



TIBCO® Offer and Price Engine

User Guide

Version 6.0.0
March 2023



Contents

Contents	2
About This Product	5
Features	6
Recommendation Engine	6
Architecture of FOS Recommendation Engine	6
Offer Eligibility and Validation	15
Decomposition	18
Product Integrity	18
Segment Eligibility	18
Data Validations	20
Get Offer Compatibilities	23
Validate Offer Compatibilities	25
Group and Record Constraints	25
Offer Rule Extension	26
Eligibility Rule	28
Ineligibility Rule	29
Incompatibility Rule	30
Offer Price Processing	31
Get Prices Determinations and Calculations	36
Offer Discount Processing	37
Java Extension	41
Search	45
Misspelled Search Terms	46
Index	46
Browse	49
Product Id and Product Id Ext.	49

Implementation of LDAP	50
API Monitoring	55
WebClient Configuration	60
Data Models	61
Model Loading Process	61
Online Model Loading	62
Catalog Web Service Model Loading	64
Offline Model Loading by using Catalog Client Service	65
Shopping Cart Service	68
Authorization Service	69
Create User	69
Update User	71
Get User	72
Delete User	73
Generating an authorization token	73
Generating an authentication token through the REST endpoints	77
Use Cases	80
Testing Product Eligibility Scenarios	80
Scenario 1 - Testing with the SegmentsCompatibleWith Filter	80
Scenario 2 - Testing with the SegmentCompatibleWith RecordType Filter	81
Scenario 3 - Testing with the Flag ReturnBundleOfferings Enabled	81
Scenario 4 - Testing with the Focus Filter with Different Product Types and autoprovision=true	82
Scenario 5 - Testing with Maximum Number of Products in a Group is Reached	83
Testing Product Validation Scenarios	84
Scenario 1 - Testing with Maximum Number of Products in a Group is Reached	84
Scenario 2 - Testing with Minimum Number of Products in a Group is Not Reached	85
Scenario 3 - Testing with the Compatibility Relationship	85
Scenario 4 - Testing with Multiple Combinations	86

Scenario 5 - Testing with Required UDF Input	87
Scenario 6 - Testing with ProductRequiredFor with a Group and One Element Passed	87
Scenario 7 - Testing ProductRequiredFor with No Group and One Element Not Passed	88
Scenario 8 - Testing LinkDefinitions with Restrictions with Maximum Number of Products	88
Scenario 9 - Testing LinkDefinitions with Restrictions with Maximum Number of Products in a Group	89
Scenario 10 - Testing with Multiple Minimum Number of Products in a Group Not Reached	90
Scenario 11 - Testing with Products of the Same Name but Different Parents	91
Best Practices	93
Eligibility for Offer and Price Engine	93
Service Response Time	93
Model Processing Time Verification for Offline vs Online Model Loading	94
Tuning Parameters	95
Garbage Collection Recommendations	98
TIBCO Documentation and Support Services	101
Legal and Third-Party Notices	103

About This Product

TIBCO® Offer and Price Engine is a cloud-native, in-memory omnichannel server of offers and prices for digital service providers. It answers requests from a digital service provider's customer-facing channels for offers and prices, subject to business rules such as customer eligibility and product compatibility.

TIBCO Offer and Price Engine is the next generation of, and partially replaces, TIBCO® Fulfillment Order Management. To better align the direction of TIBCO Fulfillment Order Management with market demand, the product's capabilities have been reorganized into two new products:

- TIBCO® Order Management
- TIBCO® Offer and Price Engine

Customers who are currently on maintenance for TIBCO Fulfillment Order Management are entitled to upgrade to both TIBCO Order Management and TIBCO Offer and Price Engine. TIBCO will continue to support TIBCO Fulfillment Order Management, and there is currently no plan to retire TIBCO Fulfillment Order Management. New capabilities will be developed in TIBCO Order Management and TIBCO Offer and Price Engine.

Features

The following topics provide information about TIBCO Offer and Price Engine features.

Recommendation Engine

A recommendation engine is a type of mechanism that uses machine learning algorithms and suggests the most relevant products to a particular user or client. It works on a principle to find patterns in the behavior data of users that are collected in various ways.

Based on customer's purchase history, most relevant offers are recommended. Also, based on customer's catalog, similar products are recommended. This recommendation can be a comparably higher-end product than the one is in possession by the customer. This type of recommendation is called as upselling.

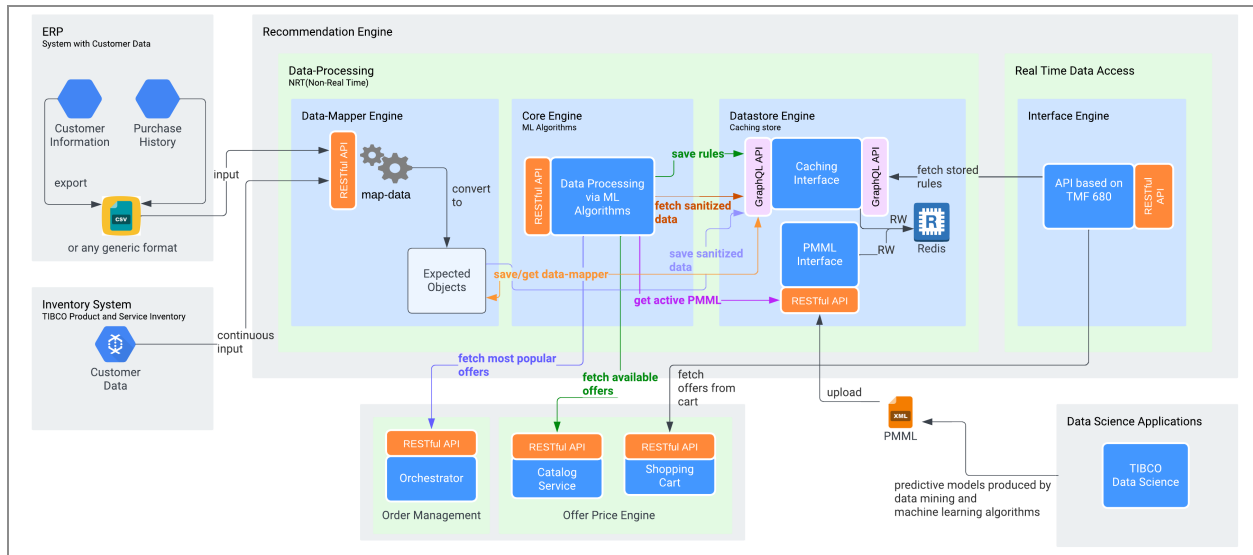
Similarly, there can be recommendation of products those fulfill additional complementary needs compared to the original items. This type of recommendation is called as cross-selling.

There are two types of recommendation systems:

- **Collaborative filtering:** Based on similar customer's past data, recommendations are given to the other customers. Core of collaborative filtering is based on nearest neighbor's search.
- **Content based filtering:** The Content-based approach uses additional information about users or items. This filtering method uses item features to recommend other items similar to what the user likes and also based on their previous actions or explicit feedback.

Architecture of FOS Recommendation Engine

To provide a recommendation to a customer, certain data is required based on which the recommendations are predicted. The data needs to be fed to the **Data-Processing** unit.



Following are the various layers of the **Data-Processing** unit:

- [Data-Mapper Engine](#)
- [Core Engine](#)
- [Datastore Engine](#)
- [Interface Engine](#)

Data-Mapper Engine

The first layer of the data processing unit is **Data-Mapper Engine**. Data such as customer information, purchase history is provided through ERP system or REST APIs to this engine in CSV or JSON format.

The following components are part of the **Data-Mapper Engine**.

- **Data Mapper:** Submit the customer's transactional data such as product Id, customer Id, transaction Id, or sample data to do the mapping.
- **Data Publisher:** Publish the customer's transactional data by providing the data mapper details created through the Data Mapper API.

Data Mapper APIs

Method: HTTP POST

Endpoint: `http://<host_address>:<port_address>/v1/jsonDataMapper` or `http://<host_address>:<port_address>/v1/csvDataMapper`

Data Mapper Parameters

Parameter	Cardinality	Description
productIdMapper	Mandatory	The product Id of the mapper.
customerIdMapper	Mandatory	The customer Id of the mapper.
transactionIdMapper	Mandatory	The transaction Id of the mapper.
dataMapperName	Mandatory	This is the name of the data mapper that is created once the API is executed. You can provide a desired name here.
Request body	Mandatory	Provide a sample data here.

Sample:

```
{
  "id": "PO_TV",
  "href": "https://host:port/productInventoryManagement/v4/product/g265-
tf85",
  "description": "product description",
  "isBundle": false,
  "isCustomerVisible": true,
  "name": "Voice Over IP Basic instance for Jean",
  "productSerialNumber": "N/A",
  "orderDate": "2017-11-01T09:37:29.961Z",
  "startDate": "2017-11-01T09:37:29.961Z",
  "status": "active",
  "@baseType": "Product",
  "@type": "Product",
  "@schemaLocation": "https://host:port/standardProduct.json",
  "relatedParty": [
    {
      "id": "Cust_01",
      "href": "https...",
      "name": "Jean",
      "role": "User",
      "@referredType": "Individual"
    }
  ]
}
```

Data Publisher APIs

Method: HTTP POST

Endpoint: `http://<host_address>:<port_address>/v1/publishJsonData` or
`http://<host_address>:<port_address>/v1/publishCsvData`

Data Publisher Parameters

Parameter	Cardinality	Description
dataMapperName	Mandatory	Name of the data mapper that you have created through Data Mapper API.
dataMapperVersion	Mandatory	Version of the data mapper that you have created through Data Mapper API.
Request body	Mandatory	Provide a sample data here.

Sample:

Sample as below:

```
{
  "id": "PO_TV",
  "href": "https://host:port/productInventoryManagement/v4/product/g265-
tf85",
  "description": "product description",
  "isBundle": false,
  "isCustomerVisible": true,
  "name": "Voice Over IP Basic instance for Jean",
  "productSerialNumber": "N/A",
  "orderDate": "2017-11-01T09:37:29.961Z",
  "startDate": "2017-11-01T09:37:29.961Z",
  "status": "active",
  "@baseType": "Product",
  "@type": "Product",
  "@schemaLocation": "https://host:port/standardProduct.json",
  "relatedParty": [
    {
      "id": "Cust_01",
      "href": "https...",
      "name": "Jean",
      "role": "User",
      "@referredType": "Individual"
    }
  ]
}
```

```

    }
  ]
}
```

Datastore Engine

Data store mainly has GraphQL APIs to perform data operations in Redis. All other services' Data Mapper, Recommendation Engine Core, and Recommendation Engine APIs call Data Store Engine to save, retrieve, or delete different data.

Example: Recommendation Data Mapper calls Data Store Engine to save data mapper and also saves sanitized customer's transactional data. Recommendation Engine Core calls GraphQL APIs to fetch customer data and saves calculated association rules in Redis.

Apart from GraphQL APIs for CRUD operations on data, Data Store Engine has REST APIs to perform CRUD operations on PMML.

Predictive Model Markup Language (PMML) Model

This functions in a similar to the algorithms that work in python. The PMML 4.4 version is supported and the Naive Bayes and k-Nearest Neighbors classifications are used as of now. You need to create and train PMML model by using data science tool. For more information on PMML, see <https://dmg.org/pmml/v4-4-1/GeneralStructure.html>

i Note: If PMML is present in your system, then it has the priority over core algorithms. If PMML is not present, then only the core algorithms are applied.

Data Store Engine PMML Controller APIs

Make specific version of PMML Model file active

Method: HTTP PUT

Endpoint: `http://<host_address>:<port_address>/v1/pmml/{version}`

Parameter	Cardinality	Description
Version	Mandatory	Version of the pmml file.

Delete specific version of PMML Model file

Method: HTTP DELETE

Endpoint: `http://<host_address>:<port_address>/v1/pmml/{version}`

Parameter	Cardinality	Description
Version	Mandatory	Version of the pmml file.

Download All PMML Model Files for Tenant ID

Method: HTTP GET

Endpoint: `http://<host_address>:<port_address>/v1/pmml`

Upload PMML file

Method: HTTP POST

Endpoint: `http://<host_address>:<port_address>/v1/pmml`

Request body: Browse the pmml file that you want to upload.

Get metadata of All PMML Model Files for Tenant ID

Method: HTTP GET

Endpoint: `http://<host_address>:<port_address>/v1/pmml/metadata`

Download Active PMML Model Files for Tenant ID

Method: HTTP GET

Endpoint: `http://<host_address>:<port_address>/v1/pmml/active`

Core Engine

This is a python based micro service. By default, Recommendation Core Engine has Apriori and Cosine Similarity algorithms. The core engine applies these two algorithms if there is no active PMML present in the system.

The Apriori algorithm runs on the customers' transactions and generates association rules for the products. In contrast, the Cosine Similarity algorithm runs on the customers' catalog and generates similar products.

If active PMML is present in the system, core engine processes PMML and generates customer predictions for customers' transactions.

Recommendation Engine API generates recommendations based on association rules, similar products, and customer predictions.

This service has the following rule APIs:

- [Find Association Rules](#)
- [Calculate Product's Similarity](#)

i Note: The Find Association Rule and Calculate Product's Similarity APIs run in the background asynchronously at the schedule time created by the user.

Find Association Rules

Method: HTTP GET

Endpoint: `http://<host_address>:<port_address>/v1/findAssociationRules/sync`

The `tenantid` field is mandatory.

Calculate Product's Similarity

Method: HTTP GET

Endpoint: `http://<host_address>:<port_address>/v1/calculateProductsSimilarity/sync`

The `tenantid` field is mandatory.

i Note: The responses from these sync APIs are not stored in a data store, this is just for viewing.

Schedule Job

The actual customer's data is huge, where the algorithms are applied to provide the recommendations. Hence this action cannot be synchronous. A specific time window has to be provided for the recommendation system to work on. A start job and stop job can determine the required time window.

- [Schedule Job Start](#)
- [Schedule Job Stop](#)

Schedule Job Start

You can schedule the job start time for each tenant.

Method: HTTP POST

Endpoint: `http://<host_address>:<port_address>/v1/schedule/job/start`

The `tenantid` and `cron_schedule` fields are mandatory.

The `tenantid` is the ID of the tenant for which job needs to be scheduled.

The `cron_schedule` is a Linux command used for scheduling tasks to be executed sometime in the future. This is normally used to schedule a job that is executed periodically. Example: to send out a notice every morning

This is used for updating jobs that are already scheduled.

For example, if a Job for tenant TIBCO is scheduled to run at 1 PM daily and later the user wants to change the schedule to run it at 2 PM daily, then POST API is called. Here, you have to add the same tenant ID and different `cron_schedule` for the tenant.

Schedule Job Stop

When you want to stop the cron job of the already scheduled tenant, then use this API.

Method: HTTP POST

Endpoint: http://<host_address>:<port_address>/v1/schedule/job/stop

The tenantid field is mandatory.

Recommendation Engine API

- [List or find query product recommendation objects](#)
- [Create query product recommendation](#)
- [List or find query product recommendation objects](#)

List or find query product recommendation objects

Method: HTTP GET

Endpoint: http://<host_address>:<port_address>/v1/queryProductRecommendation

Parameter	Cardinality	Description
recommendationPageNo	Mandatory	Recommendation page number
numberOfRecommendationsPerPage	Mandatory	Number of recommendations per page

Create query product recommendation

Method: HTTP POST

Endpoint: http://<host_address>:<port_address>/v1/queryProductRecommendation

Parameter	Cardinality	Description
noOfSimilarProducts	Mandatory	Number of similar

Parameter	Cardinality	Description
		products
Request body	Mandatory	Provide a sample data here



Note: If the `instantSyncRecommendation` flag is true, the recommendation is saved and also added in the response. If it is false, recommendation is only saved.

List or find query product recommendation objects

Method: HTTP GET

Endpoint: `http://<host_address>:<port_address>/v1/queryProductRecommendation/{Id}`

Parameter	Cardinality	Description
id	Mandatory	Id of the query product recommendation

Offer Eligibility and Validation

TIBCO Offer and Price Engine uses the same offline models schema currently being used by the TIBCO Product and Service Catalog engine. TIBCO Offer and Price Engine uses product models to get offers configured in the product catalog.

In the TIBCO OPE architecture, the top-level product has no auto-provisioned parents associated with it. All such products are considered as a product offering. Using the class and subclass features of the product model, they can be classified into logical types.

Online integration with TIBCO Product and Service Catalog is also possible using the publish catalog method.

TIBCO Offer and Price Engine integrates with offline models that are published with compliant schema without TIBCO Product and Service Catalog.

TIBCO Offer and Price Engine uses product models (optionally `OfferMappingFile` to link `OfferIds`) to get offers configured.

Relationships

TIBCO Offer and Price Engine makes use of the following relationships for offer generation and validation and price determination:

Relationship	Description
Product Comprised Of	This relationship is used to define the dependencies between the products, which lead to the decomposition of parent and child products.
Characteristic	This relationship is used to associate the characteristic entities with a product. In addition to characteristics that are fulfillment related, some internal characteristics are used by the product for internal logic. These are related to error handling.
Compatible Product	This relationship defines that two products can exist in an order or customer image.
Incompatible Product	This relationship defines that two products cannot exist in an order or customer image.
Compatible Segment	This relationship defines that a product is compatible with the segment. A product can either be compatible, incompatible, or not related to the segment. If a product is defined as both compatible and incompatible in the model, it is considered a compatible product. Segments are grouped by segment type.
Incompatible Segment	This relationship defines that a product is incompatible with the segment type and name.
Class and Subclass	This relationship can be used to classify products into different logical types.
Product	This relationship is used to define the prerequisite between two products.

Relationship	Description
Required For	With this relationship, you can create offers without defining the sequence or dependency between products. These products are implicitly provisioned with the parent.
Category	This relationship defines that a product belongs to a category. A product can belong to no category, one category, or too many categories.

Record or Relationship Attributes

This engine uses the following record attributes and relationship attributes:

Attribute	Description
RECORD_STATUS	This record status characteristic indicates whether the record is active or not.
Start Date	This record level attribute indicates whether the product is an eligible product for the requested order date if it is after the start date.
End Date	This record level attribute indicates whether the product is an eligible product for the requested order date if it is before the end date.
Link Definitions	<p>This is a type field in the characteristics that defines the linking attributes, for example, subscriber ID or any other characteristic between two products. The LinkDefinitions are evaluated for the following relationships:</p> <ul style="list-style-type: none"> • ProductComprisedOf • ProductRequiredFor • IncompatibleProducts <p>If LinkDefinitions are provided for relationships, these values must match the values for UDFs for the corresponding products in the offer and the order request.</p>

Decomposition

Eligible products are decomposed using the ProductComprisedOf and the ProductRequiredFor relationship to get the auto-provisioned products. The product is not valid if any of its auto-provisioned products are invalid due to the product integrity, segment compatibility, category compatibility, and product validity.

Product Integrity

The product integrity determines if the product is eligible for the basket and the offer.

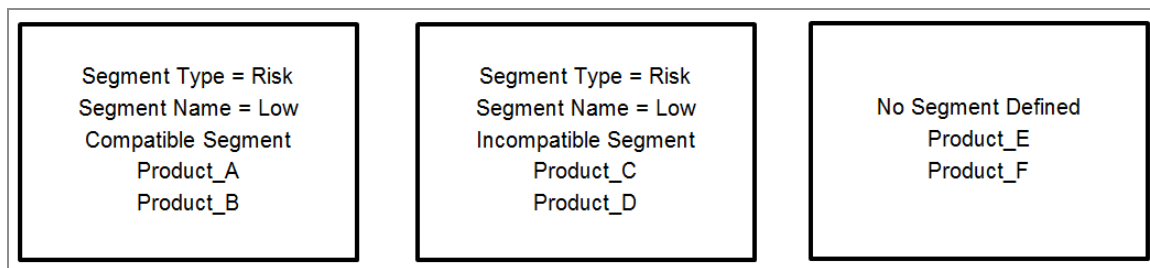
The following checks are required for determining product integrity:

- The product status must be ACTIVE.
- The order date for the offer must fall between the start date and the end date of the product. If not, the product is invalid.

Segment Eligibility

The initial list of eligible products is checked for segment eligibility with the specified segments. All products are considered to be compatible with the segment mentioned unless there is an incompatible relationship.

Scenario 1 - Same Segment Types with Different Segment Names



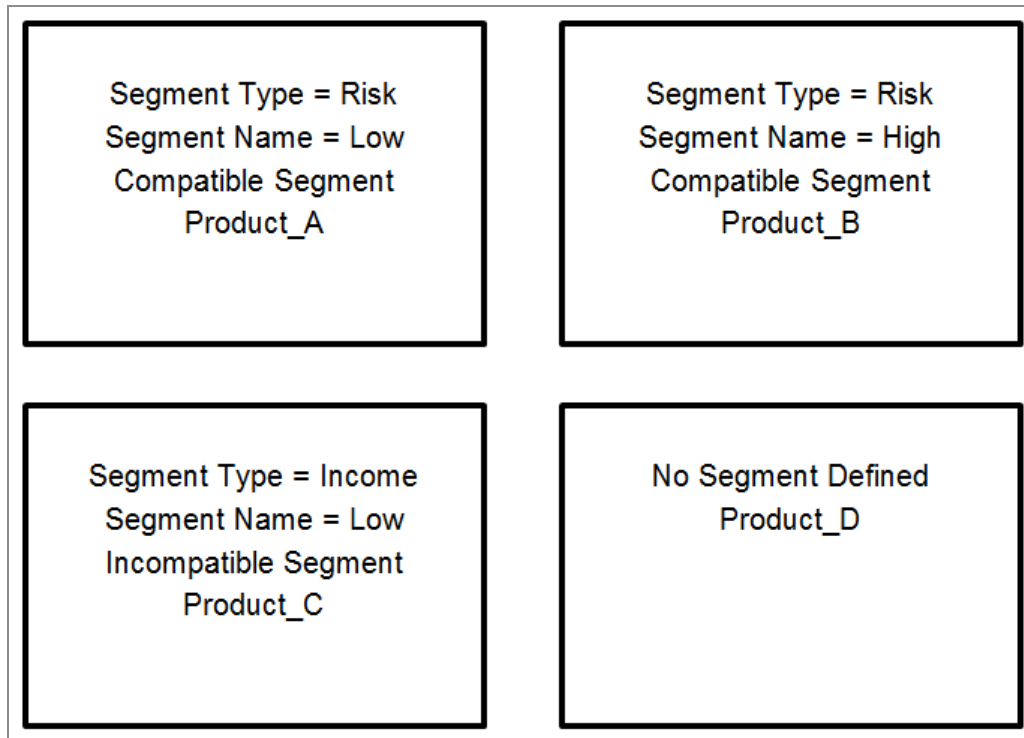
If the filter contains two segments: Segment Type = risk with Segment Name = Low and Segment Name = High, the list of eligible products would be Product_A, Product_B, Product_E, and Product_F.

Scenario 2 - Same Segment Types with Different Segment Names and Incompatible Relationship

Segment Type = Risk Segment Name = Low Compatible Segment Product_A	Segment Type = Risk Segment Name = Low Compatible Segment Product_B
Segment Type = Risk Segment Name = High Incompatible Segment Product_C	Segment Type = Risk Segment Name = High Incompatible Segment Product_D
No Segment Defined Product_E	No Segment Defined Product_F

If the filter contains 2 segments: Segment type = Risk with Segment Name = Low and Segment Name = High, the list of eligible products would be Product_A, Product_B, Product_E, and Product_F.

Scenario 3 - Different Segment Types



If the filter contains three segment types: Segment Type = Risk with Segment Name = Low, and Segment Name = High AND Segment Type = Income with Segment Name = Low, the eligible products are Product_A, Product_B, and Product_D.

Scenario 4 - Segment Types without Products

If the filter contains a segment type without any products attached to it, all products are considered to be compatible with the segment.

The product must be compatible with all segment types to be considered as an eligible product.

Data Validations

It refers to data and UDF validations of the offer requests and order requests. The offer is either validated or invalidated.

If the data is not validated within the request or input and active UDFs fail to adhere to their corresponding length, datatype, rangeValue, or regular expression, the product model gives an error message.

Property Name	Details
CheckRelevant0LUDFs	<p>Validation Flag:</p> <pre>com.tibco.af. ope.flags.chkrelevantoludfs</pre> <hr/> <p>Description:</p> <p>Validates UDFs attached to the product that are defined as input characteristics of the product. This functionality can be switched on using the configuration.</p>
CheckValid0LUDFs	<p>Validation Flag:</p> <pre>com.tibco.af. ope.flags.chkvalidoludfs</pre> <hr/> <p>Description:</p> <p>Validates mandatory characteristics attached to the product model that are found in the corresponding product instance as UDFs in the request. This functionality can be switched on using the configuration.</p>
CheckValidLinkUDFs	<p>Validation Flag:</p> <pre>com.tibco.af. ope.flags.chkvalidlinkudfs</pre> <hr/> <p>Description:</p> <p>Validates mandatory linking UDFs that are attached as UDFs to the product in the order request. This functionality can be switched on using the configuration.</p>

Property Name	Details
ValidateOrderLineUDFsDatatype	<p>Validation Flag:</p> <p>com.tibco.af. ope.flags.validateudfdatatype</p> <hr/> <p>Description:</p> <p>Validates the UDFs datatype. The datatype can be configured in the product model. The following are valid values: Currency, Digits, Date, Time, and Boolean</p>
ValidateOrderLineUDFRange	<p>Validation Flag:</p> <p>com.tibco.af. ope.flags.validateudfrange</p> <hr/> <p>Description:</p> <p>Validates that the orderline UDFs are within the range specified in the corresponding product model. This functionality can be switched on using the configuration.</p>
ValidateOrderLineUDFRegex	<p>Validation Flag:</p> <p>com.tibco.af. ope.flags.validateudfregex</p> <hr/> <p>Description:</p> <p>Validates that the orderline UDFs have the values per the regex. This functionality can be switched on using the configuration.</p>
ValidateProdDate	<p>Validation Flag:</p> <p>com.tibco.af. ope.flags.validateProdDate</p>

Property Name	Details
	<p>Description:</p> <p>Validates the products' start and end date. This functionality can be switched on using the configuration.</p>

With the `ValidateProdDate` flag, the dates required to be set must have the characteristic name as "StartDate/EndDate". The characteristic names are case-sensitive. The dates must be `xsd:datetime` format and the `valuetype` as "Input". The following is an example:

```
<characteristics>
  <name>EndDate</name>
  <description>Characteristic</description>
  <instanceMin>0</instanceMin>
  <instanceMax>0</instanceMax>
  <evaluationPriority></evaluationPriority>
  <actionID></actionID>
  <value>
    <type>Input</type>
    <rangeValue></rangeValue>
    <discreteValue>2011-01-01T00:00:00</discreteValue>
    <mandatoryValue>true</mandatoryValue>
  </value>
  ...
</characteristics>
```

Get Offer Compatibilities

The products in the offer must be compatible with the category, input field, record type, record subtype, the existing products in the offer, groups, and records to be eligible.

Category Compatibility

All the products in the offer must be compatible with all the categories specified.

Input Field Compatibility

This field is used to set the linking of relevant fields, which evaluates and returns eligible products. It checks for the link definitions from the following relationships: ProductComprisedOf, ProductRequiredFor, and (In)compatibleProducts.

Record Type and Record SubType Compatibility

All the products in the offer must be compatible with the record type and subtype specified. If there are multiple record types or record subtypes, the products must belong to at least one of the types and subtypes specified.

Product Compatibility

All the products in the offer must be compatible with all the products in the order. If no explicit incompatibility is defined, the product is compatible. Compatibility checks are done with all the auto-provisioned product chains in the orderline product and the offer product. If no compatible products exist, OPE checks for migrations in the product, as well as consequential products. All the eligibility rules must be applied in the case of migrations. If after migration the eligibility fails, add the product to the list of ineligible products. Each product is checked with only the products present in the order request for compatibility. Eligible products are not checked with each other for compatibility.

Optionally, single-use checks can be done so that the product exists only once for the customer in the order. The attribute `SingleUSE` is from the product model characteristics that is used for this functionality.

Group and Record Evaluation

The group and record constraints of ProductComprisedOf and ProductRequiredFor are evaluated for all the eligible products with the customer orders to ensure the eligible product for the customer orders do not violate the group constraints present in the product model. Eligible products are not checked with each other for group and record violations. For more information on the group and record attributes, see [Group and Record Constraints](#).

Validate Offer Compatibilities

The products in the offer must be validated with segment compatibility, product compatibility, group-level compatibility, and record-level compatibility.

Segment Compatibility

All the products in the offer must pass the segment compatibility validation. If any product does not pass the validation the offer is considered invalid.

Product Compatibility

All the products in the offer must be compatible with all the products in the order. If no explicit incompatibility is defined, the product is compatible. Compatibility checks with all the auto-provisioned product chains in the orderline product and the offer product.

Group and Record Validations

The group and record constraints of `ProductComprisedOf` and `ProductRequiredFor` are evaluated for all the eligible products with the customer order to ensure that the eligible product for the customer orders does not violate the group constraints present in the product model. For more information on the group and record attributes, see [Group and Record Constraints](#).

Group and Record Constraints

Group and record constraints check for minimum and maximum cardinality.

Group Attributes

If any product in the group causes violation for `groupMin` or `groupMax`, the offer is returned as ineligible for the get offer request and invalid for the validation request.

The following attributes are used for group evaluation:

- `groupNumber` specifies the unique identifier for the group. If no `groupNumber` is

specified, the product ID is considered as the group number so the product is in its group.

- `groupMin` specifies the minimum number of products that must be ordered for a bundle.
- `groupMax` specifies the maximum number of products that can be ordered in a bundle.

Record Attributes

If any product in the offer causes a record violation, the offer is returned as ineligible for the get offer request and invalid for the validation request

The following attributes are used for record evaluation:

- `RecordMin` specifies the minimum number of products that must be ordered for a bundle.
- `RecordMax` specifies the maximum number of products that can be ordered in a bundle.

Offer Rule Extension

TIBCO Offer and Price Engine uses Eligibility, Ineligibility, and Incompatibility rule models from the Offer and Price Designer Rule Editor to evaluate offers and products based on the conditions, characteristics, or segments set for each rule-type.

The rules are created using the Offer and Price Designer Rules Editor (available in the TIBCO Product and Service Catalog) which allows you to create rules using a graphical interface. Once the rules are created and approved they can be uploaded into OPE.

Online integration with TIBCO Product and Service Catalog is also possible using the publish catalog method. For more information about Eligibility, Ineligibility, and Incompatibility rules and on how to use the Rule Designer, see the *Product and Service Catalog Offer and Price Designer User's Guide*.

For more information on the rule models offline, see [Offline Model Loading](#).

Record or Relationship Attributes

This engine uses the following record attributes and relationship attributes:

Attribute	Description
RECORD_STATUS	Indicates whether or not the record is active.
RECORD_TYPE	Defines the type of rule. The valid values are Ineligible, Eligible, and Incompatible.
Start Date/End Date	Used to define the validity of a rule model within a certain time frame. (optional)

Filters

OPE uses the following filters to get the initial list of eligible products, which are then validated using the eligibility or ineligibility evaluation process:

Filter	Description
Focus	The focus filter considers the products that are present in the basket and order to get the initial list of eligible products. If the focus filter is present, the rest of the filters are ignored. Because the product in focus is already present with the customer, the optional products for the selected focus are used as the initial list of eligible products.
Promotions	The products in promotions filters are used to get the initial list of eligible products. If promotions are present, all other filters except focus are ignored to get the initial list of eligible products.
Segment	The segment filter provides a way to get a list of products belonging to the same segment. The products are related to the segment name and segment types. The initial list of eligible products contains a union of all the products, which includes the same and different segment types unless they have an incompatible relationship defined. Any products that do not have any compatible or incompatible relationships defined are considered to be compatible with that segment type.
Category	This filter provides a way to get the initial list of products belonging to the specified category.

Flags

The following are the flag names for get offer request and validate offer request, which must be enabled for rule extension cases. The default value for all the flags is false.

- `com.tibco.af.ope.extn.enableRuleBasedEligibilityEvaluation`
- `com.tibco.af.ope.extn.enableRuleBasedIneligibilityEvaluation`
- `com.tibco.af.ope.extn.enableRuleBasedIncompatibilityEvaluation`
- `com.tibco.af.ope.extn.missingRuleEntityDefault`

Eligibility Rule

Eligibility rules can be created using the Offer and Price Designer. For more information about how to use the Rule Designer, see the *Product and Service Catalog Offer and Price Designer User's Guide*.

Eligibility rules are based on product characteristics, segments, and inventory. A customer with specific characteristics can be eligible for a product if the product meets the segment requirements and if inventory is available. Therefore a user can have a combination of segments and inventory as a condition of this rule.

Get Offer

Eligible products computed by OPE are evaluated against the eligibility rules and are made eligible if any applicable rule's condition is satisfied.

Validate Offer

Eligible products that are present in the order lines are evaluated against eligibility rules and are made eligible if any applicable rule's condition is satisfied.

Segment Scenario

If a request contains two segment types `Customer Age IS 50 AND Country is India`, the response would be **eligible** for Product A.

Product A Characteristics COLOR=RED Characteristics RAM = 1024	Eligible Rule Product A Color is RED and RAM=1024 Condition with AND operator Segment Customer Age IS 50 Segment Country IS India
--	--

Ineligibility Rule

Ineligibility rules can be created using the Offer and Price Designer. For more information about how to use the Rule Designer, see the *Product and Service Catalog Offer and Price Designer User's Guide*.

Ineligibility rules are based on product characteristics, segments, and inventory. Similar to an eligibility rule, a customer with a specific characteristic can be ineligible for a product, if the product meets the segment requirements and if inventory is available. A user can have a combination of segments and inventory as a condition of this rule.

Get Offer

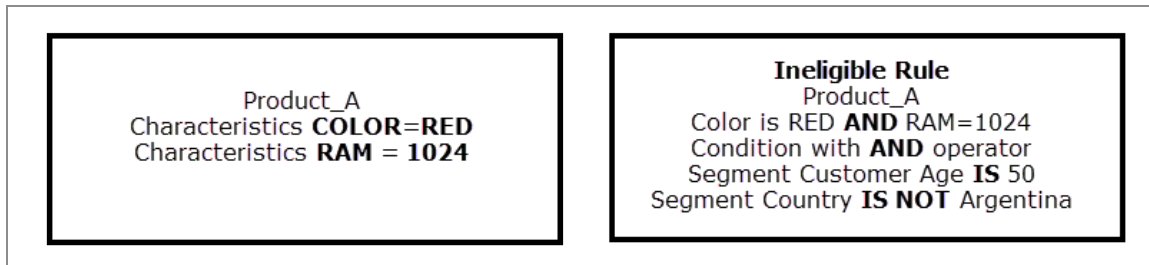
Ineligible products computed by OPE are evaluated against the ineligibility rules and are made ineligible if any applicable rule's condition is satisfied.

Validate Offer

Ineligible products that are present in the order lines are evaluated against ineligibility rules and are made ineligible if any applicable rule's condition is satisfied.

Segment Scenario

If a request contains two segment types Customer Age **IS** 50 **AND** Country **IS** India, the response would be **ineligible** for Product_A.



Incompatibility Rule

Incompatibility rules can be created using the Offer and Price Designer. For more information about how to use the Rule Designer, see the *Offer and Price Designer User's Guide*

Incompatibility rules are run between products and are based on product characteristics and UDFs.

Get Offer

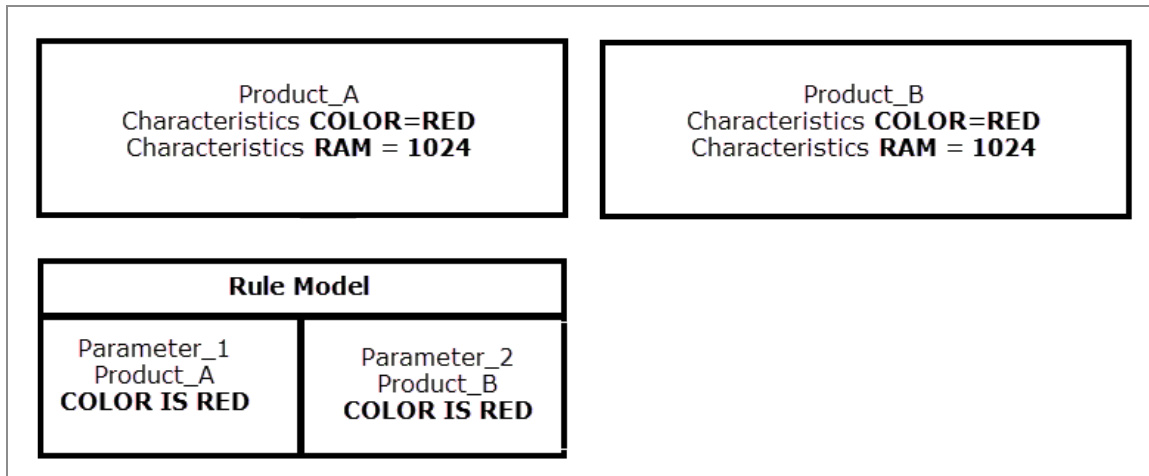
Incompatible products computed by OPE are compared with order lines and made incompatible if any applicable incompatible rule's condition is satisfied. If there are no order lines then Incompatibility rules are not applied.

Validate Offer

Incompatible products as present in the order lines are compared with products in other order lines and made incompatible if any applicable incompatible rule's condition is satisfied.

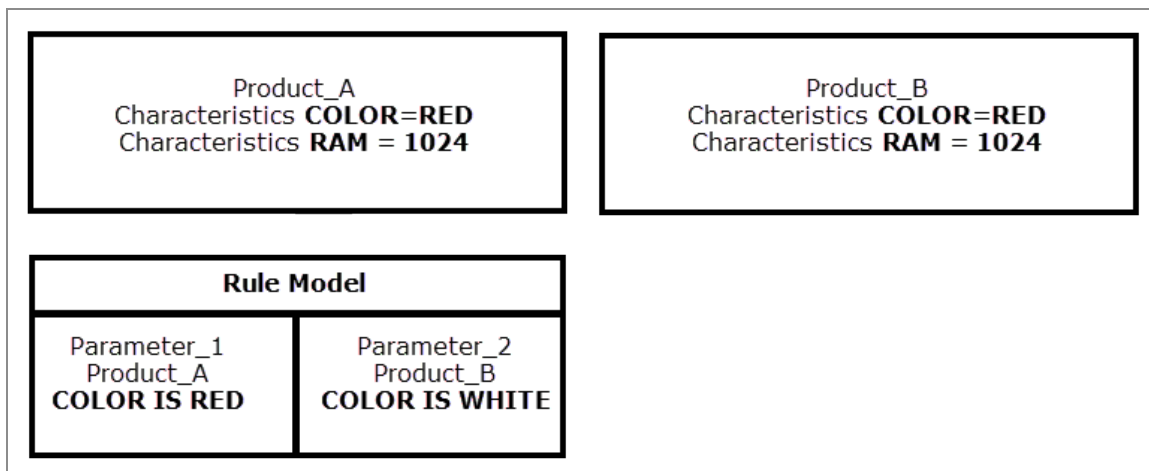
Segment Scenario 1

If a request contains one order line for **Product_B** with the Product ID of **Product_A** the response would be **incompatible** for Product_A.



Segment Scenario 2

If a request contains one order line for **Product_B** with Product ID **Product_A** the response would be **compatible** with Product_A



Offer Price Processing

TIBCO Offer and Price Engine uses the price models to get the prices configured in the price catalog and correlates these prices to a given product based on the relationship.

Price models can be loaded in the engine using the offline or the online integration, or both online and offline integration with TIBCO Product and Service Catalog. For more information on the price models offline and the discount models offline, see [Offline Model Loading](#).

Pricing Fields

The following pricing fields are the basic fields applicable to all price types.

Pricing Field	Description
RECORD_TYPE	This field defines the price type of the price. The valid values are TARIFFUSAGE, USAGE, FIXED, RECURRING, COMPOSITE, ONE_TIME, and MIGRATION_FEE.
Start Date/Start Time, End Date/End Time, Duration/Duration UOM	These characteristics are used to define the validity of a price in a certain time frame. None of these fields are mandatory. If the End Date/End Time is not present, it is calculated based on the Start Date/Start Time and the Duration/Duration UOM if they are present. Duration UOM can have Year, Month, Week, and Day as valid parameters. If only Start or End parameters are defined, only that boundary is taken into account to verify the validity of that price.
Charge Value	This characteristic is used to specify the amount in a dot-separated manner.
Currency	This characteristic is used to specify the type of currency for the price.
Charge Priority	This characteristic is used if a product provides several prices, whereby the priority provides the system a way to choose what charge to bill the customer. Therefore the price with the higher Charge Priority is taken. Lower the number the higher the priority.
Min Amount	This field attributes the minimum amount a price can have. If a price is reduced by multiple discounts, it is possible to reduce the price to a negative amount. The Min Amount attribute prevents that as it sets the amount to the specified value if the price value falls below that limit. In the end, all kinds of price and discount selections are done and all discounts, which are attached to the price, are returned even if they have not been applied due to the Min Amount.

Price Types

The following pricing types can be used in the OPE Price Model:

Price Type	Description
ONE_TIME price	This type specifies a non-recurring (one-off) charging element defined as fixed monetary values. This is modeled as a One-Time Price. One_Time prices are created by setting the Class to 'ONE_TIME'. CHARGE_PER, CHARGEMINBOUNDARY, and CHARGEMAXBOUNDARY are not used.
RECURRING Price	<p>This type specifies recurring charging elements defined as monetary values. This is modeled in TIBCO Product and Service Catalog as a RECURRING price. Recurring prices are created by setting the class to RECURRING.</p> <p>Note: Prices with Boundaries must not be attached directly to a Product. The reason is that it is nearly impossible to distinguish if the Prices must coexist for a Product or if they rule each other out. To accomplish this coexistence, use a composite price and attach appropriate prices with boundaries as children to that composite price.</p>
Usage Price	<p>This type specifies usage-based charging elements defined as monetary values. This is modeled in TIBCO Product and Service Catalog as a usage price. Usage prices are created by setting the class to USAGE.</p> <p>Note: Prices with Boundaries must not be attached directly to a Product. The reason is that it is supported to distinguish if the Prices must coexist for a Product or if they rule each other out. To accomplish this coexistence use a composite price and attach appropriate prices with boundaries as children to that composite price.</p>
COMPOSITE	The composite price is not calculated as a price itself but is meant as a container for child prices.

Relationships

Restrictive relationships are used to make a price applicable only to certain compositions of products, segments, or characteristics in a request. The following table explains the different types of relationships used and their behavior.

Relationship	Description
ProductPricedBy	<p>Prices are attached to a product using the ProductPricedBy relationship. This relationship has a ChargePriority attribute, which specifies the priority in which the charges must be applied in case there are multiple prices associated. If the ChargePriority attribute is equal for several prices, in the case of multiple prices of the same type, prices are chosen based on the restrictions it has. So if there are two valid prices configured for a product, the application takes the price with more and higher valued restrictions first.</p> <p>Note: Choosing a mechanism is only performed for prices of the same class attached to the same product. For example, ONE TIME prices are not to be compared to RECURRING prices.</p>
PriceRequiresSegment	<p>The PriceRequiresSegment relationship is used to model prices, which must be offered only to customers belonging to a certain segment. Prices can have PriceRequiresSegment relationships to multiple Segments. In that case, the price only is taken if all the segments are passed in the getPrices request.</p> <p>Note: PriceRequiresSegment relationship is considered as the most important relationship when choosing between different prices for a product.</p>
PriceRequiresProductGroup	<p>The PriceRequiresProductGroup relationship is used to model prices that must only be offered when ordering a certain composition of products. The PriceRequiresProductGroup relationship is considered as the second highest after PriceRequiresSegment relationship when choosing between different prices for a product.</p> <p>The PriceRequiresProductGroup repository has three flags in the Scope tab named CalculatedProducts, RelatedProducts, and LinkedOnly. The CalculatedProducts and RelatedProducts flags define if the RequiresProductGroup</p>

Relationship	Description
	<p>only 'searches' in CalculatedProducts and RelatedProducts passed in with the request. The LinkedOnly parameter defines if the search is only restricted to other CalculatedProducts and RelatedProducts sharing the same defined LinkDefinitions, as one of the prices must be applied to both the products to have the same Subscriber.</p> <p>The default values for CalculatedProducts and RelatedProducts are true and false for LinkedOnly, the engine looks for all products specified in the request unless it is specifically configured differently in the RequiresProductGroup. If LinkDefinitions is not configured, it is set to SubscriberID by default.</p>
PriceRequiresCharacteristic	<p>The PriceRequiresCharacteristic relationship is used to model different prices for a product based on its characteristic value. Prices can have PriceRequiresCharacteristic relationships to multiple Characteristics.</p> <div data-bbox="647 1026 1333 1129"> <p>Note: The PriceRequiresCharacteristic relationship is considered as the least important relationship when choosing between different prices for a product.</p> </div>
PriceComprisedOf Relationship for Price and Product	<p>Prices on a bundle level are either aggregated prices of the underlying offerings or set directly on the bundle level and ignore the prices of the underlying offerings. Both product and price can have children, which are aggregated towards the parent. The SubClass attribute with Override value prevents this, and the price acts as an override.</p> <p>PriceComprisedOf relationship fulfills these requirements, but the PriceComprisedOf relationship can be used to reuse multiple existing prices as children and attach a discount on all children by using the PriceAlteredByDiscount relationship on the parent level of the containing price. The override behavior of parent prices is transferred to all child prices. A child price has to have the same class as the parent otherwise it is ignored on run time. To group multiple prices with different types, the composite price has to be used.</p>

Get Prices Determinations and Calculations

All products in the get prices service are decomposed. Prices and discounts are determined in the offer; then the price is calculated.

Product Decomposition

All products defined in CalculationProduct(s), Optional product(s), and InventoryProduct(s) are decomposed in their auto-provisioned child products. CalculationProducts are decomposed in their optional or non-auto-provisioned children. To identify optional children, the parent product and the child product must share the same LinkDefinitions, such as the same subscriber, or certain UDFs. The fields that are taken for the link are specified in the ProductComprisedOf relationship. This enables the pricing part of OPE to detect child products. When setting LinkedOnly in the ProductRequiredForPriceGroup and ProductRequiredForDiscountGroup, the LinkDefinitions can differ from the LinkDefinitions that are used to decompose products.

Price Determination

The price for each product is determined by its priority for the product. In the case of the priority being the same, the weight for the price is decided as per the relationships. The price is determined for the quantity mentioned for the calculation product. This means if the quantity is specified as >1, the price is mapped for each price type (CLASS) until all the quantities are exhausted for that price type. The price for the parent price is calculated first. Then the child prices using the PriceComprisedOf relationship are calculated for the parent price quantity and are then aggregated towards the parent price. The quantity is reset for each new price type.

Discount Determination

The discount for the price is determined according to the priority mentioned in the price model using the PriceAlteredByDiscount relationship. In case the priority has the same weight for the discount, it is decided according to the relationships. The engine determines discounts for each price quantity. Unit discounts are detected for the complete quantity of the price. Discounts for child prices are applied before applying the parent price discount. Parent Price discounts are applied to the child prices as well.

Price Calculation

After the price and discount determination, all prices are resolved by aggregating child price elements up to the respective price level. In the top-level prices creation process, all attached discounts are applied accordingly, and the price amounts are adjusted accordingly. Discounts are applied in the following order or with the following conditions:

- If a discount has child discounts, first the parent discount is applied and then the child discount. If a discount has multiple direct children, the children are applied according to their order.
- The discount on a child's price is applied first.
- If a parent price has a discount, the discount is applied both to the parent price and the child price.

Offer Discount Processing

TIBCO Offer and Price Engine uses the discount models to get the discounts configured in the discount catalog and correlates these discounts to a given price based on the relationship.

Discount Fields

The following discount fields are the basic fields applicable to all discount types.

Pricing Field	Description
RECORD_TYPE	This field defines the record type of the discount. The valid values are DISCOUNT.
Start Date/Start Time, End Date/End Time, Duration/Duration UOM	These characteristics are used to define the validity of a price in a certain time frame. None of these fields are mandatory. If the End Date/End Time is not present, it is calculated based on the Start Date/Start Time and the Duration/Duration UOM if they are present. Duration UOM can have Year, Month, Week, and Day as valid parameters. If only the Start or End parameters are defined, only that boundary is taken into account to verify the validity of that price.

Pricing Field	Description
Discount Value	The discount value for the discount model.
Currency	This characteristic is used to specify the type of currency for the price.
Discount Priority	<p>This characteristic is used if a product provides several prices, whereby the priority provides the system a way to choose what charge to bill the customer. Therefore, the price with the higher Charge Priority is taken. Lower the number, the higher the priority.</p> <p>This field is used if a price has multiple discounts, whereby the priority provides the system a way to choose which discount to attach. Therefore the discount with the higher (lower number) Discount priority is taken.</p> <p>If the DiscountPriority is equal for several discounts in the case of multiple discounts attached to a price, the discount is chosen based on the restrictions it has. So if there are two valid discounts configured for a price, the application takes the discount with more or higher valued restrictions first. For example, for Price A there are three discounts configured. Discount A without any DiscountRequiresCharacteristic restriction, Discount B with a DiscountRequiresProduct restriction, and Discount C with a DiscountRequiresSegment restriction. For a getPricesRequest which fulfills the restrictions of all three discounts, Discount C is taken as it has the highest value restriction. For a getPricesRequest, which fulfills only the restrictions of Discount A and B, Discount B is taken as it has the highest value restriction. Multiple restrictions sum up whereby the summed value is taken for the determination of which discount to choose.</p>

Discount Types

The following discount types can be used in the OPE Price Model:

Price Type	Description
DISCOUNTUOM	Percent or Flat is used to define if the discount is altering a price in a

Price Type	Description
	percent manner or a total reduction manner; valid values are a percent or flat.

Relationships

Restrictive Relationships are used to make a discount applicable only to certain compositions of products, segments, or characteristics in a `getPrices` request.

Relationship	Description
DiscountRequiresProduct	<p>The DiscountRequiresProduct relationship is used to model discounts which must only be applied when ordering a certain composition of products. The ProductRequiredForDiscount is used because it follows a different mechanism and offers more flexibility in terms of modeling capabilities. Discounts can have ProductRequiredForDiscount relationships to multiple Products.</p> <p>Note: All groups must be fulfilled to make the discount applicable.</p> <p>Note: The ProductRequiredForDiscount relationship is considered as the lower important type after the DiscountRequiresSegment relationship when choosing between different prices for a product.</p>
ProductRequiredForDiscount	<p>The RequiresProductGroup repository has three flags in the scope tab named CalculatedProducts, RelatedProducts, and LinkedOnly. The CalculatedProducts and RelatedProducts flags define if the RequiresProductGroup only 'searches' in CalculatedProducts or RelatedProducts passed in with the request. The LinkedOnly parameter defines if the search is only restricted to other CalculatedProducts and RelatedProducts sharing the same defined LinkDefinitions</p>

Relationship	Description
	<p>as the one the discount must be applied to, for example, if the LinkDefinition is SubscriberID, both products have to have the same Subscriber.</p> <p>Note: The default values for CalculatedProducts and RelatedProducts are true and false for LinkedOnly, so the engine looks for all products specified in the request unless it is specifically configured differently in the RequiresProductGroup.</p> <p>Note: If LinkDefinitions is not configured, it is set to SubscriberID by default.</p>
DiscountRequiresSegment	<p>The DiscountRequiresSegment relationship is used to model discounts that must be offered only to customers belonging to a certain segment. Discounts can have DiscountRequiresSegment relationships to multiple segments. In that case, the discount is only taken if all the segments are passed in the getPrices request.</p> <p>Note: The DiscountRequiresSegment relationship is considered as the most important relationship when choosing between different prices for a product.</p>
DiscountRequiresCharacteristic	<p>The DiscountRequiresCharacteristic relationship is used to model different discounts for a price based on a characteristic value of the corresponding product. Discounts can have DiscountsRequiresCharacteristic relationships to multiple characteristics. In that case, all the restrictions must be fulfilled to make the discount applicable.</p> <p>Note: The DiscountRequiresCharacteristic relationship is considered the least important relationship when choosing between different discounts for a price.</p>

Relationship	Description
DiscountComprisedOf	<p>This relationship is used to reuse existing discounts as children and apply an aggregated discount based on the discount amount of the children.</p> <p>Note: A child discount has to have the same discount type (Percent or Flat) as the parent, otherwise, it is ignored.</p>

Java Extension

TIBCO Offer and Price Engine can be customized to support the functionality that can be integrated by using the Java extension.

The following phases and hooks are provided for every phase:

GetOffer

Phase	Hook in ope_config file
Custom Validation	customOfferValidationRule
Custom Filter Processing	customOfferFilterRules
Custom Eligibility	customEvaluateEligibilityRule
Custom Publishing	customOfferResponseRule

Validate Offer

Phase	Hook in ope_config file
Custom Validation	customOfferValidationRule
Custom Publishing	customOfferResponseRule

GetProductInformation

Phase	Hook in ope_config file
Custom Filter Processing	customOfferFilterRules (If the request contains query elements)
Custom Publishing	customProdInfoResponseRule

GetPrices

Phase	Hook in ope_config file
Custom Processing	customPriceProcessingRule
Custom Publishing	customPriceResponseRule

GetPriceInformation

Phase	Hook in ope_config file
Custom Processing	priceCustomInfoProcessingRule
Custom Publishing	customPriceResponseRule

Before you begin

For more information about the files used for the implementation of this Java extension, see the *Java API References* documentation.

To extend the execution of TIBCO Offer and Price Engine web service requests during different phases, add the java-based custom classes as described in the following steps:

Procedure

1. To implement the custom logic by extending the interfaces, create a java project such as `opeExtensions.zip`, and complete the following steps:
 - a. To add the rule actions, extend abstract class `"com.tibco.fos.opes.server.ope.extension.AbstractExtensionRuleAction"`.
 - b. To add the rule conditions, extend abstract class

```
"com.tibco.fos.opes.server.ope.extension.AbstractExtensionRuleCondition".
```

2. Create a JAR file of the project and copy it in the \$OPE_HOME/roles/ope/standalone/lib directory.
3. In ConfigValues_OPES.json file, provide the following property:

```
{ "propName": "com.tibco.af.ope.ope.ext.classes",
  "propDescription": "Fully qualified class names for OPE extension
  classes",
  "propValue": "c1::com.tibco.fom.ope.ext.GetPricesResponseCondition,
  a1::com.tibco.fom.ope.ext.GetPricesResponseAction",
  "valueType": "string",
  "isTenantProperty": "false"
}
```

4. Navigate to the \$OPE_HOME/roles/ope/standalone/config directory, and configure the custom class in the ope_config.xml rule sequence configuration file.

For example, to configure custom classes for validation, add them under the following events:

- a. Add the action class under the **"action"** element.
- b. Add the condition class under the **"condition"** element.

Note: The `<sequencedRule>` indicates an element where all the actions are executed sequentially. One condition can have multiple actions inside a `<sequencedRule>` element. Multiple `<sequenceRule>` elements are used to separate conditions and their corresponding actions.

Example for "customOfferValidation":

```
<!-- Custom Validation Class goes here -->
  <sequencedECARuleSet ruleSetName="customOfferValidationRule"
runOnFailure="false" runOnError="false">
    <event eventName="customOfferValidation" type="event">
      <eventSource>
        <endpointType>
          <uri>sape://localhost</uri>
        </endpointType>
        <inMessageType>
          <javabean/>
        </inMessageType>
      </eventSource>
      <lifetime/>
    </event>
    <sequencedRule>
      <action actionType="bean">
        <actionBean beanId="a1"
methodName="execute" interface="com.tibco.aff.eca.base.Action"/>
      </action>
      <condition conditionType="bean">
        <conditionBean beanId="c1"
methodName="evaluate"
interface="com.tibco.aff.eca.base.Condition"/>
      </condition>
    </sequencedRule>
  </sequencedECARuleSet>
```

Example for "customEvaluateEligibility":

```
<!-- Custom Validation Class goes here -->
  <sequencedECARuleSet ruleSetName="customEvaluateEligibilityRule"
runOnFailure="false" runOnError="false">
    <event eventName="customEvaluateEligibility" type="event">
```

```

        <eventSource>
            <endpointType>
                <uri>sape://localhost</uri>
            </endpointType>
            <inMessageType>
                <javabean/>
            </inMessageType>
        </eventSource>
    </lifetime/>
</event>
<sequencedRule>
    <condition conditionType="bean">
        <conditionBean beanId="c1"
methodName="evaluate"
interface="com.tibco.aff.eca.base.Condition"/>
    </condition>
    <action actionType="bean">
        <actionBean beanId="a1" methodName="execute"
interface="com.tibco.aff.eca.base.Action"/>
    </action>
</sequencedRule>
</sequencedECARuleSet>

```

Note: All the web services are supported for Java customization.

Search

The `searchOffers` API, which is part of `offersearchindex-service` is a REST API that supports an HTTP Post operation. The request and response payloads are JSON-based. The API presents filters and attributes in the request to enable full-text search using the product data models. This API supports full-text search across all fields present in the product modes that are marked for searching.

Note: Search is applicable only for product models as only product index is created in Offer Search Engine.

For more information, see the examples in *TIBCO® Offer and Price Engine Web Services Guide*.

Misspelled Search Terms

This feature suggests terms for attributes that are incorrectly spelled.

When using this feature if you specified LAPTOP, the Offer Search Engine returns a response for the LAPTOP product.

The text values in the searchable characteristics for products that use this feature have to be configured as follows:

```
<ns0:simpleRule>
  <ns0:name>SPELLSUGGESTIONTERM</ns0:name>
  <ns0:ruleSetOutcome>Laptop,PROD_A1</ns0:ruleSetOutcome>
</ns0:simpleRule>
```

This functionality is turned off by default and must be enabled by using the `com.tibco.af.ope.flags.suggest` flag in the application configuration.

Index

The Offer Search Index Service creates product indices and processes the incoming product models to index them in the Offer Search Engine.

Product Index

To create a product index for each tenant, the tenant has to be configured in the application configuration.

At startup, the Offer Search Index Service tries to create a product index for all the configured tenants by calling The Create Index API of the Offer Search Engine. This creates a `product_<tenant_id>` index in the Offer Search Engine.

Index Mapping

- By default, the Offer Search Engine uses the existing incoming product model to create the corresponding domain object. To support TIBCO Offer and Price Engine based queries and dynamic product model changes, a custom index mapping file is used. This file is used as an input along with the source product model document to index.
- For full-text purposes, all the fields eligible for full-text search are copied in the `productFullTextSearch` field of the product mapping file so that the search is

executed only across the field instead of inserting multiple fields in the search query.

- The category from the product is mapped as follows:

```
"categories" : {
  "type": "nested",
  "properties" : {
    "name" : {
      "copy_to": "productFullTextSearch",
      "type": "keyword"
    },
    "childCatList" : {
      "properties" : {
        "name" : {
          "copy_to": "productFullTextSearch",
          "type": "keyword"
        },
        "childCatList" : {
          "properties" : {
            "name" : {
              "copy_to": "productFullTextSearch",
              "type": "keyword"
            },
            "level" : {
              "type" : "long"
            }
          }
        },
        "level" : {
          "type" : "long"
        }
      }
    },
    "level" : {
      "type" : "long"
    }
  }
},
```

- Segment mapping is as follows:

```
"compatibleSegments": {
  "type": "nested",
  "properties": {
    "segmentName": {
      "copy_to": "productFullTextSearch",
      "type": "keyword"
    },
    "segmentType": {
```

```

        "copy_to": "productFullTextSearch",
        "type": "keyword"
    }
}
},

```

- Record Type/ SubTypes are nested as follows:

```

"classification": {
    "type": "nested",
    "properties": {
        "type": {
            "copy_to": "productFullTextSearch",
            "type": "keyword"
        },
        "subType": {
            "copy_to": "productFullTextSearch",
            "type": "keyword"
        }
    }
},

```

Classifications have types that contain subtypes as a part of the model, this enables aggregation on type and subtype elements.

- Suggestion terms are mapped as follows:

```

"suggestionTerms": {
    "type": "text",
    "fields": {
        "keyword": {
            "type": "keyword",
            "ignore_above": 256
        }
    }
}

```

- Category attributes are mapped as follows:

```

"filterAttributes": {
    "type": "nested",
    "properties": {
        "name": {

```

```

        "copy_to": "productFullTextSearch",
        "type": "keyword"
    },
    "value": {
        "copy_to": "productFullTextSearch",
        "type": "keyword"
    }
},

```

Indexing Product Documents

After the Offer Search Index Service has started and the tenant index created, the engine can process models and index them against Offer Search Engine product index. The Offer Search indexes the product models as follows:

1. Loads models for processing (from online and offline modes) and parses them.
2. Sends the models to the Offer Search Engine for indexing against the tenant-based product index created earlier. The products are indexed in bulk and indexes all products present per product model xml file at the same time. For example, if the product model xml contains ten products, all ten products are indexed together in a single request instead of ten individual requests.

Browse

With the filters and attributes present in the Search API, you can browse using category, segment, and record types and subtypes as specified in the Product data model.

This functionality enables you to browse for products using the navigation paths. For more information, see the examples in *TIBCO® Offer and Price Engine Web Services Guide*.

Product Id and Product Id Ext.

You can now use the same unique productID for two different products. Previously, a unique productID was assigned to each product.

You can differentiate the two products by using the PRODUCTIDEXT characteristic.

Example:

Product 1: Value of productID is 'A' but has no PRODUCTIDEXT

Product 2: Value of productID is 'A' and PRODUCTIDEXT is 1

Implementation of LDAP

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

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

- `directoryServiceDomainName=test`
- `directoryServiceRootDistinguishedName=DC=testad,DC=com`
- `ldapURLForDirectoryService=ldap://localhost:389`

Authentication and Authorization

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

Authentication Factors

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

Authorization Technique

The role-based access control technique is used to give users access to the TIBCO Offer and Price Engine resources.

Authorization Service

Authorization Service is a microservice available as part of TIBCO Offer and Price Engine 6.0.0 This microservice has the following key features:

- This service generates JWT tokens based on OAuth2 specifications.
- Grant_Type used is password

- User credentials are entered in the request body when generating the OAuth2 token.
This service accepts encrypted passwords.

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

Resource Server

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

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

- Verify token validity by using the same signing that was used to sign the token when it was generated by Authorization Service.
- Check token expiry.
- Extract claims from the token and set TenantId and authorities for the user.
- As part of the microservices configuration in TIBCO OPE, each API exposed for the user has been protected with configurable role restrictions.



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

Generating User Token

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

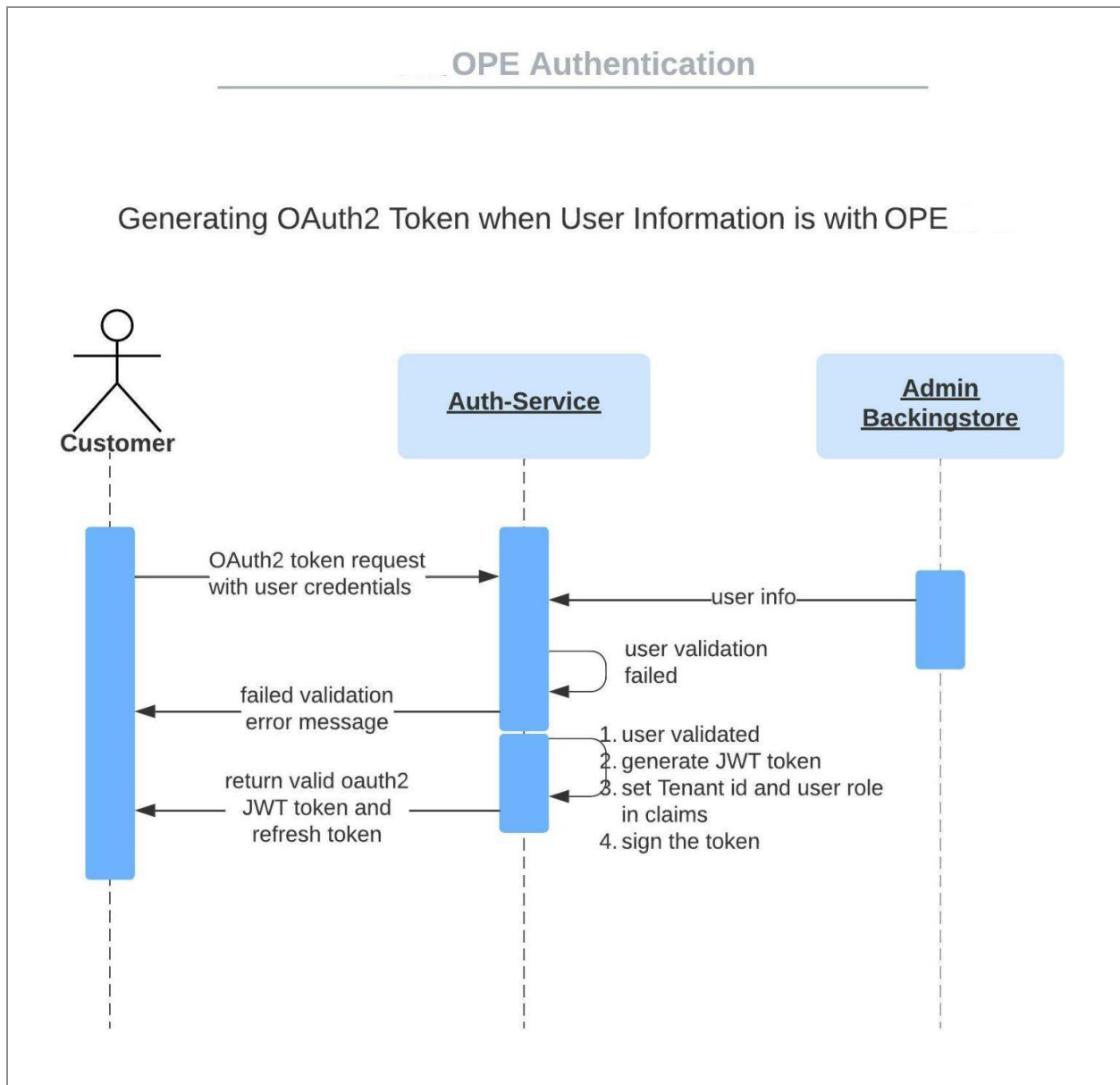
- [Information stored with Authorization Service](#)
- [Information stored in External Directory](#)
- [Using a third-party external service](#)

Information stored with Authorization Service

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

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

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

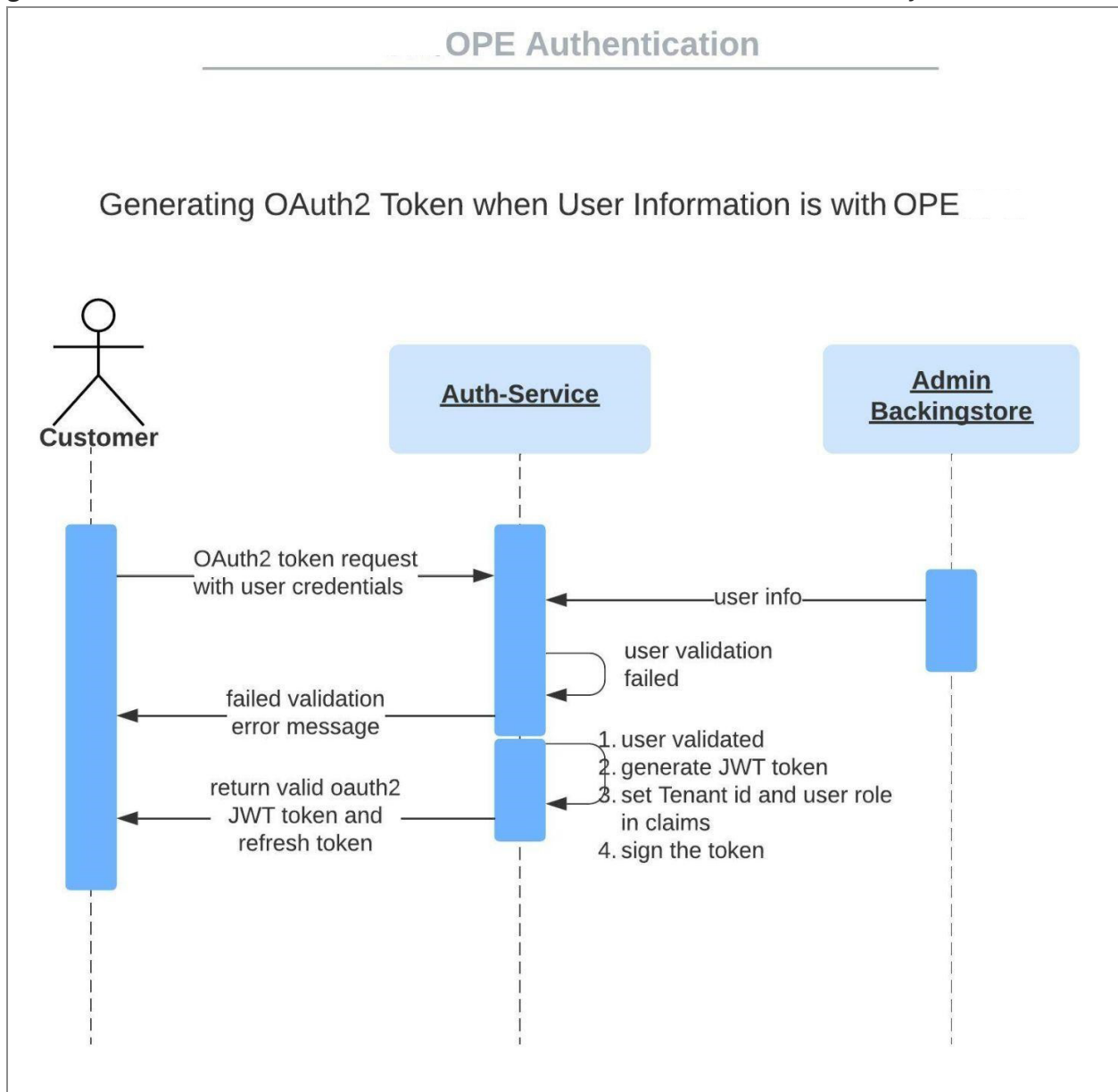


Information stored in External Directory

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

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

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



Using a third-party external service

Customers might have an existing token generation mechanism (or service) and they might prefer using it for TIBCO OPE 6.0.0

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

- `user_name`

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

If the token follows all the above rules, then all the microservices in TIBCO OPE accept the token even if it is generated from a third-party service.

Sample Token

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX25hbWUiOiJhZG1pbIIsIlRFTkF
OV
ElEIjoiveIjCQ08iLCJzY29wZSI6WyJyZWZkIiwid3JpdGUlXSwiZXhwIjoxNjQ2MDc0MjY3L
CJhdXRob3JpdGllcyI6WyJST0xFX0FETU0iOj0sImp0aSI6IjViZDlhZmExLTJjZGEtNDZjO
S05NGJjLTU0NmVhbnNjc4ZiIsImNsaWVudF9pZCI6Im9yZGVyLW1hbmFnZW1lbnQtY
2xpZW50In0.tkbrHuFBvBo8N7Tshp4uXLxhjaYjBfWCoPccpCJtlxU
```

When you read the token, it displays the following payload.

```
{
  "header": {
    "alg": "HS256",
    "typ": "JWT"
  },
  "payload": {
    "user_name": "admin",
    "TENANTID": "TIBCO",
    "scope": [
      "read",
      "write"
    ],
    "exp": 1646074267,
    "authorities": [
      "ROLE_ADMIN"
    ],
    "jti": "5bd9afa1-2cda-46c9-94bc-
546ed69a678f",
    "client_id": "order-management-client"
  }
}
```

API Monitoring

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

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

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

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



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

JMX MBeans

Before you begin

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

Procedure

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

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

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

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

Prometheus

Procedure

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

Property Name	Value	Description
monitoringSystem	prometheus	<p>The monitoring system to view the application metrics.</p> <p>(Default: Null)</p> <p>Set the value (such as JMX, prometheus, elastic, Dynatrace) as per the required tool. You can also set multiple values here by comma separating them.</p>

2. Open the OPE management prometheus `http://<host>:<port>/management/prometheus` endpoint to view the metrics. You can also install the Prometheus application to view the prometheus metrics in a graphical representation.

Elasticsearch

Before you begin

Install the Kibana tool on your machine.

Procedure

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

Property Name	Value	Description
monitoringSystem	elastic	<p>The monitoring system to view the application metrics.</p> <p>(Default: Null)</p> <p>Set the value (such as JMX, prometheus, elastic, Dynatrace) as per the required tool. You can also set multiple values here by comma separating them.</p>
management.metrics.export.elastic.enabled	true	Determines whether to enable the Elastic metrics or not
management.metrics.export.elastic.host	http://localhost:9200	Elastic search Url
management.metrics.export.elastic.index	micrometer-metrics	Management

Property Name	Value	Description
		Metrics Elastic Index
management.metrics.export.elastic.step	1m	Time interval for sending metrics
management.metrics.export.elastic.userName	client's ElasticSearch Username (Default = "NULL")	ElasticSearch Username
management.metrics.export.elastic.password	client's ElasticSearch Password (Default = "NULL")	Encrypted ElasticSearch Password

2. Open the Kibana tool to view the metrics.

Dynatrace

Before you begin

Install Dynatrace tool on your machine.

Procedure

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

Property Name	Value	Description
monitoringSystem	Dynatrace	The monitoring system to view the application metrics. (Default: Null) Set the value (such as JMX,

Property Name	Value	Description
		prometheus, elastic, Dynatrace) as per the required tool. You can also set multiple values here by comma separating them.
management.metrics.export.dynatrace.uri	The host on which the Dynatrace tool is installed	The host on which the Dynatrace tool is installed
management.metrics.export.dynatrace.api-token	Access token generated from the Dynatrace tool	Access token generated from the Dynatrace tool
management.metrics.export.dynatrace.device-id	The Id of the device on which the Dynatrace tool is installed	The Id of the device on which the Dynatrace tool is installed
management.metrics.export.dynatrace.step	1m	Time interval for sending metrics
management.metrics.export.dynatrace.enabled	true	Determines whether to enable the dynatrace metrics or not. (Default: false)

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

WebClient Configuration

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

As a workaround, a retry mechanism is added.

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

Data Models

TIBCO Offer and Price Engine uses a variety of data models (catalogs) to support the product functionality:

Data Models	Description
Product Model	Used to evaluate product eligibility and handle web service queries like <code>getProductInformation</code> .
Rule Model	Used to support functionalities such as eligibility, ineligibility, and compatibility rules.
Discount Model	Used to support the offering and pricing functionality.
Price Model	
Category Model	Used to support the search and browse functionality.

Model Loading Process

The models mentioned in Data Models must be loaded into TIBCO Offer and Price Engine so that they can be used by different components. These data models are modeled as catalogs using repositories and relationships readily available in TIBCO Product and Service Catalog. After modeling the catalogs in TIBCO Product and Service Catalog, they can be made available to TIBCO Offer and Price Engine and these models are loaded through catalog service.

Use the following ways to load the models into TIBCO Offer and Price Engine:

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

Online Model Loading

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

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

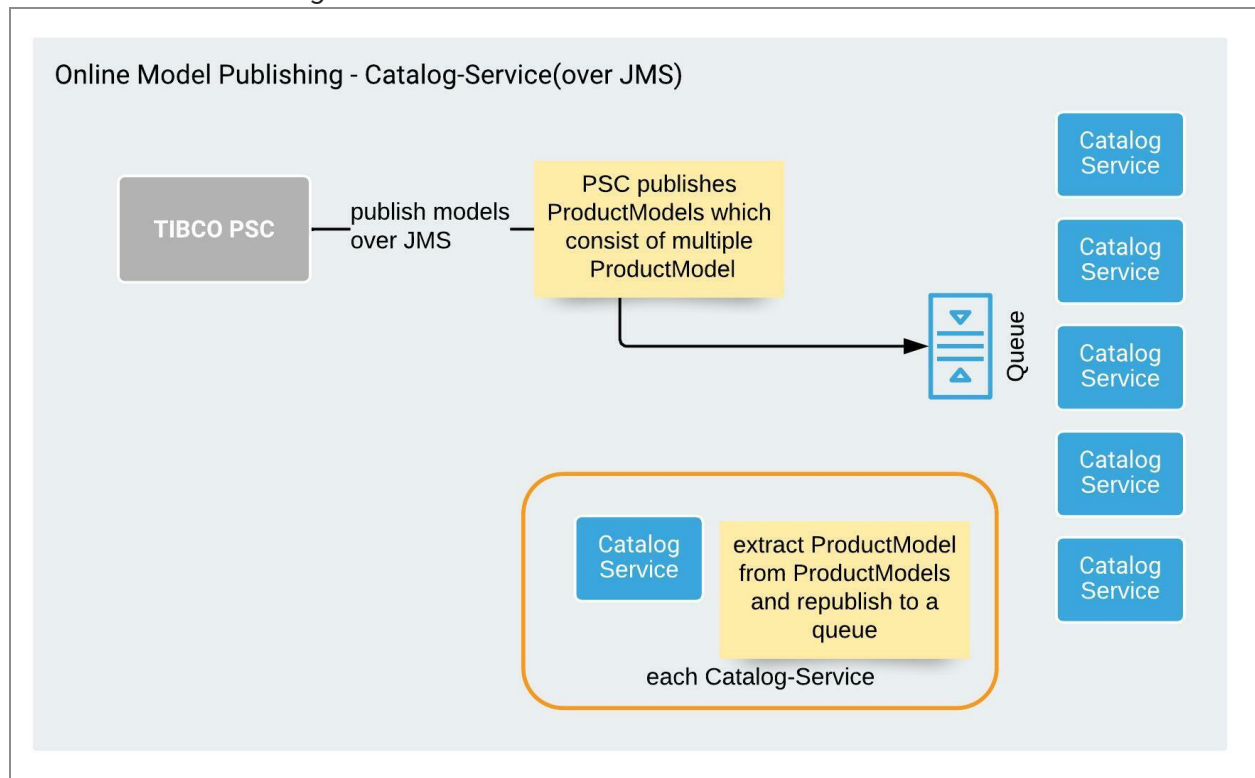
Invoke the request against the running instance of TIBCO Product and Service Catalog on the URL, which typically looks like

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

See the *TIBCO Product and Service Catalog* documentation for more details.

**Note:**

In the \$OPE_HOME/roles/configurator/standalone/config/ConfigValues_CatalogService.xml file, you must enable the `com.tibco.catalog.loading.using.ems` flag.

Online Model Publishing

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

Model (Catalog)	TIBCO Product and Service Catalog JMS Topic
Product	tibco.ac.productmodel.topic
category	tibco.ac.categorymodel.topic
discount	tibco.ac.discountmodel.topic
price	tibco.ac.pricemodel.topic
rule	tibco.ac.rulemodel.topic

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

TIBCO Product and Service CatalogSourceTopic	TIBCO Offer and Price Engine Target Queue
tibco.ac.productmodel.topic	tibco.aff.catalog.product.request
tibco.ac.categorymodel.topic	tibco.aff.catalog.category.request
tibco.ac.discountmodel.topic	tibco.aff.catalog.discount.request
tibco.ac.pricemodel.topic	tibco.aff.catalog.price.request
tibco.ac.rulemodel.topic	tibco.aff.catalog.rule.request

Catalog Web Service Model Loading

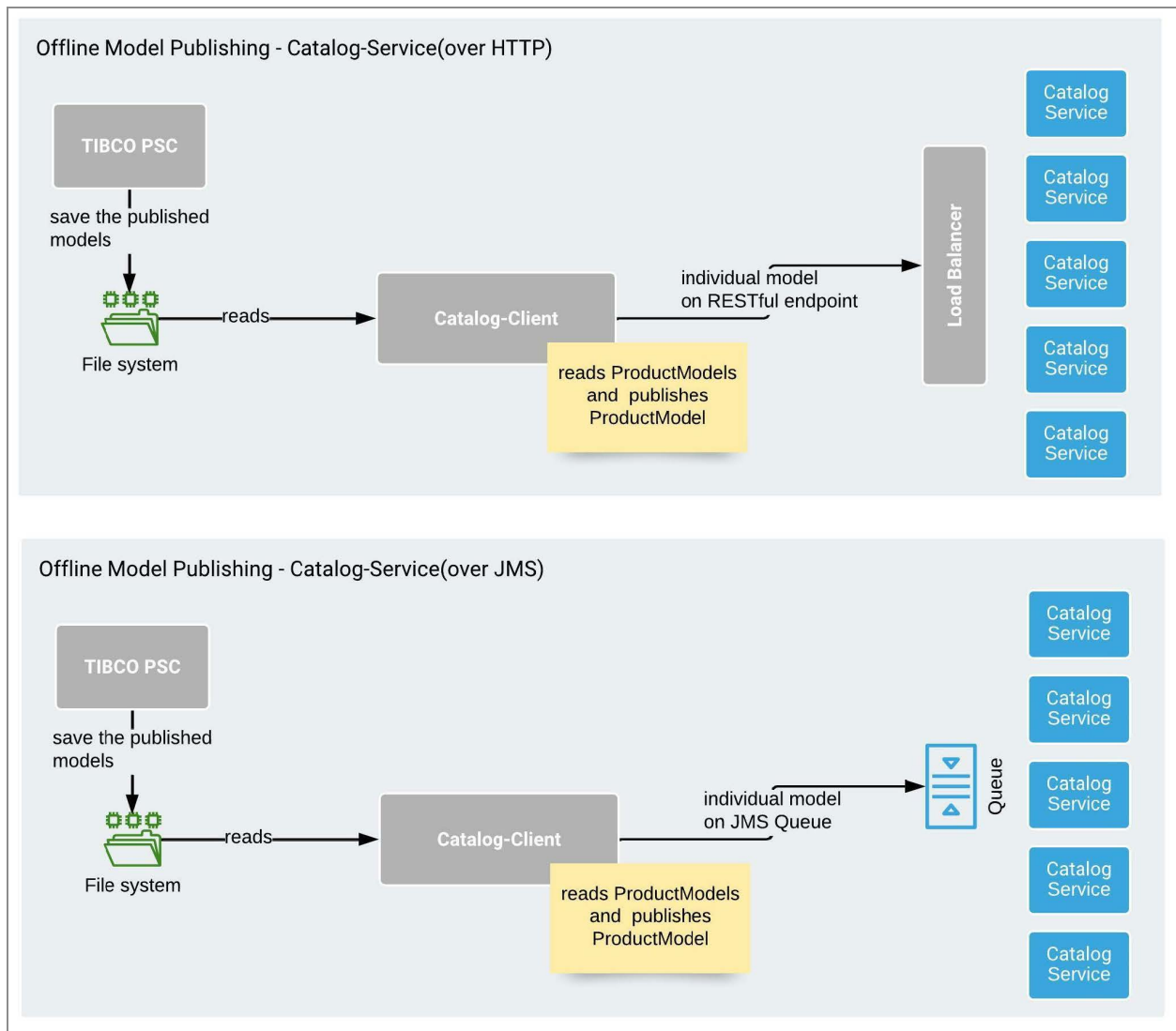
Catalog Web Service is used to load the models into TIBCO Offer and Price Engine. The URL for catalog service is `PROTOCOL://<HOST>:<PORT>/swagger-ui.html#/` Following are the types of Catalog web services:

- Post request for `/v1/productmodel`
Operation `SubmitProductModel` to load single product model
- Post request for `/v1/productmodel/bulk`
Operation `SubmitProductModels` to load multiple product models
- Delete request for `/v1/productmodel/bulk`
Operation `PurgeProductModel` to purge product model
- Post request for `/v1/categoryModel`
Operation `SubmitCategoryModel` to load single category model
- Post request for `/v1/categoryModel/bulk`
Operation `SubmitCategoryModels` to load multiple category models
- Delete request for `/v1/categoryModel/bulk`
Operation `PurgeCategoryModels` to purge category model

- Post request for /v1/discountModel
Operation SubmitdiscountModel to load single discount model
- Post request for /v1/discountModel/bulk
Operation SubmitdiscountModels to load multiple discount models
- Delete request for /v1/discountModel/bulk
Operation PurgediscountModels to purge discount model
- Post request for /v1/offerIdMappings
Operation SubmitOfferIdMappingsModel to load offerIdMappings model
- Delete request for /v1/offerIdMappings
Operation PurgeAllOfferIds to purge offerIdMappings model
- Post request for /v1/priceModel
Operation SubmitpriceModel to load single price model
- Post request for /v1/priceModel/bulk
Operation SubmitpriceModels to load multiple price models
- Delete request for /v1/priceModel/bulk
Operation PurgepriceModels to purge price model
- Post request for /v1/ruleModel
Operation SubmitruleModel to load single rule model
- Post request for /v1/ruleModel/bulk
Operation SubmitruleModels to load multiple rule models
- Delete request for /v1/ruleModel/bulk
Operation PurgeruleModels to purge rule model

Offline Model Loading by using Catalog Client Service

By using the catalog client service, you can upload offline files on JMS or HTTP. Through this catalog client service, you can upload any models from the offline directory.



To use the service, perform the following configurations in the `application.properties` file:

- Set the `catalogMode` flag as OPE for Offer and Price Engine.
- Set `catalogPublishMode` as JMS or REST per the requirement.
- Set the authorization properties to generate auth token.
- Set JMS or HTTP configuration and Offline model loading catalog URL.

The following REST service is used for uploading offline models.

Operation Upload Offline Models

Endpoint: `http://<host_address>:<port_address>/v1/uploadModel`

Method: POST

Select the **modelType** from the drop-down list and enter the **tenantId** for which you want to publish the models.

Models from the offline model loading directory are uploaded.



Note: It is a good practice not to use catalog client service on a cloud platform as the models are fetched from a local directory. To use this service with JMS, enable the `com.tibco.catalog.loading.using.ems` flag in the `$OPE_HOME/seed-data/app-properties/ConfigValues_CatalogService.json` file.

Shopping Cart Service

The REST APIs in the Shopping Cart service are used to perform various actions on the products in a cart. For example, you can save, update, delete, or get products in the cart. Specifications from TM Forum R19.0.0 are followed for the shopping cart APIs. For more information about the Shopping Cart APIs, see *TIBCO® Offer and Price Engine Web Services Guide*.

Authorization Service

Token-based authentication is implemented in TIBCO Offer and Price Engine to ensure secure access to TIBCO Offer and Price Engine Server REST APIs, and to support multitenancy. The authentication service in TIBCO Offer and Price Engine uses JSON WebToken(JWT) to validate user credentials (user name, password, and tenantID).

The following functions are covered under the Authorization Service:

- [Create User](#)
- [Update User](#)
- [Get User](#)
- [Delete User](#)

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

Create User

This request is used to create a new user.

Method: HTTP POST method

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

Create User Parameters

Parameter		Cardinality	Description
X-API-AppId		Mandatory	The application ID is used for getting user details. The default ID is auth.
X-API-Key		Mandatory	This key is used for getting user details. The default ID is auth.
userInfo (Body)	enabled	Mandatory	The value can be true or false. true makes

Parameter	Cardinality	Description
		the user accessible through the configurator UI and <code>false</code> makes the user disable.
password	Mandatory	The password to be used for the user.
tenantId	Mandatory	This is the TENANT value as stored in the users table in the database. If the <code>tenantId</code> is not present in the database, then a new TENANT is created.
userName	Mandatory	It specifies the user name to be created or modified.
userRoles	Mandatory	It assigns the role to the user. The default valid role values are <code>ROLE_ADMIN</code> and <code>ROLE_USER</code> . You can override the default roles if required.

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

Example for the Create and Modify User request:

```
{
  "user": [
    {
      "enabled": true,
      "password": "testpassword",
      "tenantId": "testTenant",
      "userName": "testuser",
      "userRoles": [
        "ROLE_ADMIN"
      ]
    }
  ]
}
```

Update User

This request is used to create a new user or update the existing one.

Method: HTTP PUT method

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

Update User Parameters

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



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

Example for the Update User request:

```
{
  "user": [
    {
      "enabled": true,
      "password": "testpassword",
      "tenantId": "testTenant",
      "userName": "testuser",
      "userRoles": [
        "ROLE_ADMIN"
      ]
    }
  ]
}
```

Get User

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

Method: HTTP GET method

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

Get User Parameters

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

Delete User

This request is used to delete the existing user.

Method: HTTP DELETE method

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

Delete User Parameters

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

Example for Delete User request:

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

Generating an authorization token

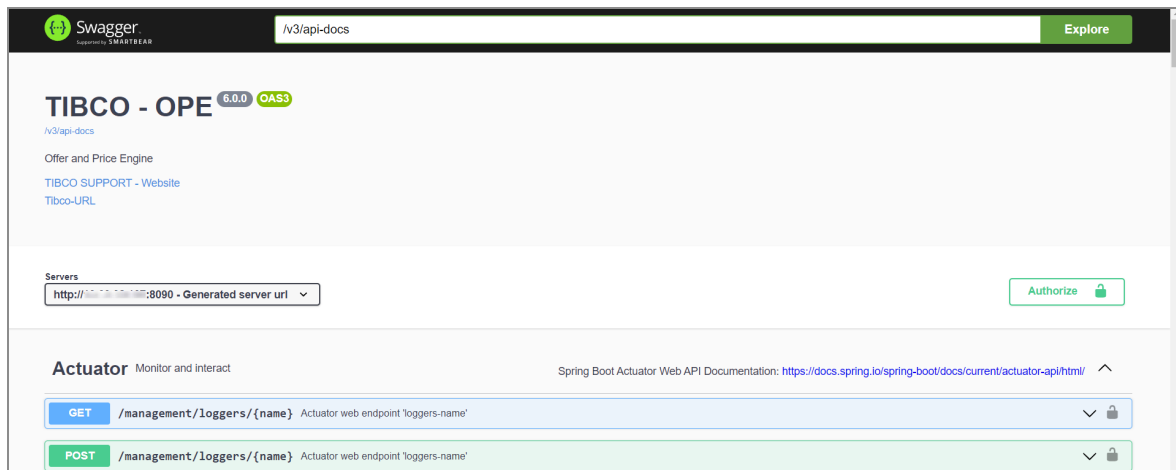
This token can be used to access operations of the services such as Catalog service, OPE endpoints service, Offer Search Indexing service, and Shopping Cart service. For the Configuration service and Authorization service, a token is not required.

Procedure

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

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

2. Click the **Authorize** button.



The **Available authorizations** window opens.

3. Pass the following mandatory parameters:

Authorization parameters and description

Element Name	Element Type	Description
username	String	username@tenantId
password	String	Existing password
Client credentials location		Select Authorization header or Request body from the drop-down options.
client_id	String	order-management-client

Element Name	Element Type	Description
client_secret	String	order-management-secret

Available authorizations x

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

OAuth Password (OAuth2, password)

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

username:

password:

Client credentials location:

Authorization header ▾

client_id:

client_secret:

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

☐ *read*
read scope

☐ *write*
write scope

Authorize

Close

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

Result

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

**Note:**

You can disable authorization by setting the value of `enableSecureAPI` flag as `false` from the `ConfigValues_Common.json` file under `<OPE_HOME>/seed-data/app-properties` directory, and upload the latest properties in the admin database.

Auth service for other OPE services authentication is not required when `enableSecureAPI` is set as `false`, but the configurator-ui service still needs the auth service to be functional to log in into the Configurator UI.

For soap services, to hit any OPE service for the non-default tenants, add a header with the tenant ID name and value of the tenant ID.

Generating an authentication token through the REST endpoints

Perform the following procedure to generate the authentication token through the REST endpoints:

Procedure

1. To authorize a particular service, open the postman client.
2. On the **Authorization** tab, fill the details in the **Username** and **Password** fields and then click **Send**.

POST `http://...:9091/oauth/token` Send

Params **Authorization** Headers (10) Body Pre-request Script Tests Settings

TYPE
Basic Auth

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)

Username: `order-management-client`

Password: `.....` ☐ Show Password

3. On the **Body** tab, fill the details in the **Username** and **Password** fields and then click

Send.

REST client interface showing a POST request to `http://[redacted]:9091/oauth/token`. The 'Body' tab is selected, showing a form with the following fields:

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> grant_type	password	
<input checked="" type="checkbox"/> scope	read+write	
<input checked="" type="checkbox"/> username	admin@TIBCO	
<input checked="" type="checkbox"/> password	admin	
Key	Value	Description

Result

An authorization token is generated for the particular service. This token is unique and valid only for the dedicated user with a tenant ID. The access token is valid for a limited period.

Authorization Token APIs

- [Generate authorization header](#)
- [Generate OAuth token](#)

Generate Authorization Header

This request is used to generate the authorization header for OAuth token endpoint.

Method: HTTP GET method

Endpoint: `http://<host_address>:<port_address>/v1/generateAuthHeader`

Generate Authorization Header Parameters

Parameter	Cardinality	Description
client_id	Mandatory	order-management-client
client_secret	Mandatory	order-management-secret

Generate OAuth Token

This request is used to generate authorization OAuth token.

Method: HTTP POST method

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

Generate Authorization Header Parameters

Parameter	Cardinality	Description
grant_type	Mandatory	You can select password or refresh token.
scope	Mandatory	You can select read, write, or 'read write'.
refresh_token		Refresh token from previously generated token. Required only when grant_type=refresh_token
username		Required only when grant_type=password
password		Required only when grant_type=password
tenantId		Required only when grant_type=password
Authorization	Mandatory	
Content-Type	Mandatory	

Use Cases

Review the use cases to have a better understanding of the different scenarios OPE can be used for.

Testing Product Eligibility Scenarios

Test product eligibility with the following five scenarios.

Scenario 1 - Testing with the SegmentsCompatibleWith Filter

Send an eligibility request using the SegmentCompatibleWith filter.

Procedure

1. Create three products: Phone1, Phone2, Phone3.
2. Set the following segment type and segment names for Phone1, Phone2, and Phone3:
 - Phone1 is compatible with `segmentType = country` and `segmentName = US`.
 - Phone2 is compatible with `segmentType = country` and `segmentName = Canada`.
 - Phone3 is compatible with `segmentType = country` and `segmentName = US`.
3. Request eligible products with the segment filter as `segmentType = country` and `segmentName = US`.

Result

Passing this specific segment filter in the eligibility request retrieves Phone1 and Phone3 as eligible products for the order because they are compatible with the segment type.

Scenario 2 - Testing with the SegmentCompatibleWith RecordType Filter

Send an eligibility request using the SegmentCompatibleWith RecordType filter.

Procedure

1. Create three products: Phone1, Phone2, Phone3.
2. Set the following segment types and names for Phone1, Phone2, and Phone3:
 - Phone1 as segmentType = country and SegmentName = US
 - Phone2 as segmentType = country and SegmentName = Canada
 - Phone3 as segmentType = country and SegmentName = US
3. Set the following RecordTypes types for Phone1, Phone2, and Phone3:
 - Phone1 as RecordType = White
 - Phone2 as RecordType = White
 - Phone3 as RecordType = Black
4. Request eligible products for segmentType = country, segmentName = US and RecordType = White.

Result

Passing this specific segment in the eligibility request and this specific RecordType retrieves only Phone1 because it is compatible with the segment and with the RecordType.

Scenario 3 - Testing with the Flag ReturnBundleOfferings Enabled

Send an eligibility request when the flag ReturnBundleOfferings is enabled.

Procedure

1. Create a group with Phone1 and SIM_Card as the parent products and Data_Plan as the child product.

2. Create another group with Phone2 as the parent product and Data_Plan as the child product.
3. Set `autoprovision = false` for Data_Plan.
4. Set the `ReurnBundleOfferings = true`.
5. Add Data_Plan to your order.
6. Send a request for all eligible products.

Result

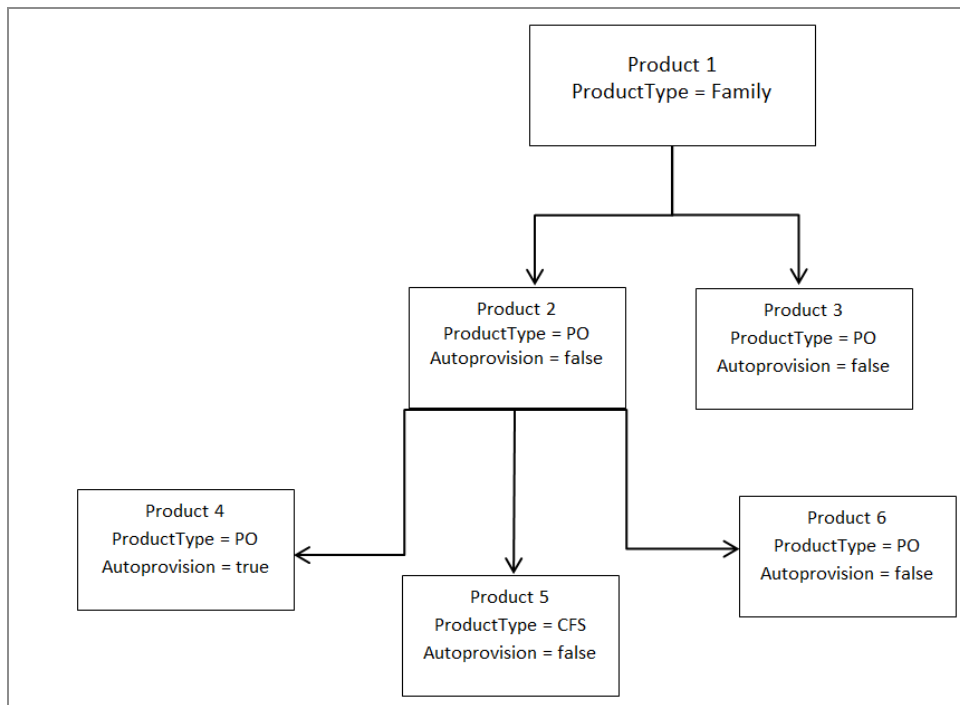
Phone1, Phone2, and SIM_Card are returned as eligible products with Data_Plan in the order.

Scenario 4 - Testing with the Focus Filter with Different Product Types and `autoprovision=true`

Send an eligibility request using the focus filter for products with different types when some products have `autoprovision` set to `false` and others set to `true`.

Procedure

1. Set a hierarchy of products with different ProductTypes.
2. Have some products with `autoprovision = true` and some with `autoprovision = false`.



3. Set the flag `decomposeProducts="false"`, so the eligible products are not decomposed for `autoprovision = true` and for product and product type filter. The autoprovisioned products are still evaluated for segment and product compatibility.
4. Request eligible products for `ProductType = PO`.

Result

Passing the specific ProductTypes in the eligibility request retrieves Product 2, Product 3, Product 4, and Product 6 because they are compatible with the ProductType and with `decomposeProducts="false"`.

Scenario 5 - Testing with Maximum Number of Products in a Group is Reached

Send an eligibility request when the maximum number of products in a group is reached.

Procedure

1. Create a hierarchy of products within a group. Name the group Phone Bundle.
2. Set the maximum number of instances of the products in the group to two.

3. Add two products with `AUTOPROVISION = false` from the Phone Bundle group to the order. So now the order has 3 products - Bundle and its two child products.
4. Children from the Phone Bundle group are evaluated for eligibility.

Result

All the remaining children in the Phone Bundle group are not eligible because the group has a restriction of the maximum number of two, so only one Bundle Product can be eligible.

Testing Product Validation Scenarios

Test product validation with the following 11 scenarios.

Scenario 1 - Testing with Maximum Number of Products in a Group is Reached

Send a validation request when the maximum number of products in a group is reached.

Procedure

1. Create a hierarchy of products within a group. Name the group Charging Bundle.
2. Set the maximum number of instances of the products with `AUTOPROVISION = false` in the group to four.
3. Add five products from the Charging Bundle to the order.
4. Send a request of validation for the order for bundle products and five child products.

Result

The validation triggers an error because the Charging Bundle only supports a maximum of four.

Scenario 2 - Testing with Minimum Number of Products in a Group is Not Reached

Send a validation request when the minimum number of products in a group is not reached.

Procedure

1. Create a hierarchy of products within a group. Create two products in that group: Phone1 and Phone2 with AUTOPROVISION = false.
2. Set the restriction of a minimum number of one for products within the group.
3. Do not add Phone1 or Phone2 to the order.
4. Send a request for validation for the order.

Result

The validation triggers an error because the group is defined to have at least one element.

Scenario 3 - Testing with the Compatibility Relationship

Send a validation request with incompatible products.

Procedure

1. Create two incompatible products with each other.
2. Add both products to your basket.
3. Send a request for validation for the order.

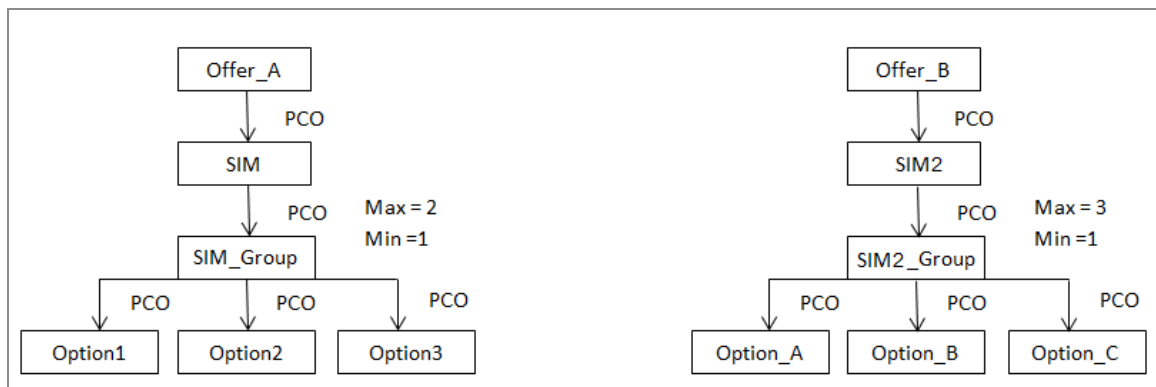
Result

During validation, an error occurs due to the incompatibility rule set for both products.

Scenario 4 - Testing with Multiple Combinations

Procedure

1. Create a hierarchy of products called Offer_A. Make Offer_A ProductComprisedOf SIM. SIM is comprised of SIM_Group, which is ProductComprisedOf Option1, Option2, and Option3.
2. Create a hierarchy of products called Offer_B. Make Offer_A ProductComprisedOf SIM. SIM is comprised of SIM_Group which is ProductComprisedOf Option1, Option2, and Option3.



3. Set the following rules for the products:
 - Offer_A incompatible with Offer_B
 - Option1 incompatible with Option3
 - Option2 incompatible with Option3
 - Option3 incompatible with Option_C
4. Add to your order Offer_A with two Option1, one Option2, and one Option3.
5. Add to your order Offer_B with OptionA, OptionB, and OptionC.

Result

The order returns an error because it is not eligible due to the following constraints:

- Option1 and Option2 are incompatible with Option3.
- Offer_A is incompatible with Offer_B.
- Option_C is incompatible with Option3.

Scenario 5 - Testing with Required UDF Input

Send a validation request when UDF input is required.

Procedure

1. Create a product in which you have specified that the input characteristic gender must be passed in the request.
2. Set the gender characteristic to active.
3. Set the flag `validateData = true` in the offer validation request.
4. Add the product to your basket without defining the gender.

Result

You get an error in the reply during validation when that product is in your basket and the flag `validateData = true` without the UDF gender entry.

Scenario 6 - Testing with ProductRequiredFor with a Group and One Element Passed

Send a validation request with a product group and one element that is not passed.

Procedure

1. Create a group that has the restriction of minimum and the maximum number of one and `ProductComprisedOf` two products: Phone and SIM_Card.
2. Make the phone product have a `ProductRequiredFor` relationship (`ocvValidationReq=true`) for the products Data1 and Data2.
3. In the relationship between Data1 and Data2, define the maximum number of one. (Also, the group has the restriction of minimum and maximum number of one)
4. Add phone, Data1, and Data2 to your basket.

Result

During validation when you add Data1 and Data2 products to your basket, an error occurs because the maximum number is defined as one for the group.

Scenario 7 - Testing ProductRequiredFor with No Group and One Element Not Passed

Send a validation request with no group and one element that is not passed.

Procedure

1. Create a product bundle that has ProductComprisedOf Phone1.
2. Set Phone1 ProductComprisedOf SIM_Card.
3. Set SIM_Card to have ProductRequiredFor Data_Plan.
4. Set Data_Plan with a restriction of minimum and the maximum number of one (ocvValidationReq=false).
5. Add Data_Plan and SIM_Card to your basket.

Result

During validation, when you add both products (the product SIM_Card and Data_Plan) to your basket, an error occurs because the minimum and maximum number are defined as one for Data_Plan, which was already added to the basket from the product SIM_Card.

Scenario 8 - Testing LinkDefinitions with Restrictions with Maximum Number of Products

Send a validation request with LinkDefinitions with a maximum number of product restrictions.

Procedure

1. Create a parent product called Parent1 with ProductComprisedOf a child product called Child1.
2. Define a record level restriction for Child1 of the maximum number of one.
3. Add group-level restriction of the maximum number of one.
4. In the relationship ProductComprisedOf, define a LinkDefinitions value composed by name and address.

5. Add two pairs of one Parent1 and one Child1 with different values for name and address UDFs to the basket.

Result

This validation is successful because each pair has a maximum of one Child1 and defines the name and address for both products.

Scenario 9 - Testing LinkDefinitions with Restrictions with Maximum Number of Products in a Group

Send a validation request with LinkDefinitions with a maximum number of products in a group restriction.

Procedure

1. Create a parent product called Parent1 that ProductComprisedOf a child product called Child1.
2. Define a record level restriction for Child1 of the maximum number of two.
3. Add a group level restriction of the maximum number of one.
4. In the relationship ProductComprisedOf, define a LinkDefinitions value composed by name and address.
5. Add one Parent1 and two Child1 with the same values for name and address UDFs to the basket.

Result

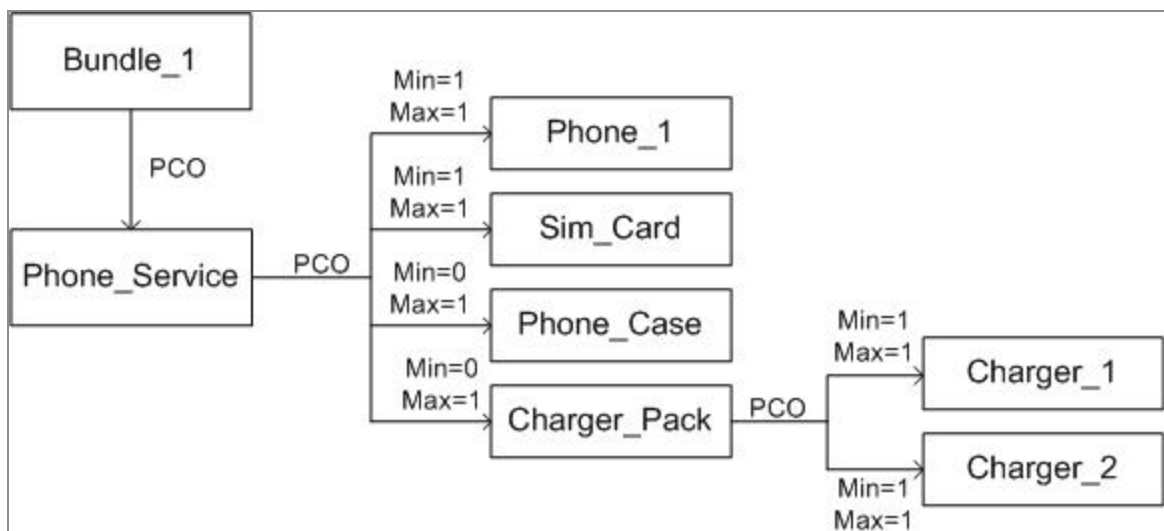
This validation is unsuccessful because Parent1 has two Child1 associated due to the same name and address UDF values, while just one child per group is allowed.

Scenario 10 - Testing with Multiple Minimum Number of Products in a Group Not Reached

Send a validation request when multiple products with a required minimum number are not reached in a group.

Procedure

1. Create a hierarchy of products within a group. Name the group Bundle_1. Make Bundle_1 ProductComprisedOf Phone_Service with AUTOPROVISION=false and with a restriction of minimum and maximum number one.
2. Make the Phone_Service ProductComprisedOf of the following four products:
 - a. Phone_1 with a restriction of minimum and the maximum number of one.
 - b. Sim_Card with a restriction of minimum and the maximum number of one.
 - c. Phone_Case with a restriction of a minimum of zero and a maximum of one.
 - d. Charger_Pack with a restriction of a minimum of zero and a maximum of one.
3. Make the Charger_Pack ProductComprisedOf of the following two products:
 - a. Charger_1 with a restriction of minimum and the maximum number of one.
 - b. Charger_2 with a restriction of minimum and the maximum number of one.



4. Add Bundle_1 to your basket without including the Phone_Service, Phone_1, or the Sim_card.

Result

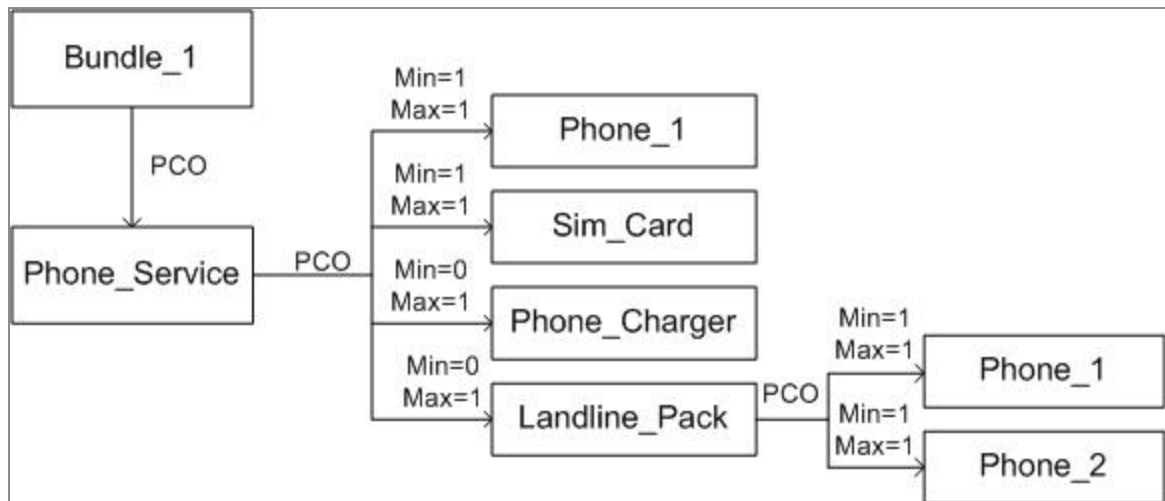
During validation, a cascade of validateOffer errors occurs for the missing Phone_Service, Phone_1, and sim_card. With the same hierarchy of products, if you include Bundle_1 in your basket along with Phone_Service, Phone_1, sim_card, and the Charger_Pack, without Charger_1 and Charger_2, a cascade of validation errors occur for the missing Charger_1 and the missing Charger_2.

Scenario 11 - Testing with Products of the Same Name but Different Parents

Send a validation request with two products that have the same name but different parents.

Procedure

1. Create a hierarchy of products within a group. Name the group Bundle_1. Make Bundle_1 ProductComprisedOf Phone_Service with AUTOPROVISION=false and with a restriction of minimum and maximum number one.
2. Make the Phone_Service ProductComprisedOf of the following four products:
 - a. Phone_1 with a restriction of minimum and the maximum number of one.
 - b. Sim_Card with a restriction of minimum and the maximum number of one.
 - c. Phone_Charger with a restriction of a minimum of zero and a maximum of one.
 - d. Landline_Pack with a restriction of a minimum of zero and a maximum of one.
3. Make the Landline_Pack ProductComprisedOf of the following two products:
 - a. Phone_1 with a restriction of minimum and the maximum number of one.
 - b. Phone_2 with a restriction of minimum and the maximum number of one.



4. Add Bundle_1 to your basket with Phone_Service, Phone_Charger, Landline_Pack, and Phone_2.

Result

During validation, a cascade of validateOffer errors occurs for Phone_1 with Phone_Service parent, Sim_Card with Phone_Service parent, and Phone_1 with Landline_Pack parent.

Best Practices

Eligibility for Offer and Price Engine

The following models of eligibility are supported:

- Segment eligibility

Segment eligibility is configured in the product catalog. Each product is configured with one too many segments. Eligibility is determined by specifying a list of segments on the request message. For each segment specified in the request, there must be an exact match on the product for it to be returned. The product might have additional segments and still be eligible. The list of products matching the specified segments is then returned, adjusted by the specified filters. This method is suitable for all TIBCO OPE environments. It also reduces the number of relationships to be configured in the catalog.

- Filters

The use of filters whenever applicable is encouraged. Using filters has the following advantages:

- All commercial products are not evaluated for offerings and prices.
- Increases performance, because only a limited set of offers is evaluated.

Service Response Time

Improve the response of the GetOffer service with product information size by returning only the required product characteristics by specifying them in the request. For more information, see **Get Offer Service** in *TIBCO® Offer and Price Engine Web Services Guide*.

Model Processing Time Verification for Offline vs Online Model Loading

Data models can be loaded for TIBCO Offer and Price Engine using either offline or online loading.

In the offline model loading process, generated models are copied to a file system and the model XML files are moved either to the `success` or `failure` folders depending on the outcome of the model processing.

In the online model loading process, models are first posted on TIBCO EMS queues, and TIBCO Offer and Price Engine reads models from those EMS queues.

In the offline model loading process, the outcome of the model processing can be verified by looking at the `success` or `failure` folders. As a result, production tasks such as calculating model processing time can be completed without any additional changes. For online model loading, custom code has to be written based on the correlation ID to identify the model processing rate.

Tuning Parameters

Use the following tuning parameters to improve the performance of TIBCO Offer and Price Engine.

Global Cache

In the JVM memory cache, the models required for the start action are retained. The rest of the models are fetched from a relational database or Redis.

In TIBCO Offer and Price Engine, for each model, such as rule model, category model, a dedicated cache is provided. Global cache has an expiry period, and initially, it is empty. The following properties are associated with each global cache:

- Number of models to be cached
- Global cache expiry period (**Default:** 30 seconds)
- Default value setting (true/false)

Improve the performance of the engine by loading the required products in memory.

To load all the products in memory, the following flag starting with enabled must be set to true (it's disabled by default) in ConfigValues_OPE.xml under <Category description="Catalog Store Configuration for OPES" name="Catalog Store Configuration for OPES" visibility="Basic"> category. Also, you need to set the number of models you want to store in the cache and cache expiry period property accordingly.

Property Name	Default Value
com.tibco.fos.model.cacheType.cache.maxNoProductcached	1000
enableProductModelGlobalCache	false
productCacheExpiryPeriod	30
com.tibco.af.opes.cacheType.cache.maxNoPriceCached	1000

Property Name	Default Value
enablePriceModelGlobalCache	false
priceCacheExpiryPeriod	30
com.tibco.af.opes.cacheType.cache.maxNoDiscountCached	1000
enableDiscountModelGlobalCache	false
discountCacheExpiryPeriod	30
com.tibco.af.opes.cacheType.cache.maxNoRuleCached	1000
enableRuleModelGlobalCache	false
ruleCacheExpiryPeriod	30
com.tibco.af.opes.cacheType.cache.maxNoCategoryCached	1000
enableCategoryModelGlobalCache	false
categoryCacheExpiryPeriod	30
com.tibco.af.opes.cacheType.cache.maxNoOfferIdsCached	1000
enableOfferIdsModelGlobalCache	false
offerIdsCacheExpiryPeriod	30
com.tibco.af.opes.cacheType.cache.maxNoFilterCached	1000
enableFilterProductGlobalCache	false
filterProductCacheExpiryPeriod	30
com.tibco.af.opes.cacheType.cache.maxNoSegmentCached	1000
enableSegmentProductGlobalCache	false

Property Name	Default Value
segmentProductCacheExpiryPeriod	30
com.tibco.fos.model.cacheType.cache.maxNoTopLevelProductsCached	1000
enableTopLevelProductsGlobalCache	false
topLevelProductsCacheExpiryPeriod	30

Response Caching

To cache the response, the hash for each request is calculated. Any type of unique element (for example, the `businesstransactionid`) that you add, is treated as a new request by the OPE engine. Therefore, if you want to use response caching, you must eliminate using all the unique elements in the requests.

Improve the performance of the engine by loading the required responses in memory.

To use Response Caching, the following flag starting with `enabled` must be set to `true` (it's disabled by default) in `ConfigValues_OPE.xml` under `<Category description="Api Response Cache Configuration for OPES" name="Api Response Cache Configuration for OPES" visibility="Basic">` category. Also, set the number of responses for a particular API and expiry period of the cache property accordingly.

Property Name	Default Value
enableOpeApisResponseCache	true
getOfferResponseCacheExpiryPeriod	30
com.tibco.af.opes.cache.maxNoGetOfferResponseCached	1000
validateOfferResponseCacheExpiryPeriod	30
com.tibco.af.opes.cache.maxNoValidateOfferResponseCached	1000
getProdInfoResponseCacheExpiryPeriod	30
com.tibco.af.opes.cache.maxNoGetProdInfoResponseCached	1000

Property Name	Default Value
getPricesResponseCacheExpiryPeriod	30
com.tibco.af.opes.cache.maxNoGetPricesResponseCached	1000
getPriceInfoResponseCacheExpiryPeriod	30
com.tibco.af.opes.cache.maxNoGetPriceInfoResponseCached	1000
getCategoryResponseCacheExpiryPeriod	30
com.tibco.af.opes.cache.maxNoGetCategoryResponseCached	1000

Garbage Collection Recommendations

- **Use G1GC (-XX:+UseG1GC)**

G1GC is the default garbage collection algorithm for JDK 9 and later.

In production TIBCO Offer and Price Engine holds a large amount of model data and related business objects in memory. Due to the generation of large request and response objects, applications might warrant a large amount of garbage collection.

TIBCO Offer and Price Engine recommends you use G1GC as this divides the heap into a set of regions. Most GC operations can then be performed a region at a time rather than on the entire Java heap or an entire generation thus improving the garbage collection process.

- **Set Total Heap (-Xms, -Xmx)**

The important factor affecting garbage collection performance is total available memory. Because collections occur when generations fill up, throughput is inversely proportional to the amount of memory available

TIBCO Offer and Price Engine recommends using the same value for Xms and Xmx flags

- **Setting Young Generation Sizing (-XX:NewRatio or -XX:NewSize and -XX:MaxNewSize)**

After total available memory, the second most influential factor affecting garbage

collection performance is the proportion of the heap dedicated to the young generation.

The value of the young generation must be at least a third of the total size of the heap i.e. ratio of the young generation to the old generation must be at least 1:3. If performance tests show too many minor GCs, this size can be increased further.

- **String Deduplication (-XX:+UseStringDeduplication)**


Strings consume a lot of memory in any application. With Oracle Java 8 Update 20, G1 introduced a string deduplication feature where a hashcode of a string is stored along with a weak reference to its character array. When it finds another string with the same hashcode, one string is modified to point to the char array of the second string.

OPE recommends you use this flag.

- **Maximum Garbage Collection pause time (-XX:MaxGCPauseMillis)**

MaxGCPauseMillis sets a target for the maximum GC pause time.

TIBCO Offer and Price Engine recommends you use this flag.

 **Note:** A very low value for this flag inversely affects throughput.

- **Number of threads for parallel and concurrent garbage collection (-XX:ParallelGCThreads, -XX:ConcGCThreads)**

ParallelGCThreads sets the number of threads used during parallel phases of the garbage collectors and ConcGCThreads sets the number of threads used for concurrent garbage collection. For both, the default value varies with the platform on which the JVM is running.

- **Other Garbage Collection flags**

- **Print all Garbage Collection logs in a separate log file**

```
-XX:+PrintGC -XX:+PrintGCDetails -XX:+PrintGCApplicationStoppedTime
-XX:+PrintGCApplicationConcurrentTime -XX:+PrintAdaptiveSizePolicy
-XX:+PrintTenuringDistribution -XX:+PrintGCTimeStamps
-XX:+PrintGCDateStamps
```

- **Print heap dump**

```
-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath
```

- **Rolling Garbage Collection logs**

- `-Xloggc -XX:+UseGCLogFileRotation -XX:GCLogFileSize`

- **Minimal Kernel Time**

Garbage Collection pauses are affected by kernel CPU utilization time. Garbage Collection logs display information about kernel time along with user time (application Garbage Collection time). High amounts of kernel time can cause longer Garbage Collection pauses as application JVM competes with other application(s) running on the same host for the core computing resources.

As part of production deployment, Garbage Collection needs to make sure that the host machine is not running too many resource-intensive applications along with TIBCO Offer and Price Engine.

TIBCO Documentation and Support Services

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

How to Access TIBCO Documentation

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

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

Product-Specific Documentation

The following documentation for this product is available on the [TIBCO® Offer and Price Engine](#) documentation page:

- *TIBCO® Offer and Price Engine Release Notes*
- *TIBCO® Offer and Price Engine Installation and Configuration Guide*
- *TIBCO® Offer and Price Engine Concepts Guide*
- *TIBCO® Offer and Price Engine User Guide*
- *TIBCO® Offer and Price Engine Web services Guide*
- *TIBCO® Offer and Price Engine Security Guidelines*

How to Contact TIBCO Support

Get an overview of [TIBCO Support](#). You can contact TIBCO Support in the following ways:

- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the [TIBCO Support](#) website.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to [TIBCO Support](#) website. If you do not have a user name, you can request one by clicking **Register** on

the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the [TIBCO Ideas Portal](#). For a free registration, go to [TIBCO Community](#).

Legal and Third-Party Notices

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

TIBCO, the TIBCO logo, and the TIBCO O logo are either registered trademarks or trademarks of Cloud Software Group, Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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

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

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. CLOUD SOFTWARE GROUP, 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.

This and other products of Cloud Software Group, Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (<https://www.tibco.com/patents>) for details.

Copyright © 2010-2023. Cloud Software Group, Inc. All Rights Reserved.