system are decomposed into their individual products, services and resources. An optimized order plan workflow process is then generated based on those basic building blocks to ensure an accurate order fulfillment. Optimization can take into account both product rules, as well as customer inventory and other data to arrive at the final order plan.

- **Jeopardy Manager System (JeoMS)** - The Jeopardy Management System is a key component of Fulfillment Order Management (FOM). Jeopardy management is the process of monitoring execution of a set of tasks in a plan to fulfill a customer order. In FOM, 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 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.

- **Order Capture System (OCS)** - The Order Capture System is a web application component with a UI that allows you to create, manage, and submit TIBCO® Fulfillment Orchestration Suite (FOS) orders based on what a subscriber already has. Order Capture System allows you to select subscribers and browse validated products, services, or bundles from the TIBCO® Fulfillment Catalog (FC). OCS interacts with the Runtime Model Server, which handles data, synchronizes it with all systems, and monitors the life cycle of the shopping cart.

- **Order and Pricing Engine (OPE)** - Offer and Price Engine is a part of the FOM engine that replaces the existing Offer Configuration and Validation (OCV) engine. The Order and Price Engine evaluates eligibility for products, validates the orders, and provides prices.

There are additional components, that are explained in more details in the *TIBCO Fulfillment Order Management User’s Guide*. Here are few of those additional components:

- **Fulfillment Order Management User Interface (OMSUI)** - Provides operators a GUI to manage and track orders. Order Management System (OMS) persists order data and allows operators to search, view, track and trace orders, as well as allows users to take actions on order/order lines.

- **Router** - Orders from OMS may optionally be routed to an alternate Orchestration engine. In this case, Fulfillment Order Management Router can read inbound orders and, based on rules, may route to an external orchestration engine.

- **Common Logging** - All Fulfillment Order Management components report to a common logging component for the ease of maintenance and operations management of the system.
Figure 30: Fulfillment Order Management Architecture
Order Management Server

Order Management Server (OMS) provides an input interface for external systems.

Figure 31: Order Management Server

OMS exposes Web service that can be used by external systems to submit orders to the Fulfillment Order Management.

OMS hosts the following web services:
1. SubmitOrder
2. AmendOrder
3. CancelOrder
4. getOrderDetails
5. GetOrders
6. GetOrderExecutionPlan
7. PerformBulkOrderAction
8. SyncSubmitOrder
9. WithdrawOrder
10. SuspendOrder
11. ActivateOrder
12. GetEnrichedExecutionPlan

See the TIBCO Fulfillment Order Management Web Services Guide for more details on the Web Services definitions.
Orchestrator

Orchestrator receives orders from OMS. Then Orchestrator executes a series of tasks in a defined order. Orchestrator interacts with several other components to store orders, to create plan specifications. Orchestrator is also responsible to communicate with external systems (process components).

Figure 32: Orchestrator

Orchestrator is responsible for the following:

1. Manage the overall order lifecycle.
2. Store the order in cache.
3. Optionally determine order feasibility by sending the order to a Feasibility Provider which returns the result back to Orchestrator.
4. Develop a plan for fulfilling the order by sending the order to AOPD which returns the plan specification.
5. Use this plan specification to create the execution plan for an order.
6. Store the plan in cache.
7. Interpret the execution plan and coordinate order fulfillment by invoking the correct Process Components in the correct order.
8. Update the order status to complete in the cache at the end of the order lifecycle.

When OMS sends an order, submitted by an external system, to the Orchestrator, the order may refer to several products.

For each ordered product, a series of plan items must be completed sequentially for that product to be provided. The Product Catalog maintains the link between product and plan item. Orchestrator receives the requests for order fulfillment. Orchestrator component in turn sends the order to AOPD to analyze the order and the Product Catalog, and determines the plan of action to fulfill the order. Orchestrator then uses this plan to reach the goal by invoking the process component associated with each plan item in the defined sequence to fulfill an order. For details, see TIBCO Fulfillment Order Management Administration Guide.

The actual fulfillment of the product happens by invoking the process component - typically implemented as Fulfillment Provisioning cartridges or the BPM workflow processes - described by the plan fragment assigned to the product in the Product Catalog. The invocation of the process components in a specific sequence and at specific time is known as the order orchestration, which is done by the Orchestrator.

Management Orchestrator receives AOPD-generated execution plan for order orchestration. It has one to many inter-dependent plan items, which typically maps to the order lines in the order. See figure Figure 33: Order and Execution Plan on page 60.
There can be one-to-one, one-to-many, or many-to-one relationships between the order lines and the plan items based on the Product Catalogue modeling.

- In case of Affinity between two products, the two order lines requesting these two products have a single plan item in the execution plan, to fulfill the product products simultaneously.
- In case of a bundled product comprising of sub-products and services, the order line requesting this product have multiple plan items in the execution plan, one corresponding to each sub-product or service.

A plan item contains the process component which has to be invoked for the fulfillment of a particular product. AF Orchestrator invokes the process components and starts executing the plan contained in the plan items according to the dependencies between them. The execution plan, and hence the order is considered to be COMPLETE or FULFILLED once all the process components corresponding to the plan items are executed successfully.

The high level relationships between the order and plan entities are shown in the following figure:

**Figure 34: Order, Plan, Plan Fragment and Process Component**

Plan item, Plan fragment, and Process Component are inter-related concepts.

Here is the brief description of each of these concepts:

- *Plan Item* is one of the steps in a plan that must be executed to reach the goal of fulfilling an order line, and eventually the order. The plan item is configured with the name of the Process Component, which must be invoked to fulfill a product. The name of the Process Component is provided by Fulfillment Order
Management AOPD during plan development, and gathered from the Product Catalogue using the name of the product.

- **Plan Fragment** is the model definition of a Process Component, which fulfills a particular product. Products are linked to plan fragments in the Product Catalogue. The name of a plan fragment is the same as the name of the Process Component that it describes.
- **Process Component** is the physical implementation of the tasks required to fulfill a product. It is described by a plan fragment and invoked as a plan item step in a plan.
Automated Order Plan Development

In TIBCO® Fulfillment Order Management context, an order and the corresponding execution plan respectively represent the following:
• What (goal) to fulfill/achieve, and
• How to fulfill/achieve that particular goal.

Automated Order Plan Development (AOPD) is the core component of TIBCO Fulfillment Order Management, which transforms the What part, i.e. order into How - the execution plan.

Automated Order Plan Development (AOPD) receives orders from Orchestrator. AOPD decomposes an order into a plan. The plan is used to fulfill the corresponding order.

AOPD takes into account the specifications of the required products and the products currently provided to a customer.

AOPD uses a Product Catalog to decompose the orders. Typically, the Product Catalog can be TIBCO® Fulfillment Catalog (FC).

Figure 35: Automated Order Plan Development

When an order is received, its order lines are decomposed using a Product model.

A Product Model contains Bundles and Products Services. A Product model also contains concepts such as sequencing and dependencies.

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 services, products, and resources required. For example, an order line may contain a bundle, which may be comprised 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.

An incoming order to TIBCO® Fulfillment Order Management consists of one to many order lines, with each line requesting a product or service to be fulfilled. Orchestrator sends the order received from the OMS component to AOPD for the execution plan generation. AOPD component has the active reference of the product and the customer catalog.

AOPD generates the execution plan by applying various rules on the incoming order against the product, customer catalogs and the optional inventory for the customer coming along with the order in execution plan generation request. See figure Figure 36: Plan Generation by AOPD Inputs and Outputs on page 63

1. Plan development performance is related to the size of the catalog and order being decomposed. Where possible, the size of both should be reduced to improve performance.
2. Plan Fragments should not be modeled that do not do anything at execution time. Only plan items that do useful work should go into the plan.
3. Milestones and overlapping sequencing should be used instead of empty plan items for dependencies.
For a product requested in an order line, AOPD creates a plan item and assigns the plan fragment corresponding to the action specified in the order line from the Product Catalog. All such plan items are added into the execution plan. The plan is further optimized by applying rules for features such as Affinity, Single Use, and so on. On completion, the generated execution plan is sent back to the Fulfillment Order Management Orchestrator for the order orchestration process.

Figure 36: Plan Generation by AOPD Inputs and Outputs

The typical order fulfillment flow in TIBCO® Fulfillment Order Management is represented by the following sequence diagram:
Figure 37: Plan Generation and Execution Sequence

Error Messages and Handling

AOPD provides error handling and returns meaningful responses in case of errors detected during the stages of plan development. During plan generation if any errors are reported, AOPD will stop the plan generation immediately and will return an error response. The error handling also takes care of circular dependency in the plan and returns an appropriate response.
Manual Order Plan Development

Manual Order Plan Development (MOPD) gives users the control to modify the AOPD developed plan. In MOPD, plan development happens through Fulfillment Order Management User Interface under predefined logical boundaries.

1. Order is submitted using JMS or HTTP.
2. OMS picks up the order request and saves the record into the database.
3. Router Component of OMS routes the order request to Orchestrator using JMS or can invoke the orchestrator service directly (in Process).
4. Orchestrator sends the order Request to AOPD/JMS/third party for draft plan generation.
5. Orchestrator receives the draft plan and halts the further execution until final plan is not received.
6. User searches for the manual order which is supposed to be edited manually.
7. User retrieves the order details for manual order and gets order details in OMSUI.
8. User traverse to the draft plan (showing MOPD plan similar to execution Plan) which is saved earlier.
9. User can see the draft plan in user interface.
10. User instructs to get draft plan in UI. This indicates that user wants to edit the plan now.
11. User can edit the plan now by adding/modifying/deleting Plan-Item, Milestones and Dependencies in the plan developed by AOPD.
12. User edits the draft plan through FOM-UI and instructs to save the plan in database.
13. Plan is saved in database.
14. User instructs to execute this plan.
15. Plan is validated using the validation framework provided by server.
16. If plan is valid then Orchestrator is notified with the final plan and it starts execution of the plan.
17. If plan is not valid then application returns back specifying that plan is not valid and plan can be corrected.

Server and Client Side Validations

A plan manually modified after being instructed to go into execution by the user does not go for AOPD rule validation. Hence, it is user’s responsibility to modify the plan correctly. Some of the plan level validation by default happen at the client side which is covered in the TIBCO Fulfilment Order Management Administration guide.
Order Capture System

Order Capture System (OCS) is a new component in TIBCO Fulfillment Suite to create, manage, and submit TIBCO Fulfillment Orchestration orders.

This component is a web application which you can use to do the following:
- Select subscribers
- Browse validated products, services, or bundles from TIBCO Fulfillment Catalog
- Create orders for selected subscribers and submit them to TIBCO Fulfillment Order Management

To construct an order, OCS requests information from the following systems:
- TIBCO Fulfillment Catalog, which provides product definitions for subscribers
- A subscriber inventory, which provides subscriber details (such as names, addresses, IDs, and segments)
- Offer and price engine (OPE), which provides the eligible products for a specific subscriber, validates the order, and provides the price of what will be provisioned.

OCS then submits the orders to TIBCO Fulfillment Order Management.

The following figure illustrates the interconnection between OCS and the Fulfillment Orchestration sub-systems:

![Order Capture System Architecture](image)

**Figure 38: Order Capture System Architecture**

OCS is an optional component that is a part of Fulfillment Orchestration and shipped within TIBCO Fulfillment Order Management.

**Order Capture System Architecture**

The OCS application is a web application hosted in a Tomcat server. The Tomcat hosted application accesses external systems to browse the product catalog and create, submit, and amend orders in TIBCO Fulfillment Order Management.

The web application is composed of 2 parts:
- A javascript part running in the browser called OCS UI, which accesses the Tomcat hosted application...
• A Tomcat hosted java application responsible for accessing the storage and external systems called OCS Server

Order Capture System User Interface

OCS UI is a tool for browsing products, services, or bundles in the TIBCO Fulfillment Catalog, creating new orders for selected subscribers, and submitting them to TIBCO Fulfillment Order Management.

You can use OCS UI to do the following:
• Select a subscriber.
• View subscriber details.
• Browse the catalog.
• Select items from the list of products.
• Add items (products or bundles) to the shopping cart.
• Review the shopping cart.
• Edit selected items.
• Submit an order to TIBCO Fulfillment Orchestration (such as a new order, an amended order, an updated order, or a ceased order).

Order Capture System External Systems

OCS is a web based application that is used to browse products and bundles, and submit, amend, or cancel orders to TIBCO Fulfillment Orchestration Suite. To construct an order, the server part of OCS connects to the multiple external systems and sends them requests.

The nominal path to construct and submit an order is as follows:

1. Search for and select a subscriber.

When you search for a specific subscriber, OCS connects to a subscriber inventory service to retrieve a list of subscribers matching your search criteria. You can then select a specific subscriber from this list.

2. Configure the products or bundles for the selected subscriber.

This involves three different external systems:
• TIBCO Fulfillment Catalog, which provides the definitions of products and bundles
• An eligibility service, which provides the eligible products or bundles for the selected subscriber
• A pricing service, which provides the prices of the products or bundles.

Within TIBCO Fulfillment Order Management, both eligibility and pricing are grouped into TIBCO Offer and Price Engine (OPE).

3. After the products and bundles are configured, you can submit the order to TIBCO Fulfillment Order Management.

OCS uses a database to store data.

| Subscriber Inventory | OCS uses this service to select subscribers based on a specific search string. For more information on the WSDL defining this service, see TIBCO Fulfillment Order Management Web Services guide. This interface is not implemented in TIBCO Fulfillment Suite. Instead, it must be implemented by projects that have access to a CRM. For demonstration purposes, OCS provides a hosted implementation, where you can create demo subscribers and subscriber details. This implementation is on by default and can be toggled on or off from the OCS configuration file. For more information, see the TIBCO® Fulfillment Order Management Administration guide. |

TIBCO® Fulfillment Order Management Concepts and Architecture
TIBCO Fulfillment Catalog

TIBCO Fulfillment Catalog defines and manages life cycles of commercial and technical offerings. It also contains the metamodel for the services and products you can browse and choose from. Characteristics, such as name and type of attributes of the services and products, are configured by the catalog. OCS accesses the TIBCO Fulfillment Catalog segment and product information through an offline export. For more information, see the TIBCO® Fulfillment Catalog User’s Guide.

Offer and Price

In the TIBCO Fulfillment Suite, this interface is implemented by the TIBCO Offer and Price Engine. This engine determines which products are eligible for the subscriber, based on the current shopping cart and the subscriber’s purchased products or services. This engine also determines pricing information based on the product, service IDs, and characteristics you are viewing or selecting. OCS accesses this engine through the WSDL located in $AF_HOME/schemas/wsdl/http/OfferPriceServiceHTTP.wsdl. You can implement this WSDL to provide your own rendition of this service in place of OPE implementation.

TIBCO Fulfillment Order Management

OCS uses TIBCO Fulfillment Order Management to browse products, submit new orders, and cancel or amend existing orders. OCS accesses TIBCO Fulfillment Order Management through the Order Management web service. The WSDL used is $AF_HOME/schemas/wsdl/http/OrderServiceHTTP.wsdl. For more information on how to configure the web service details, see the TIBCO® Fulfillment Order Management Administration guide.

Database

OCS uses a database to retain information gathered from external systems. For example, it retains characteristic values for products that a subscriber has selected until the order is submitted to TIBCO Fulfillment Order Management. Once submitted, the order is discarded from OCS and the master copy is stored within TIBCO Fulfillment Order Management. Therefore, the database can be considered as a temporary cache for OCS. Data within this database can be safely discarded; only uncommitted work will be lost.
Offer and Price Engine

Offer and Price Engine (OPE) is a component of TIBCO Fulfillment Order Management that replaces Offer Configuration and Validation (OCV). The Offer and Price Engine (OPE) provides support for determining product eligibility for the subscriber, and pricing for the offer. The existing functionality of OCV, such as get offer and validate offer and other customer calls, are subsumed and enhanced by the new implementation of OPE. In addition, the engine provides support to get the product information present for offerings present in the system and price information.

For more information on these services, see the OPE Web Services sections in the *TIBCO Fulfillment Order Management Web Services* guide.

**Evaluate Product Eligibility**
This functionality evaluates the products for the subscriber based off the eligibility rules applied to all products, the given subscriber’s order. This is done with the Get Offer service provided by OPE.

**Offer Validation**
Previously, all the orders coming from TIBCO Fulfillment Order Management was optionally evaluated for validity using the Offer Configuration and Validation (OCV) engine. Now, TIBCO Fulfillment Order Management routes these orders to OPE for offer validation. This is done with the Validate Offer service provided by OPE.

**Product Pricing**
This functionality gives the pricing of a particular order based on the product catalog and the given set of products in conjunction with segments. The engine accepts multiple combinations of products and multiple order requests and generates corresponding prices in response of these combinations. This is done with the Get Prices service provide by OPE.

**Product Information**
This functionality gives the product ID, relationships, attributes, and so on when requesting for product information. It gets the information present in the product model for the product.

**Pricing Information**
This functionality fetches the pricing for all products, including individual products and the total of all products within the offer based on the product ID list specified in the request.
Process Components

This component can be implemented in a variety of technologies depending on the required functionality. Typically this will be TIBCO Fulfillment Provisioning (FP).

All Process Components must adhere to the service contract specified by Orchestrator in order to be considered a valid Process Component. This means implementation of three types of request events and providing two types of responses. These components should be accessed via standard JMS event interface wherever possible. Individual Process Components should be stand-alone components which allows for changing the Process Component collection dynamically in real-time without requiring an order management outage.

All external component integrations will be through the Process Components component. These integrations will generally either be service calls to perform automated tasks, or callouts to start a manual workflow.

The integration pattern for automated service calls will take the form of Process Components sending out an event to an adapter layer which will include relevant order and order line data as requested from the cache. This adapter layer will then transform the data into the format required by the back-end service and then invoke that service. When it has completed it will send a response back to Process Components to complete the step in the flow.

Process Components are responsible for the following:

1. Implement the tasks required to fulfill a particular product on an order. This may be done in any JMS-enabled technology provided the interface specification for a Process Component is satisfied.
2. Accept requests from Orchestrator to start executing a new fulfillment process.
3. Request required information from the cache that is required as part of a fulfillment process.
4. Execute the required business process for fulfilling a particular product that a customer may order. This may take the form of invoking back-end service calls, business process management, or manual tasks as appropriate for the implementing technology.
5. Update information in the cache as part of the fulfillment process if required.
6. Return the execution results to Orchestrator.
7. Suspend execution of a fulfillment process when requested by Orchestrator. Respond to the suspend request by returning to Orchestrator confirmation of a successful suspend or normal completion of the fulfillment process.
8. Resume execution of a suspended fulfillment process when requested by Orchestrator through to completion from the point of suspension.
9. Cancel execution of a suspended fulfillment process when requested by Orchestrator. The tasks following the point of suspension are not executed. Cancellation may require rollback of previously completed tasks in the fulfillment process, or a simple abort of the execution process.
Feasibility Provider

This component can be implemented in a variety of technologies depending on the required functionality provided it meets the interface specification requirements for a Feasibility Provider.

Feasibility checking is an optional step in the order lifecycle that analyzes the order to determine if it can be fulfilled. Feasibility checking may involve validating the order contains the required products, physical network capacity checking, or inventory stock level check. The Feasibility Provider is a customer-implemented component because feasibility checking is highly customized to the requirements of a particular customer. OPE may optionally be used as a component of an overall feasibility check to determine order validity.

It is accessed through a JMS event interface.
Jeopardy Management

Jeopardy is implemented as a tightly coupled component in FOM along with existing features like OMS, AOPD, and Orchestrator. Notifications to Jeopardy will be sent as low level API calls or through in-process communication, which is similar to other FOM component communication. The advantages of 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 FOM 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 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 Fulfillment Order Management](image)

**Figure 39: Architecture of Fulfillment Order Management**

The following diagram is a representation of the Plan and Order Execution using Jeopardy:
In-memory (Cache) Plan and Process Component Repository

The performance of Jeopardy has been improved by an in-memory implementation of plan and process repository. The plan repository is managed as a component in the Orchestrator finite state machine. If the server is abruptly terminated, both the plan, and the process component repositories will be backed by the database that acts as a secondary level cache to safeguard the changes.
The in-memory cache provides local cache access and extreme performance. It uses the database as the persistence system to persist information across server startups. The Hibernate feature continues to use active spaces as a silent second level cache for any spillovers from the main memory cache.

The application uses Orchestrator state machine’s state as the primary in memory caching system. It also uses an existing Orchestrator caching and spill over mechanism to handle additional caching of data running out of memory. This practically sets no limit on the in-memory cache size.

**Datastore Compatibility**

A new datastore (secondary cache) has been added to the system. The application can now be deployed with in-memory cache along with the database as a secondary cache.

The configuration to setup the database as a datastore, can be configured in `Configvalues_JEOMS.xml` file as follows:

```xml
<ConfValue description="Operation Data Store Type" name="Operational Data Store Type" propname="com.tibco.jm.config.odstoreType" sinceVersion="2.0" visibility="Advanced">
  <ConfString default="cache" value="cache"/>
</ConfValue>
```

With the colocated architectural changes, the application cannot run in standalone mode with Cache as a datastore. This restriction is added to keep the orchestrator component as a master component, to control the jeopardy notification functionalities.

**Database: Secondary Cache Option**

Database is implemented as a second-level cache for durability purposes. The database is configured as the application second level cache in `Configvalues_JEOMS.xml` file as follows:

```xml
<ConfValue description="Operation Data Store Type" name="Operational Data Store Type" propname="com.tibco.jm.config.odstoreType" sinceVersion="2.0" visibility="Advanced">
  <ConfString default="cache" value="cache"/>
</ConfValue>
```

Cache is the default value and the only configuration supported in colocated mode.

Hibernate will use active spaces as a dialect for silently maintaining in memory spill overs.
**Batch Notification and Messages**

Following are the notifications sent by JeOMS:

1. **JMS notification**: To save jeopardy messages and risk region in the database when a jeopardy condition is detected for a plan or planItem.
2. **JMS notification**: To report an alert in dashboard when a jeopardy condition is detected for a plan or planItem.
3. **Database notification**: To update cache.

In case of existing architecture, notifications are processed and saved into the database. Processing is sequential and not in the batch. Each notification has a dedicated connection from connection pool. With the new architecture, there are options to synchronously process the JMS notifications sequentially, and also to execute the notification in the batch.

**Batch Event Processing**

Events are consumed by state machine and notifications generated by the state machines that are posted to the Batch process. Events update the cache repositories for any status updates, or event notifications related to jeopardy detection. The following is the sequence of activities involved:

1. State Machine receives the events from JMS.
2. Events are consumed by State machine.
3. State machine generates database notifications and JMS notifications that are posted to Batch Processes.
Router

The Content-based router in OMS allows to route the order to the correct destination based on the contents of the order message.

Content-based routing routes messages are based on the actual content of the message itself, rather than by a destination specified by the message. Content-based routing works by opening a message and applying a set of rules to its content to determine the destination of a message. By freeing the sending application from the need to know anything about where an order should be routed for fulfillment, content-based routing provides a high degree of flexibility to configure multiple type of Orchestration engine. For details, see TIBCO® Fulfillment Order Management Administration Guide.
Key Functionality

Fulfillment Order Management provides the following functionalities:

• **Fulfillment Order Management Configurator Graphical User Interface (GUI)** - configures the various settings for Fulfillment Order Management system using GUI mode.

• **Attribute-based Decomposition** - filters execution plan depending on the orderline attributes of the products ordered.

• **Affinity** - allows different plan fragment types to be grouped together on the same order.

• **Order Amendments** - Order Amendment now handled through new Order Management Server and Fulfillment Order Management Orchestration engine.

• **Offline Catalog** - enables the Fulfillment Order Management application to fulfill the orders and reduces the dependency on the Fulfillment Catalog to be Online.

• **Centralized Logging** - provides the logging and reporting capability for analyzing the data and generate meaningful reports.

• **Dependent and Sibling Product** - enables grouping products on requests thereby allowing for sibling products and their children products to be sent to a dependent product. The ability is built in the Product Model to indicate that a product is dependent on its peer or peer’s hierarchy.

• **Shared Attributes** - used when two Products (parent to child and sibling) share the same attribute and its corresponding value, and an update in the value of one needs to be reflected in the other.

• **Custom Action Support** - allows you to submit custom actions.