TIBCO® Fulfillment Catalog User's Guide

Software Release 4.0
July 2017

Two-Second Advantage®

Important Information

Contents

Figures

TIBCO Documentation and Support Services

Documentation for this and other TIBCO products is available on the TIBCO Documentation site. This site is updated more frequently than any documentation that might be included with the product. To ensure that you are accessing the latest available help topics, visit:

https://docs.tibco.com

Product-Specific Documentation

The following documents for this product can be found on the TIBCO Documentation site:

- *TIBCO Fulfillment Catalog Installation Guide*
- *TIBCO Fulfillment Catalog Product Catalog Guide*
- *TIBCO Fulfillment Catalog User's Guide*
- *TIBCO Fulfillment Catalog Web Services Guide*
- *TIBCO Fulfillment Catalog Offer and Price Designer User's Guide*
- *TIBCO Fulfillment Catalog Release Notes*

How to Contact TIBCO Support

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

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:

  http://www.tibco.com/services/support

- If you already have a valid maintenance or support contract, visit this site:

  https://support.tibco.com

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

How to Join TIBCO Community

TIBCO Community is an online destination for TIBCO customers, partners, and resident experts. It is a place to share and access the collective experience of the TIBCO community. TIBCO Community offers forums, blogs, and access to a variety of resources. To register, go to the following web address:

https://community.tibco.com

TIBCO Fulfillment Catalog Overview

TIBCO® Fulfillment Catalog is a plug-in for TIBCO MDM that enables you to easily manage and maintain complex Product Offerings.

The Product Offerings, Services and Rules for Pricing, Provisioning and Eligibility are actively maintained within the TIBCO® Fulfillment Catalog's Data Repository. For details on the product catalog and data models, see the *Product Catalog* documentation.

A complete hierarchical interface allows you to create, edit, search, and maintain data. User access, responsibilities, and lifecycle workflow can be configured for the ongoing management of this information. The complete data model for your product offerings and their components can be exported to the downstream order provisioning systems, such as TIBCO® Fulfillment Order Management.

This document describes the features of TIBCO® Fulfillment Catalog. For detailed information about User Management and Basic Record Management, see *TIBCO MDM User's Guide*.

PRODUCT Association with PLANFRAGMENT

A product can be associated with Plan Fragments for different actions using the following relationships:

- ProductHasProvidePlanFragment
- ProductHasUpdatePlanFragment
- ProductHasCeasePlanFragment
- ProductHasCancelPlanFragment
- ProductHasCustomPlanfragment

Modeling the MILESTONE and PLANFRAGMENT in Fulfillment Catalog

Perform the following steps to model the Products, PlanFragments, or Milestones:

Procedure

> Creation of milestones are not mandatory for the creation of a plan fragment from FC 3.0.0 onwards. To create plan fragments without milestones, see Modeling the PLANFRAGMENT and PRODUCT in Fulfillment Catalog. The creation of milestones is relevant only if intermediate milestones need to be created for a plan fragment.

1. Create the **Milestone** records.

   > Create separate Milestones with Milestone names as `START` and `END` respectively, in addition to any intermediate milestones for association with a plan fragment. While creating the `START` milestone, create the `END` milestone using `MilestoneToMilestone` relationship.

   > There is no fixed nomenclature for Milestone ID but Milestone names must be `START` and `END`.

2. Create the **PlanFragment** records and associate the Milestones created earlier using the `PlanFragmentHasMilestone` relationship appropriately.
3. Decide the sequence of milestones in a plan fragment by creating the `MilestoneToMilestone` relationship between the associated milestones in a plan fragment. `MilestoneToMilestone` relationship can only be created between the milestones associated with the same plan fragment.

The MilestoneToMilestone relationship must be modeled between all possible combinations of milestone pairs that may appear in the plan item in the execution plan generated by AOPD. Although the product model can have any number of milestones defined, the actual plan item may contain only the subset of these milestones due to the dependency modeling and the products being ordered.

Assume that PF_PROVIDE is a plan fragment associated to a product P1 for PROVIDE action in the product model. Also, it contains four milestones namely START, M1, M2, and END. Now, based on the dependencies modeled and the products ordered, one of the three milestone combinations given below can come into the plan item generated for fulfillment of P1.

- START, M1, END
- START, M2, END
- START, M1, M2, END

In order to support any of the three milestone combinations mentioned above, the MilestoneToMilestone relationships must be modeled for the following milestone pairs so as to have the corresponding sections in the plan fragment model.

- START->M1
- START-M2
- M1->M2
- M1->END
- M2->END

While processing this plan item in the execution plan reply, Orchestrator will try to find the required plan fragment sections in order to sort the milestones in a proper sequence based on the typical duration value. This is done so that the plan item can be properly represented on OMS UI Gantt chart. If any of the required section is missing in the plan fragment, Orchestrator will fail with an exception to process the execution plan.

For example, in the case of the START-M1-M2-END combination, the sequence of M1 and M2 after START is decided on the basis of the typical durations for START->M1 and START->M2. If START->M1 is 2000 ms and START->M2 is 1000 ms, milestone M1 will be sequenced after M2. So the sequence will be START->M2->M1->END.

4. Decide the dependencies between the milestones of two separate plan fragments by creating the `MilestoneDependsOn` relationship between the associated milestones in the plan fragments. `MilestoneDependsOn` relationship can only be created between the milestones associated with different plan fragments.
5. Create the Product record and associate the plan fragments for four actions using four different `ProductHas[*Action*]PlanFragment` relationships. For instance, `ProductHasProvidePlanFragment`.
6. You can also associate the newly created plan fragments to the existing products by using the relationships explained in point 5. However, you will have to remove the plan fragment attributes specified in INTERNAL and AFFINITY tabs.

Modeling the PLANFRAGMENT and PRODUCT in Fulfillment Catalog

In the new version of Fulfillment Catalog, you can model a plan fragment and link it to a product without creating milestones. Perform the following steps to model the products/plan fragments:

Procedure

1. Create the **PlanFragment** records.

2. Enter the values for the fields **Typical Duration** and **Maximum Duration** in the **SLA** tab during Plan Fragment creation. This will allow you to create a Plan Fragment without the necessity of creating associated milestones.

   > Although you can skip the step of creation of milestones using the fields **Typical Duration** and **Maximum Duration** in the **SLA** tab, but during publishing the products the START and END milestones will be automatically added.

3. Create the Product record and associate the plan fragments.
4. You can also associate the newly created plan fragments to the existing products by using the relationships. However, you will have to remove the plan fragment attributes specified in INTERNAL and AFFINITY tabs.

ProductHasCustomPlanfragment Relationship

The `ProductHasCustomPlanfragment` is a new relationship added in the PRODUCT repository to model the products having custom actions defined in the PLANFRAGMENT repository, other than the PROVIDE, CEASE, UPDATE, and CANCEL actions

The `ProductHasCustomPlanFragment` relationship is associated with the PRODUCT repository having target repository as PLANFRAGMENT. The `ActionIDs` must be selected from the ACTION repositories.

Hierarchy Management

You can use the Hierarchy Management utility to:

- View existing data for each repository.
- View existing relationship model for a given record in any repository. You can drill down till the last level of relationship.
- Manipulate multiple relationship tree by selecting from dropdown and then save them all at one.
- Modify record attribute or attributes of existing record.
- Modify relationship attribute or attributes of existing relationship.
- Delete existing relationship.
- Add new relationship between two records (same repository or different repositories).
- Create a new record and use it on the canvas.
- View record details on the right pane.
- Cache data at client side, which results in improved performance of the overall utility.

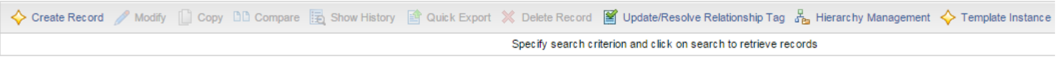
Changes in Hierarchy Management

In the earlier version of Fulfillment Catalog the Hierarchy Management module could be accessed using the options **Hierarchy View** and **Hierarchy Edit** which were available on the **Browse and Search** screen. If you did not have modify record or relationship permission you could still get into the **Hierarchy Management** through the **Hierarchy View** option and then toggle to **Edit** mode and gain access to modify record or relationship attributes.

Hierarchy Management screens were available only for the PRODUCT repository. This meant that only a limited set of data could be managed through the screens.

The two menu options **Hierarchy View** and **Hierarchy Edit** have been consolidated into one **Hierarchy Management**, which is available on the **Browse and Search** screen and also for all the repositories.

Update/Resolve Relationship Tag Button

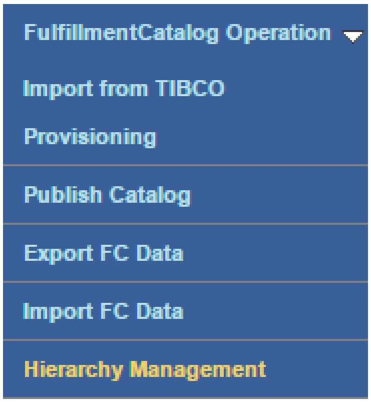


Specify search criterion and click on search to retrieve records

The idea of merging them into one is to provide an entry to the screens and then control the permissions for various actions as per the permissions available to the logged-in user. For example, if the user has the permission to modify the record, the menu would be **Modify Record Attribute**. However, if the user does not have the permission to modify the record, the same menu changes to **View Record Attribute**.

The **Hierarchy Management** screens can be accessed through **Hierarchy Management** submenu available within the menu **FulfillmentCatalog Operation**.

Hierarchy Management Menu Item



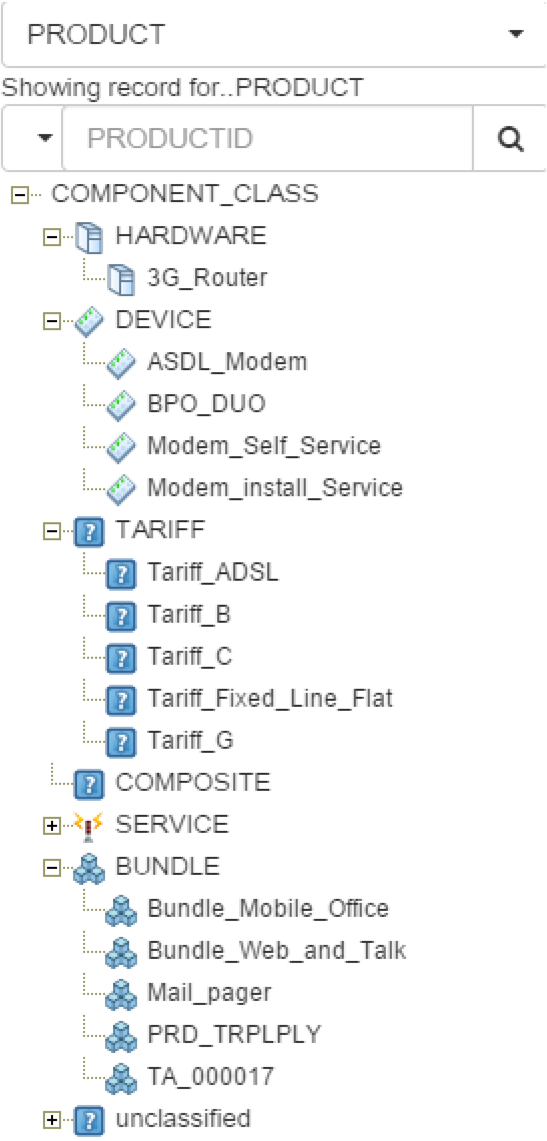
Structure of Classification Tree Panel

The classification panel contains three sections:

Repository Dropdown

It contains a list of all the repositories that are present. You can select a repository for which you wants to see the records. Repository Dropdown will be enabled only when canvas is blank.

Hierarchy Management Repository Dropdown Menu



PRODUCT

Showing record for..PRODUCT

PRODUCTID

- COMPONENT_CLASS
  - HARDWARE
    - 3G_Router
  - DEVICE
    - ASDL_Modem
    - BPO_DUO
    - Modem_Self_Service
    - Modem_install_Service
  - TARIFF
    - Tariff_ADSL
    - Tariff_B
    - Tariff_C
    - Tariff_Fixed_Line_Flat
    - Tariff_G
  - COMPOSITE
  - SERVICE
  - BUNDLE
    - Bundle_Mobile_Office
    - Bundle_Web_and_Talk
    - Mail_pager
    - PRD_TRPLPLY
    - TA_000017
  - unclassified

Search Option

It will help you search for a specific record using a record ID in the classification tree. You get the following choices for searching a record:

- **Exact Search**: Searches all records with record ID exactly matching to a search text.
- **Similar Search**: Searches all records with record ID containing a search text. Similar Search is default search option.

Classification Tree

Gives a tree representation of records for a repository selected in the repository dropdown. Records are classified based on the classification scheme defined for that repository.

Creating a New Record

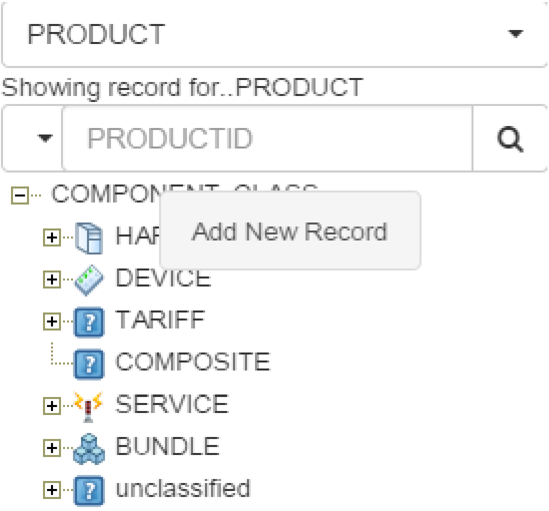
For creating a new record, a context menu is provided. This option can be seen when you right click either the classification scheme node or the classification code node in the classification tree.

Procedure

If you have the permission to create a record, the context menu will be enabled, otherwise it would be disabled.

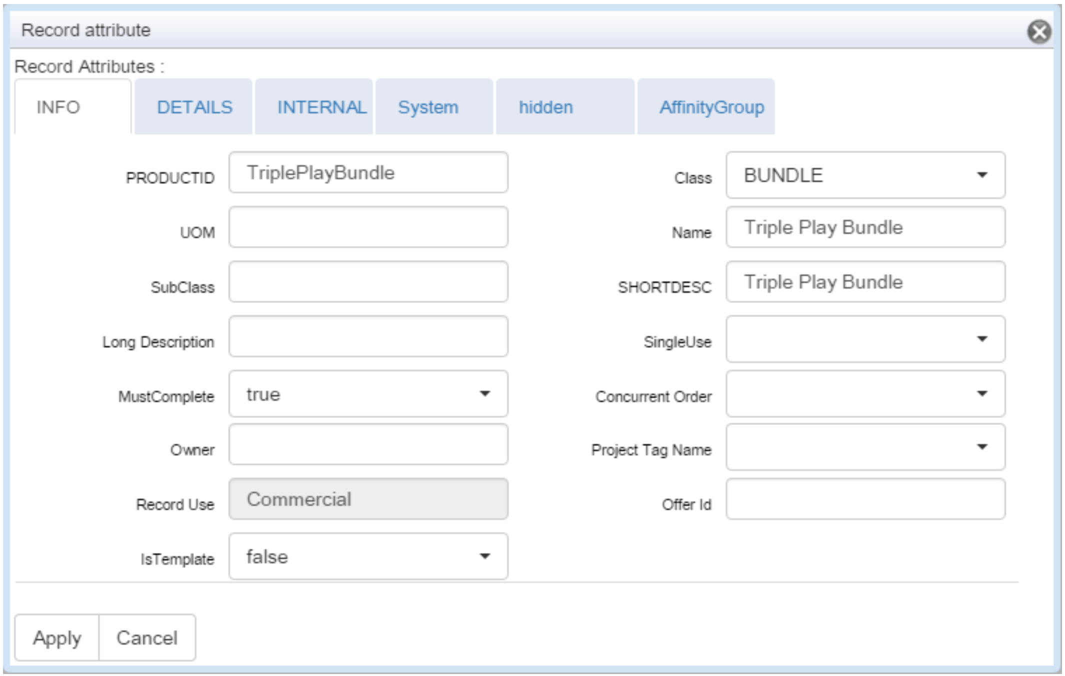
1. Click **Add New Record** menu. A record attribute panel will be opened to create a new record.

   Hierarchy Management Add New Record

2. After adding record attributes and applying the changes, the record will be created and the classification tree will get updated with the new record. After the necessary information is added click the **Apply** button.

   Hierarchy Management Record Attributes

PRODUCTID	TriplePlayBundle	Class	BUNDLE
UOM		Name	Triple Play Bundle
SubClass		SHORTDESC	Triple Play Bundle
Long Description		SingleUse	
MustComplete	true	Concurrent Order	
Owner		Project Tag Name	
Record Use	Commercial	Offer Id	
IsTemplate	false		

   The record is added in the classification tree under a specific classification code as defined by Class attribute. If no classification code defined, the record is added under unclassified.

> The new record is not saved into the database yet. It is only for the purpose of using it on the canvas.

3. To save a newly created record in database, you will have to drag an drop the record on the canvas and then click the **Save and Process** button.

Canvas Toolbar

Canvas toolbar contains few options to control the behavior and data on the canvas. The Canvas Toolbar contains following options:

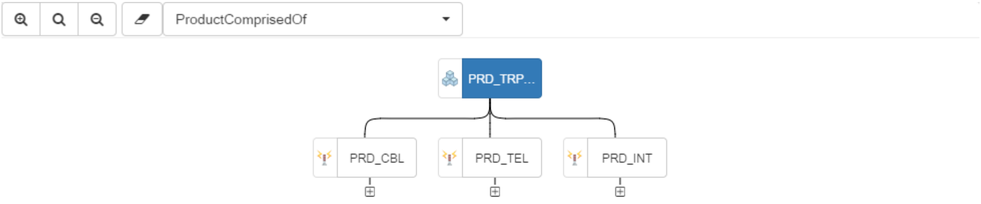
- **Zoom In, Out or Reset** buttons: Used to enlarge or reduce the size of boxes on the canvas for easy navigation through the hierarchy. Zoom panel also contains an extra button Reset Zoom, which sets the zoom to its original level.
- **Clear** Button: The **Clear** button will help to clear the data on canvas. After clearing the canvas, the repository dropdown from classification tree panel will be enabled and the user will be able to select a record of specific repository. This could be used when you want to change the root record.
- **Relationship** Dropdown: The list of relationships in the dropdown consists of Forward as well as Reverse relationship(s), for a given repository. Additionally the list is controlled through a corresponding configurable property `com.tibco.fc.hm.<repository_name>.relationships` in the `ConfigValues.xml` file. For example, the property name for the PRODUCT repository will be `com.tibco.fc.hm.product.relationships`. Through this property, the Relationship dropdown can also be controlled to show none, some, or all of the Forward and Reverse relationships. The following are the scenarios:

    - If the `com.tibco.fc.hm.<repository_name>.relationships` property is missing for a repository, then the dropdown will have all the forward relationships as well as all the reverse relationships.
    - If the `com.tibco.fc.hm.<repository_name>.relationships` property exists, then the mentioned valid relationships (comma separated, forward and/or reverse) in the `com.tibco.fc.hm.<repository_name>.relationships` property will be displayed in the dropdown.

    Even the default relationship for which the canvas will be created, when a record is dragged and dropped onto the canvas, can be controlled using the property `com.tibco.fc.hm.<REPOSITORY_NAME>.defaultrelationship` in `ConfigValues.xml`. You can select any relationship for which you want to see the related records of root record.

The following diagram gives a graphical representation of a root record and its child records, which are related through relationship selected in relationship dropdown i.e. PlanFragmentHasMilestone:

Hierarchy Management Root Record and Child Records



Canvas Panel

Canvas contains a graphical representation of a record.

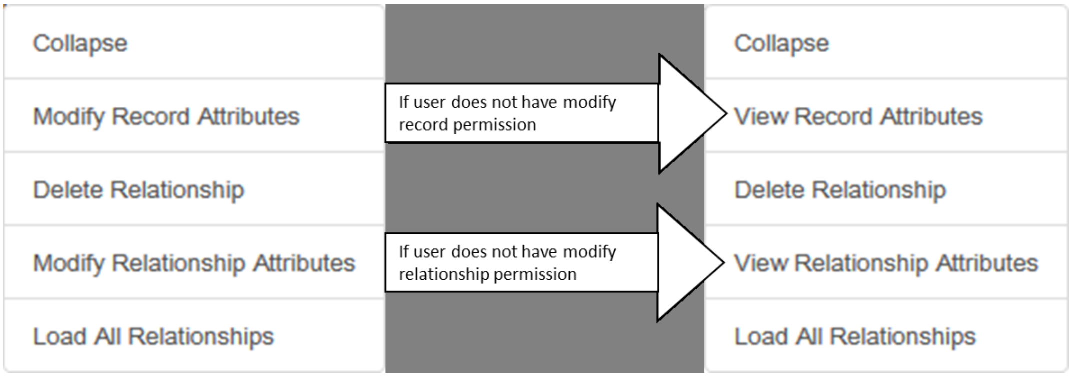
The functioning of the Canvas has changed. See Dynamic Context Menu topic for more details.

Dynamic Context Menu

The context menu on any node in canvas changes according to the privileges given to the logged-in user.

The following image shows how a menu changes if user lacks the necessary privileges:

Hierarchy Management Dynamic Popup Menu



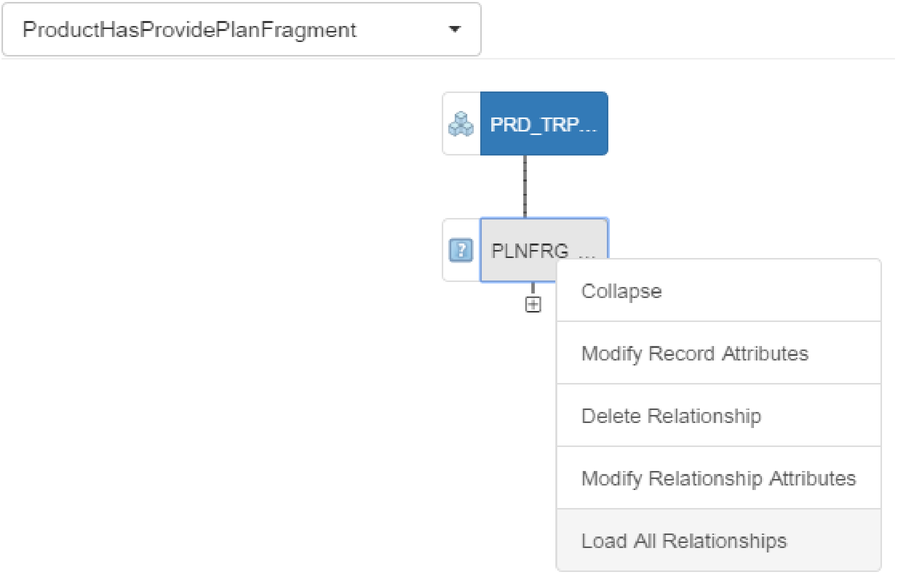
If user does not have modify record permission

If user does not have modify relationship permission

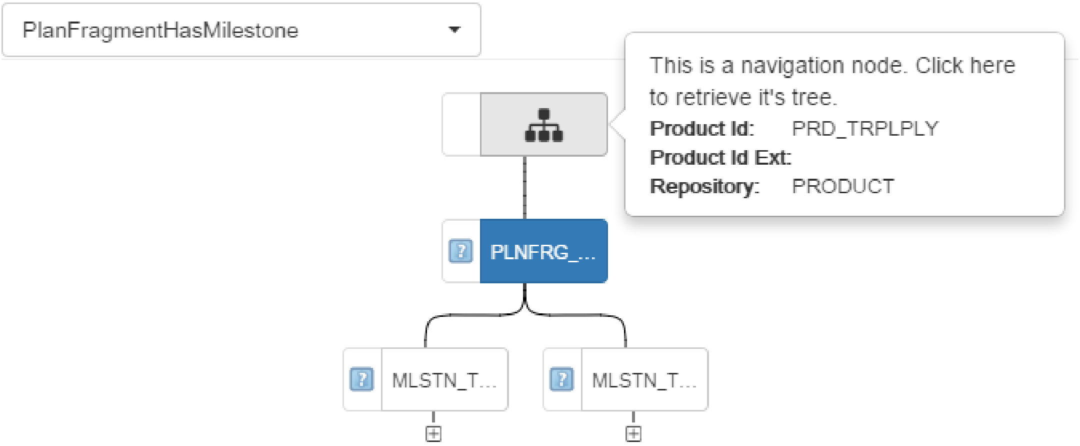
If the user does not have the necessary privileges, the **Delete Relationship Menu** will be disabled.

Collapse	Using this menu, collapses all the children of the node for which the context menu was used.
Modify Record Attributes	This menu is used to open the record detail screen in edit mode to modify one or more attributes. Clicking **Apply** button only saves the changes in memory and not in the database. The changes would reflect in database upon successful execution of **Save and Process**.
Delete Relationship	This is used to delete the relationship for the selected child from its parent.
Modify Relationship Attributes	This menu is used to open the relationship attribute detail screen in edit mode to modify one or more attributes. Clicking **Apply** button only saves the changes in memory and not in the database. The changes would reflect in database upon successful execution of **Save and Process**.
Load All Relationships	This menu is used to drill down relationships of the selected node. Example: In a hierarchy of `ProductHasProvidePlanFragment`, you can view relationships of the planfragment record by clicking **Load All Relationships** for the planfragment node.

Hierarchy Load All Relationships Part 1



Hierarchy Load All Relationships Part 2



This is a navigation node. Click here to retrieve it's tree.

Product Id: PRD_TRPLPLY

Product Id Ext:

Repository: PRODUCT

Based on the displayed screenshots, if you select the menu **Load All Relationships** for child record 'Logistica', which is of planfragment repository, it gives a graphical representation of 'Logistica' for selected relationship `PlanFragmentHasMilestone`.

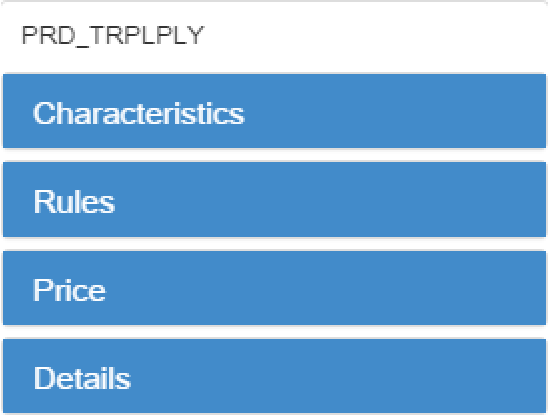
This tree will also contain a **Navigation Node**, which is a path, to get back to the tree of a parent node of a child.

Properties Panel

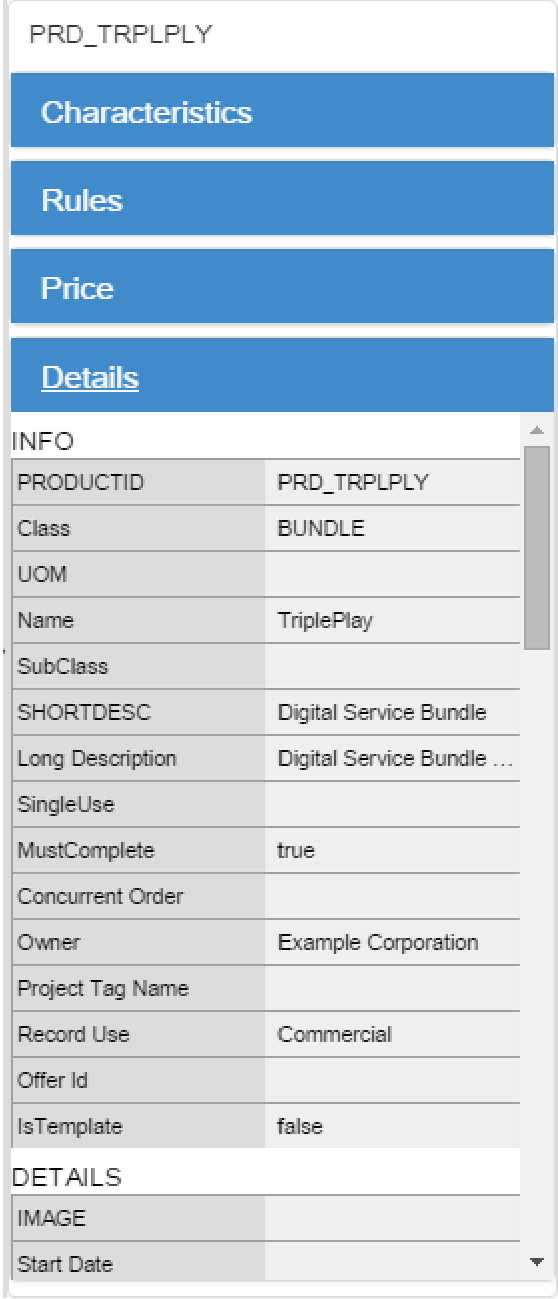
The properties panel is the same as the earlier version but has an improved user interface.

For the PRODUCT repository, the properties panel will have four tabs: Characteristics, Rules, Price, and Details.

Properties Panel for the PRODUCT Repository



Properties Panel with Details for the PRODUCT Repository



INFO

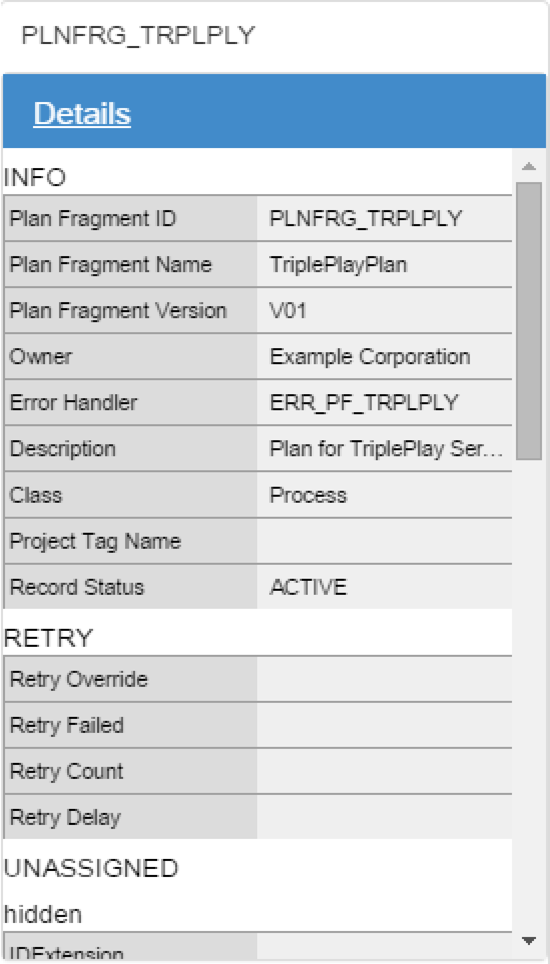
PRODUCTID	PRD_TRPLPLY
Class	BUNDLE
UOM	
Name	TriplePlay
SubClass	
SHORTDESC	Digital Service Bundle
Long Description	Digital Service Bundle ...
SingleUse	
MustComplete	true
Concurrent Order	
Owner	Example Corporation
Project Tag Name	
Record Use	Commercial
Offer Id	
IsTemplate	false

DETAILS

IMAGE	
Start Date	

For all other repositories, the properties panel will have a single tab called Details which gives the record attributes details.

Properties Panel with Details for Other Repositories



PLNFRG_TRPLPLY

Details

INFO

Plan Fragment ID	PLNFRG_TRPLPLY
Plan Fragment Name	TriplePlayPlan
Plan Fragment Version	V01
Owner	Example Corporation
Error Handler	ERR_PF_TRPLPLY
Description	Plan for TriplePlay Ser…
Class	Process
Project Tag Name	
Record Status	ACTIVE

RETRY

Retry Override	
Retry Failed	
Retry Count	
Retry Delay	

UNASSIGNED

hidden

IDExtension

Publish Catalog

TIBCO® Fulfillment Catalog Model Publisher allows you to publish a full model, or an incremental model, to a predefined TIBCO Enterprise Message Service™ topic, or to a file based on the appropriate configuration.

The model publish feature is available through GUI as well as through webservice. To use webservice, you must send a web service request which are different for different operations. See Model Publish Operation topic in the *TIBCO Fulfillment Catalog Web Services Guide* for more details.

The Publish Catalog feature uses a workflow in order to make it customizable in case of different behavior requirement. The workflow is detailed in next section.

The schemas are located in the `$AC_HOME/schema` directory. The XML samples for invoking the publish operation are located in the `$AC_HOME/samples/wsrequests/ModelPublish` directory.

Workflow Definition

The new `wfin26bulkmodelpublisherv1.xml` workflow definition is created in the TIBCO Fulfillment Catalog to publish the PRODUCT, CUSTOMER, SEGMENT, PLANFRAGMENT, DISCOUNT, PRICE, and ACTION models. The workflow gets invoked for the full publish and also for the delta publish. The workflow executes the standard EvaluateSubset activity and returns the record list.

Based on the repository name, the `Process<reponame>SubsetData` activity processes the record list and publishes the data model according to the channel name configured in the workflow. The channel name can be JMS, FILE, or BOTH.

Where: `reponame` is the actual repository name with respect to the data model, for example, PRODUCT, PARTY (CUSTOMER), SEGMENT, PLANFRAGMENT, DISCOUNT, PRICE, and ACTION.

The following table lists the custom activity parameters and their description.

ProcessProductSubsetData Parameter

Activity : ProcessProductSubsetData	
Parameter	**Description**
`InRecordList`	Record list received from the `EvaluateSubset` activity.
`recordCount`	Record processed by the `EvaluateSubset` activity.
`ProductModelMap`	Product model XSL file location.
`ProductModelBatchSize`	Complete product model published in batch based on the specified batch size.
`ProductModelCatName`	Logical topic name mapped to the actual topic name in the `ConfigValues.xml` file.
`connShareMode`	Connection share mode value to connect to the TIBCO Enterprise Message Service server.
`transacted`	Boolean value.
`MasterCatalog`	Based on this parameter, creates the model for product, customer, or segment.
`RelationshipName#`	Refers to the relationship name.
`publishAction`	Refers to the parameter value related to header action such as Bulk (for bulk model).
`Channel`	Valid value is either `JMS`, `FILE`, or `BOTH`. `JMS` is used for the Online catalog publication and `FILE` is used for the Offline Catalog publication. For more information on the Offline Catalog, see Customization of Publish Catalog Workflow. When you set `BOTH`, the data is published and the files are created at the same time.
`Metadata Required`	This parameter should be commented if publish model is triggered through a process and not through UI.
`folderLocation`	Actual file location. This is applicable when `FILE` or `BOTH` value is used for the `Channel` parameter.

Activity : ProcessProductSubsetData	
Parameter	**Description**
`logRawModel`	A boolean flag to log raw product model into the `elink.log` file. Raw model XML is generated by process activity before applying stylesheet. Value should be set to `false`, if model is configured for a large number of relationships (for instance, `ProductComprisedOf`) to avoid out of memory error.
`CatalogUseForFilter`	Refers to the catalog that should be published. It can be All, Commercial, or Technical.
`TypeOfPublish`	Refers to the publish type. It can be full publish or delta publish.
`Tenant`	Refers to the TenantId in the header of published model. By default the "enterprise name" appears as the TenantId in the published model. It is not recommended to directly assign the value for this parameter in the workflow as it affects all Model Publish activities for the enterprise.

ProcessCustomerSubsetData Parameter

Activity: ProcessCustomerSubsetData	
Parameter	**Description**
`InRecordList`	Record list received from the `EvaluateSubset` activity.
`recordCount`	Record processed by the `EvaluateSubset` activity.
`CustomerModelMap`	Customer model XSL file location.
`CustomerModelBatchSize`	Complete customer model published in batch based on the specified batch size.
`CustomerModelCatName`	Logical topic name mapped to the actual topic name in the `ConfigValues.xml` file.
`connShareMode`	Connection share mode value to connect to the TIBCO Enterprise Message Service server.
`transacted`	Boolean value.
`MasterCatalog`	Based on this parameter, creates the model for product, customer, or segment.
`RelationshipName#`	Refers to the relationship name.
`publishAction`	Refers to the parameter value related to header action such as Bulk (for bulk model).

Activity: ProcessCustomerSubsetData	
Parameter	**Description**
`Channel`	Valid value is either `JMS`, `FILE`, or `BOTH`. `JMS` is used for the Online catalog publication and `FILE` is used for the Offline Catalog publication. For more information on the Offline Catalog, see Customization of Publish Catalog Workflow. When you set `BOTH`, the data is published and the files are created at the same time.
`Metadata Required`	This parameter should be commented if publish model is triggered through a process and not through UI.
`folderLocation`	Actual file location. This is applicable when `FILE` or `BOTH` value is used for the `Channel` parameter.
`logRawModel`	A boolean flag to log raw customer model into the `elink.log` file. Raw model XML is generated by process activity before applying stylesheet. Value should be set to `false`, if model is configured for a large number of relationships (for instance, `ProductComprisedOf`) to avoid out of memory error.
`TypeOfPublish`	Refers to the publish type. It can be full publish or delta publish.
`Tenant`	Refers to the TenantId in the header of published model. By default the "enterprise name" appears as the TenantId in the published model. It is not recommended to directly assign the value for this parameter in the workflow as it affects all Model Publish activities for the enterprise.

ProcessSegmentSubsetData Parameter

Activity: ProcessSegmentSubsetData	
Parameter	**Description**
`InRecordList`	Record list received from the `EvaluateSubset` activity.
`recordCount`	Record processed by the `EvaluateSubset` activity.
`SegmentModelMap`	Segment model XSL file location.
`SegmentModelBatchSize`	Complete segment model published in batch based on the specified batch size.
`SegmentModelCatName`	Logical topic name mapped to the actual topic name in the `ConfigValues.xml` file.
`connShareMode`	Connection share mode value to connect to the TIBCO Enterprise Message Service server.

Activity: ProcessSegmentSubsetData	
Parameter	**Description**
`transacted`	Boolean value.
`MasterCatalog`	Based on this parameter, creates the model for product, customer, or segment.
`publishAction`	Refers to the parameter value related to header action such as Bulk (for bulk model).
`Channel`	Valid value is either `JMS`, `FILE`, or `BOTH`. `JMS` is used for the Online catalog publication and `FILE` is used for the Offline Catalog publication. For more information on the Offline Catalog, see Customization of Publish Catalog Workflow. When you set `BOTH`, the data is published and the files are created at the same time.
`Metadata Required`	This parameter should be commented if publish model is triggered through a process and not through UI.
`folderLocation`	Actual file location. This is applicable when `FILE` or `BOTH` value is used for the `Channel` parameter.
`logRawModel`	A boolean flag to log raw segment model into `elink.log`. Raw model XML is generated by process activity before applying stylesheet. Value should be set to `false`, if model is configured for a large number of relationships (for instance, `ProductComprisedOf`) to avoid out of memory error.
`TypeOfPublish`	Refers to the publish type. It can be full publish or delta publish.
`Tenant`	Refers to the TenantId in the header of published model. By default the "enterprise name" appears as the TenantId in the published model. It is not recommended to directly assign the value for this parameter in the workflow as it affects all Model Publish activities for the enterprise.

ProcessPlanFragmentSubsetData Parameter

Activity: ProcessPlanFragmentSubsetData	
Parameter	**Description**
`InRecordList`	Record list received from the `EvaluateSubset` activity.
`recordCount`	Record processed by the `EvaluateSubset` activity.
`PlanFragmentModelMap`	Plan Fragment model XSL file location.

Activity: ProcessPlanFragmentSubsetData	
Parameter	**Description**
`PlanFragmentModelBatchSize`	Complete planfragment model published in batch based on the specified batch size.
`PlanFragmentModelCatName`	Logical topic name mapped to the actual topic name in the `ConfigValues.xml` file.
`connShareMode`	Connection share mode value to connect to the TIBCO Enterprise Message Service server.
`transacted`	Boolean value.
`publishAction`	Refers to the parameter value related to header action such as Bulk (for bulk model).
`Channel`	Valid value is either `JMS`, `FILE`, or `BOTH`. `JMS` is used for the Online catalog publication and `FILE` is used for the Offline Catalog publication. For more information on the Offline Catalog, see Customization of Publish Catalog Workflow. When you set `BOTH`, the data is published and the files are created at the same time.
`Metadata Required`	This parameter should be commented if publish model is triggered through a process and not through UI.
`folderLocation`	Actual file location. This is applicable when `FILE` or `BOTH` value is used for the `Channel` parameter.
`logRawModel`	A boolean flag to log raw plan fragment model into the `elink.log` file. Raw model XML is generated by process activity before applying stylesheet. Value should be set to `false`, if model is configured for a large number of relationships (for instance, `ProductComprisedOf`) to avoid out of memory error.
`TypeOfPublish`	Refers to the publish type. It can be full publish or delta publish.
`Tenant`	Refers to the TenantId in the header of published model. By default the "enterprise name" appears as the TenantId in the published model. It is not recommended to directly assign the value for this parameter in the workflow as it affects all Model Publish activities for the enterprise.

ProcessActionSubsetData Parameter

Activity: ProcessActionSubsetData	
Parameter	**Description**
`InRecordList`	Record list received from the `EvaluateSubset` activity.
`recordCount`	Record processed by the `EvaluateSubset` activity.
`ActionModelMap`	Action model XSL file location.
`ActionModelBatchSize`	Complete Action model published in batch based on the specified batch size.
`ActionModelCatName`	Logical topic name mapped to the actual topic name in the `ConfigValues.xml` file.
`connShareMode`	Connection share mode value to connect to the TIBCO Enterprise Message Service server.
`transacted`	Boolean value.
`publishAction`	Refers to the parameter value related to header action such as Bulk (for bulk model).
`Channel`	Valid value is either `JMS`, `FILE`, or `BOTH`. `JMS` is used for the Online catalog publication and `FILE` is used for the Offline Catalog publication. For more information on the Offline Catalog, see Customization of Publish Catalog Workflow. When you set `BOTH`, the data is published and the files are created at the same time.
`Metadata Required`	This parameter should be commented if publish model is triggered through a process and not through UI.
`folderLocation`	Actual file location. This is applicable when `FILE` or `BOTH` value is used for the `Channel` parameter.
`logRawModel`	A boolean flag to log raw Action model into the `elink.log` file. Raw model XML is generated by process activity before applying stylesheet. Value must be set to `false`, if model is configured for a large number of relationships (for instance, `ProductComprisedOf`) to avoid out of memory error.
`TypeOfPublish`	Refers to the publish type. It can be full publish or delta publish.

Activity: ProcessActionSubsetData	
Parameter	**Description**
`Tenant`	Refers to the TenantId in the header of published model. By default the "enterprise name" appears as the TenantId in the published model. It is not recommended to directly assign the value for this parameter in the workflow as it affects all Model Publish activities for the enterprise.

ProcessPriceSubsetData Parameter

Activity: ProcessPriceSubsetData	
Parameter	**Description**
`InRecordList`	Record list received from the `EvaluateSubset` activity.
`recordCount`	Record processed by the `EvaluateSubset` activity.
`PriceModelMap`	Price model XSL file location.
`PriceModelBatchSize`	Complete Price model published in batch based on the specified batch size.
`PriceModelCatName`	Logical topic name mapped to the actual topic name in the `ConfigValues.xml` file.
`connShareMode`	Connection share mode value to connect to the TIBCO Enterprise Message Service server.
`transacted`	Boolean value.
`publishAction`	Refers to the parameter value related to header action such as Bulk (for bulk model).
`Channel`	Valid value is either `JMS`, `FILE`, or `BOTH`. `JMS` is used for the Online catalog publication and `FILE` is used for the Offline Catalog publication. For more information on the Offline Catalog, see Customization of Publish Catalog Workflow. When you set `BOTH`, the data is published and the files are created at the same time.
`CatalogUseForFilter`	The value of `CatalogUse` selected from the UI Bulk Model Publish screen.
`MetadataRequired`	This parameter should be commented if publish model is triggered through a process and not through UI.
`folderLocation`	Temporary folder location. This is applicable when `FILE` or `BOTH` value is used for the Channel parameter.

Activity: ProcessPriceSubsetData	
Parameter	**Description**
`logRawModel`	A boolean flag to log raw Action model into the `elink.log` file. Raw model XML is generated by process activity before applying stylesheet. Value must be set to false, if model is configured for a large number of relationships (for instance, `ProductComprisedOf`) to avoid out of memory error.
`TypeOfPublish`	Refers to the publish type. It can be full publish or delta publish.
`Tenant`	Refers to the TenantId in the header of published model. By default the "enterprise name" appears as the TenantId in the published model. It is not recommended to directly assign the value for this parameter in the workflow as it affects all Model Publish activities for the enterprise.

ProcessDiscountSubsetData Parameter

Activity: ProcessDiscountSubsetData	
Parameter	**Description**
`InRecordList`	Record list received from the `EvaluateSubset` activity.
`recordCount`	Record processed by the `EvaluateSubset` activity.
`DiscountModelMap`	Discount model XSL file location.
`DisModelBatchSize`	Complete Discount model published in batch based on the specified batch size.
`DiscountModelCatName`	Logical topic name mapped to the actual topic name in the `ConfigValues.xml` file.
`connShareMode`	Connection share mode value to connect to the TIBCO Enterprise Message Service server.
`transacted`	Boolean value.
`publishAction`	Refers to the parameter value related to header action such as Bulk (for bulk model).

Activity: ProcessDiscountSubsetData	
Parameter	**Description**
`Channel`	Valid value is either `JMS`, `FILE`, or `BOTH`. `JMS` is used for the Online catalog publication and `FILE` is used for the Offline Catalog publication. For more information on the Offline Catalog, see Customization of Publish Catalog Workflow. When you set `BOTH`, the data is published and the files are created at the same time.
`CatalogUseForFilter`	The value of `CatalogUse` selected from the UI Bulk Model Publish screen.
`MetadataRequired`	This parameter should be commented if publish model is triggered through a process and not through UI.
`folderLocation`	Temporary folder location. This is applicable when `FILE` or `BOTH` value is used for the Channel parameter.
`logRawModel`	A boolean flag to log raw Action model into the `elink.log` file. Raw model XML is generated by process activity before applying stylesheet. Value must be set to false, if model is configured for a large number of relationships (for instance, `ProductComprisedOf`) to avoid out of memory error.
`TypeOfPublish`	Refers to the publish type. It can be full publish or delta publish.
`Tenant`	Refers to the TenantId in the header of published model. By default the "enterprise name" appears as the TenantId in the published model. It is not recommended to directly assign the value for this parameter in the workflow as it affects all Model Publish activities for the enterprise.

Publish Catalog Showing CatalogUse Options and Behavior

The following table shows the behavior of product records with their child records when publish catalog is used with or without CatalogUse option.

ProductA is the parent record and ProductB, ProductC, and ProductD are the child records of ProductA with PCO relationship.

PRODUCT RECORDS hierarchy in Fulfillment Catalog repository	Selected CatalogUse in PublishCatalog screen	Published product xml files in Zip file or JMS
ProductA (All) ProductB (All) ProductC (Technical) ProductD (Commercial)	All (Default)	**ProductA (All)** ProductB (All) ProductC (Technical) ProductD (Commercial)
ProductA (All) ProductB (All) ProductC (Technical) ProductD (Commercial)	Commercial	**ProductA (All)** ProductB (All) ProductD (Commercial)
ProductA (All) ProductB (All) ProductC (Technical) ProductD (Commercial)	Technical	**ProductA (All)** ProductB (All) ProductC (Technical)
ProductA (Commercial) ProductB (All) ProductC (Technical) ProductD (Commercial)	Commercial	**ProductA (Commercial)** ProductB (All) ProductD (Commercial)
ProductA (Commercial) ProductB (All) ProductC (Technical) ProductD (Commercial)	All (Default)	**ProductA (Commercial)** ProductB (All) ProductC (Technical) ProductD (Commercial)
ProductA (Commercial) ProductB (All) ProductC (Technical) ProductD (Commercial)	Technical	ProductB (All) ProductC (Technical)
ProductA (Technical) ProductB (All) ProductC (Technical)	Technical	**ProductA (Technical)** ProductB (All) ProductC (Technical)
ProductA (Technical) ProductB (All) ProductC (Technical)	Commercial	ProductB (All)

PRODUCT RECORDS hierarchy in Fulfillment Catalog repository	Selected CatalogUse in PublishCatalog screen	Published product xml files in Zip file or JMS
ProductA (Technical) ProductB (All) ProductC (Technical)	All (Default)	**ProductA (Technical)** ProductB (All) ProductC (Technical)

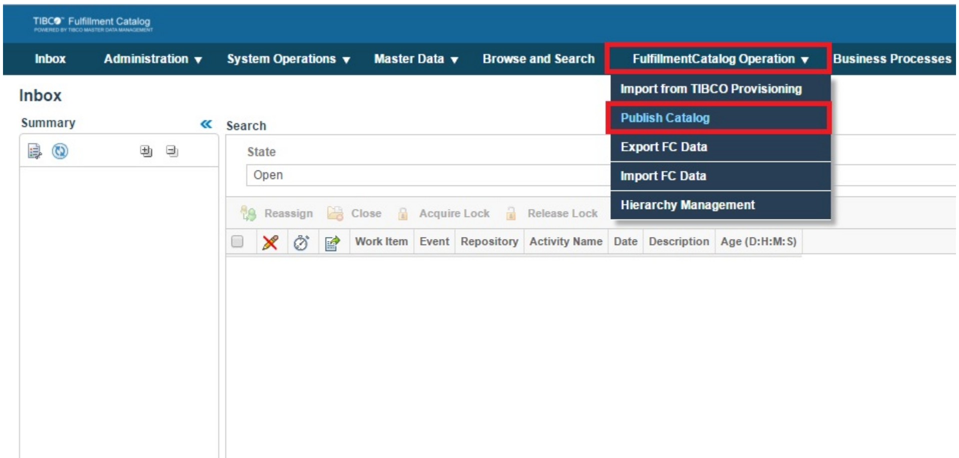
Accessing and Performing Full Data Publish

When performing Full Publish of Catalog, all the records for a selected model, in an enterprise, are published. In case of PRODUCT repository, if there are any changes to the offerID, the `OfferIdMappings.xml` file will also be published. To fully publish the catalog using the Fulfillment Catalog graphical user interface, perform the following steps:

Procedure

1. Log in to Fulfillment Catalog.
2. Click **FulfillmentCatalog Operation** > **Publish Catalog**.

   Accessing Publish Catalog from the Fulfillment Catalog Interface

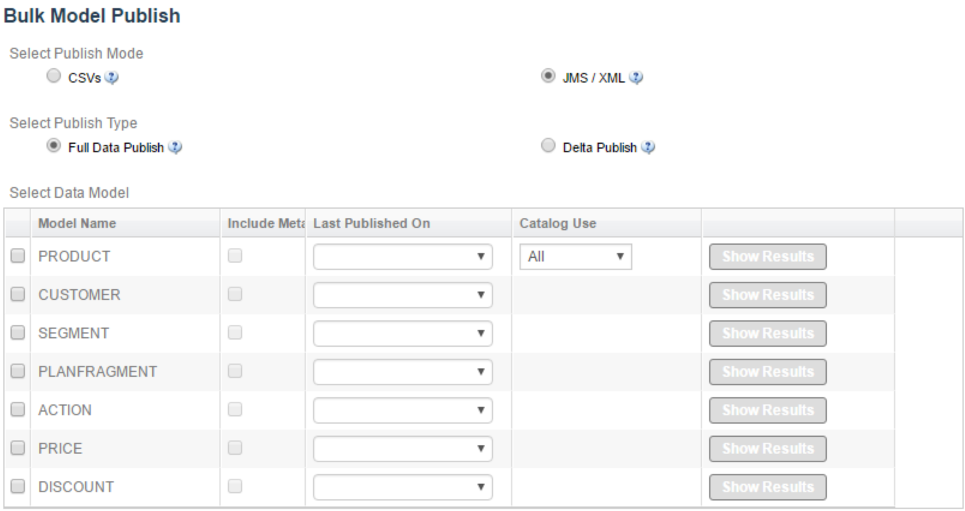


3. Select the **Publish Mode** from the CSVs and JMS / XML options.

   - **CSVs**: Exports repository record and generates CSV files. Set of exported data (or a zip file) can be downloaded from the event log.
   - **JMS / XML**: Triggers the publish bulk model workflow (`wfin26bulkmodelpublisherv1.xml`). Publish type JMS / XML signifies that it reads channel parameter set in the workflow and published messages on JMS or creates the data files, or both at the same time.

4. Select **Full Data Publish** for the **`Select Publish Type`** field.

Bulk Model Publishing



Bulk Model Publish

Select Publish Mode
○ CSVs ● JMS / XML

Select Publish Type
● Full Data Publish ○ Delta Publish

Select Data Model

	Model Name	Include Meta	Last Published On	Catalog Use	
☐	PRODUCT	☐	▾	All ▾	Show Results
☐	CUSTOMER	☐	▾		Show Results
☐	SEGMENT	☐	▾		Show Results
☐	PLANFRAGMENT	☐	▾		Show Results
☐	ACTION	☐	▾		Show Results
☐	PRICE	☐	▾		Show Results
☐	DISCOUNT	☐	▾		Show Results

5. Select the data model to publish.
6. **Include Metadata**: In the previous version of Fulfillment Catalog, when publishing the model, only the record values were being exported to JMS / XML. Although only the record value was sufficient for the OMS, the new engine, OCS (Order Capturing System), required the metadata of the record as well. The metadata is only needed when the data is written to offline files, and not when the data is sent to JMS. The provision of a checkbox lets the user to enable the system to create a separate file for metadata. Since, records belonging to a repository have similar metadata, there is only a single file written for it.

   > If **Include Metadata** checkbox is selected when the **Channel** parameter in workflow is set to JMS, the checkbox value is discarded.

7. **Catalog Use**: There is a new attribute introduced in PRODUCT repository called Records.

   The select option enables the user to select the records that need to be exported. This option is applicable for exporting data to JMS as well as exporting data to XML files. The provided values are All, Technical, and Commercial. The impact of the selection is described as follows:

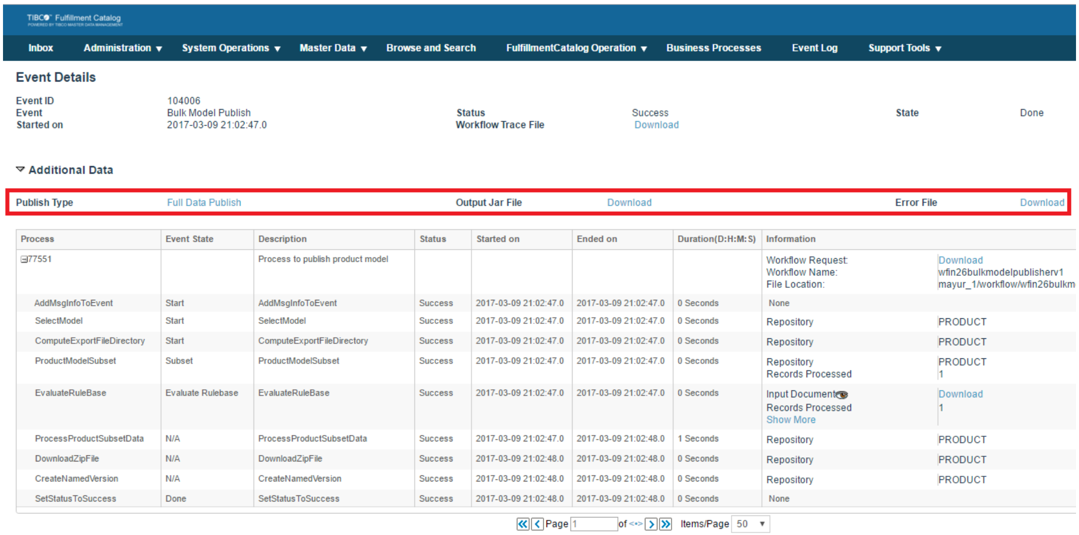
   - **All**: Fulfillment Catalog will export all the (Confirmed if VersionOption=”CONFIRMED” or Latest if VersionOption=”LATEST”) data irrespective of the value of `RecordUse` attribute for individual record.
   - **Technical**: Records having `RecordUse` attribute value as **All** or **Technical**, and also conforming to the `VersionOption` criteria, are exported.
   - **Commercial**: Records having `RecordUse` attribute value as **All** or **Commercial**, and also conforming to the `VersionOption` criteria, are exported.

8. Click **Publish** to start the workflow.
9. The Success page with the published model data status is displayed.

   > The `JMS / XML` option signifies that it reads the channel parameter set in the workflow and published messages on JMS or creates the data files, or both at the same time. Also, respective text data - `ProductID`, `ProductIDExt` (only if it exists) must be provided, as applicable.

   If the Channel parameter in workflow is set to FILE or BOTH, then the published catalog ZIP file will be available for download from the Event log, only if the publish event is successful.

Event Log and Download Option of the Published Catalog



Event Details

Event ID 104006
Event Bulk Model Publish
Started on 2017-03-09 21:02:47.0
Status Success
Workflow Trace File Download
State Done

Additional Data

Publish Type Full Data Publish
Output Jar File Download
Error File Download

Process	Event State	Description	Status	Started on	Ended on	Duration(D:H:M:S)	Information	
77551		Process to publish product model					Workflow Request: Workflow Name: File Location:	Download wfin26bulkmodelpublisherv1 mayur_1/workflow/wfin26bulkm
AddMsgInfoToEvent	Start	AddMsgInfoToEvent	Success	2017-03-09 21:02:47.0	2017-03-09 21:02:47.0	0 Seconds	None	
SelectModel	Start	SelectModel	Success	2017-03-09 21:02:47.0	2017-03-09 21:02:47.0	0 Seconds	Repository	PRODUCT
ComputeExportFileDirectory	Start	ComputeExportFileDirectory	Success	2017-03-09 21:02:47.0	2017-03-09 21:02:47.0	0 Seconds	Repository	PRODUCT
ProductModelSubset	Subset	ProductModelSubset	Success	2017-03-09 21:02:47.0	2017-03-09 21:02:47.0	0 Seconds	Repository Records Processed	PRODUCT 1
EvaluateRuleBase	Evaluate Rulebase	EvaluateRuleBase	Success	2017-03-09 21:02:47.0	2017-03-09 21:02:47.0	0 Seconds	Input Document Records Processed Show More	Download 1
ProcessProductSubsetData	N/A	ProcessProductSubsetData	Success	2017-03-09 21:02:47.0	2017-03-09 21:02:48.0	1 Seconds	Repository	PRODUCT
DownloadZipFile	N/A	DownloadZipFile	Success	2017-03-09 21:02:48.0	2017-03-09 21:02:48.0	0 Seconds	Repository	PRODUCT
CreateNamedVersion	N/A	CreateNamedVersion	Success	2017-03-09 21:02:48.0	2017-03-09 21:02:48.0	0 Seconds	Repository	PRODUCT
SetStatusToSuccess	Done	SetStatusToSuccess	Success	2017-03-09 21:02:48.0	2017-03-09 21:02:48.0	0 Seconds	None	

When the event is successful a ZIP file is generated which contains XMLs for models present in the respective repositories.

> For the PRODUCT model an additional XML is generated, which contains the offerID, used in the product models, along with a mapping key. This mapping key is then used in the actual XML models instead of the offerID value to reduce the size of the individual XMLs.

Delta Publish Overview

The Delta Publish of catalog can also be referred to as incremental publish. When you have an environment with a considerable amount of repository data, and if you make minor changes to the data, publishing all the data for a small increment takes a lot of time because the entire data set gets published.

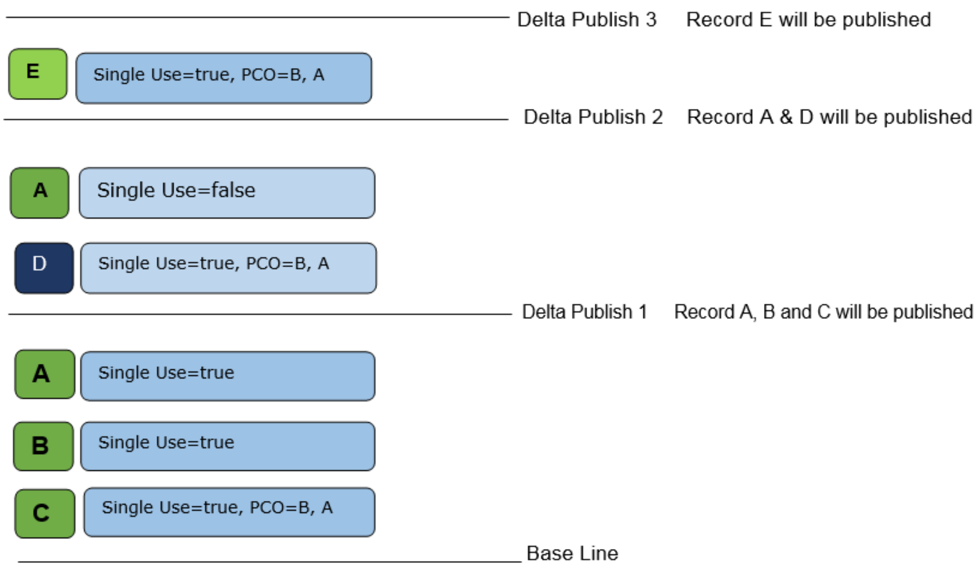
The Delta Publish feature addresses the issue by publishing only what has been changed. You can select the date and time and upon selection only those records will be published which are created or modified after selected date and time.

Delta Publish publishes records based on the last modified date of the record. The last modified date of the record is modified when the record attribute and the relationship attribute is modified.

> - In case of PRODUCT repository, if there are any changes to the offerID, the `OfferIdMappings.xml` file will also be published.
> - If you invoke Delta Publish for the first time on an enterprise, the entire data in the repository will be published.

The timestamp is recorded after each Delta Publish.

Pictorial Representation of Delta Publish


Delta Publish 3
Record E will be published
E
Single Use=true, PCO=B, A
Delta Publish 2
Record A & D will be published
A
Single Use=false
D
Single Use=true, PCO=B, A
Delta Publish 1
Record A, B and C will be published
A
Single Use=true
B
Single Use=true
C
Single Use=true, PCO=B, A
Base Line

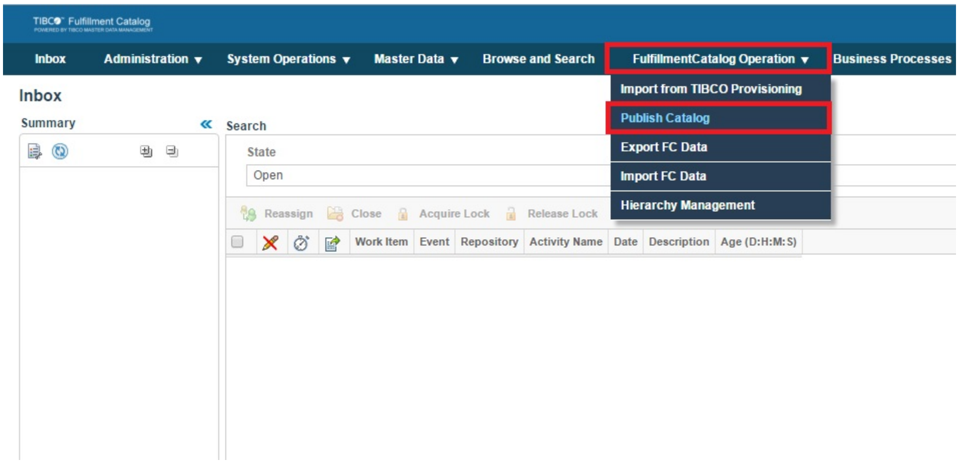
Accessing and Performing Delta Publish

Perform the following steps to access and perform delta publish:

Procedure

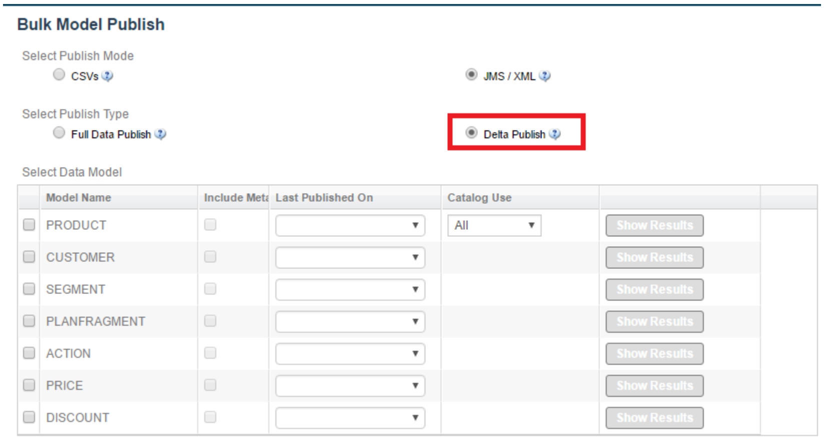
1. Click **Fulfillment Catalog Operation** > **Publish Catalog**.
   The Bulk Model Publish page is displayed.

   Accessing Publish Catalog from the Fulfillment Catalog Interface

2. Select the option **Delta Publish**.

Bulk Model Publish Options



Bulk Model Publish

Select Publish Mode

○ CSVs ● JMS / XML

Select Publish Type

○ Full Data Publish ● Delta Publish

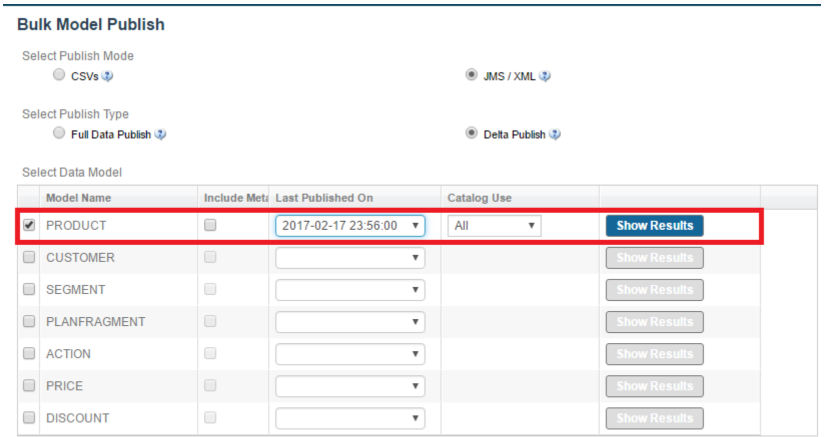
Select Data Model

	Model Name	Include Meta	Last Published On	Catalog Use	
☐	PRODUCT	☐		All	Show Results
☐	CUSTOMER	☐			Show Results
☐	SEGMENT	☐			Show Results
☐	PLANFRAGMENT	☐			Show Results
☐	ACTION	☐			Show Results
☐	PRICE	☐			Show Results
☐	DISCOUNT	☐			Show Results

3. Select the required model name and provide values for the following fields:

   - **`Include Metadata?`** (the default value is No)
   - **`Last Published On`** (value only applicable for Delta Publish)
   - **`Catalog Use`** (the default value is All)

Selecting the PRODUCT Repository for Delta Publish



Bulk Model Publish

Select Publish Mode

○ CSVs ● JMS / XML

Select Publish Type

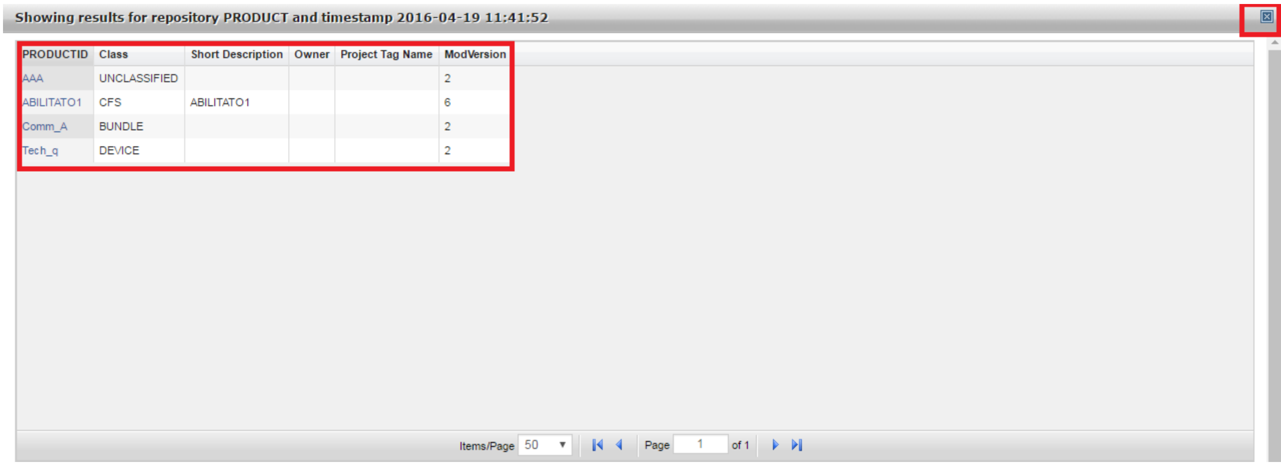
○ Full Data Publish ● Delta Publish

Select Data Model

	Model Name	Include Meta	Last Published On	Catalog Use	
☑	PRODUCT	☐	2017-02-17 23:56:00	All	Show Results
☐	CUSTOMER	☐			Show Results
☐	SEGMENT	☐			Show Results
☐	PLANFRAGMENT	☐			Show Results
☐	ACTION	☐			Show Results
☐	PRICE	☐			Show Results
☐	DISCOUNT	☐			Show Results

4. Click the **Show Results** button.
   The delta publish results are displayed in a new window.

Results for the PRODUCT Repository of Delta Publish

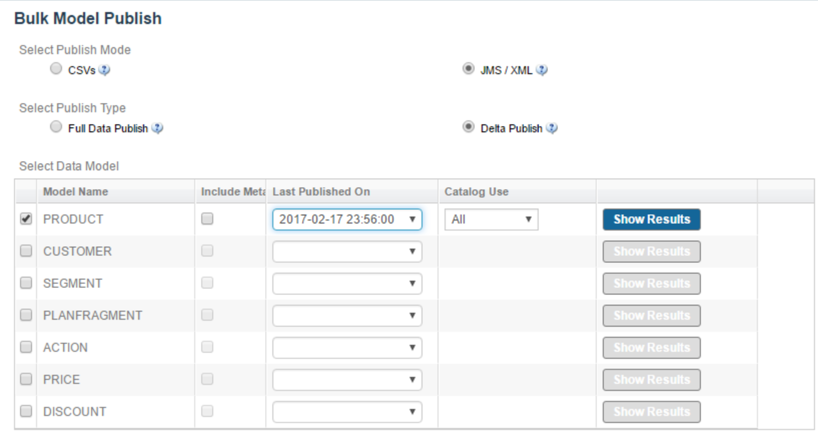


Showing results for repository PRODUCT and timestamp 2016-04-19 11:41:52

PRODUCTID	Class	Short Description	Owner	Project Tag Name	ModVersion
AAA	UNCLASSIFIED				2
ABILITATO1	CFS	ABILITATO1			6
Comm_A	BUNDLE				2
Tech_q	DEVICE				2

5. Click the **X** to close the window.
   The Bulk Model Publish page is displayed.

Delta Publishing the Results for the PRODUCT Repository



Bulk Model Publish

Select Publish Mode

○ CSVs ● JMS / XML

Select Publish Type

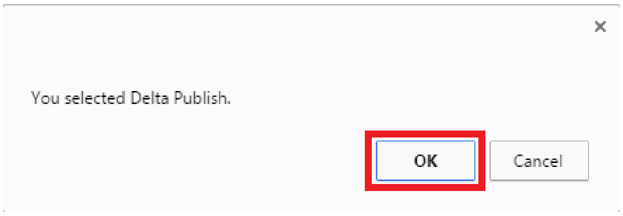
○ Full Data Publish ● Delta Publish

Select Data Model

	Model Name	Include Meta	Last Published On	Catalog Use	
☑	PRODUCT	☐	2017-02-17 23:56:00	All	Show Results
☐	CUSTOMER	☐			Show Results
☐	SEGMENT	☐			Show Results
☐	PLANFRAGMENT	☐			Show Results
☐	ACTION	☐			Show Results
☐	PRICE	☐			Show Results
☐	DISCOUNT	☐			Show Results

6. Click the **Publish** button.
   A dialog box pops up indicating that you have selected delta publish.

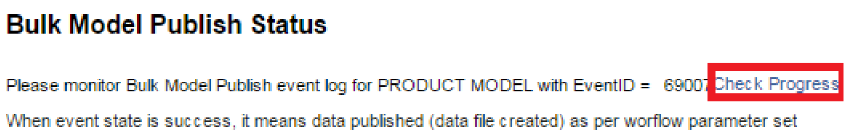
   Confirmation Box for Delta Publish



You selected Delta Publish.

7. Click **OK** to continue.
   The Bulk Model Publish Status page is displayed.

   Bulk Model Publish Status for Delta Publish



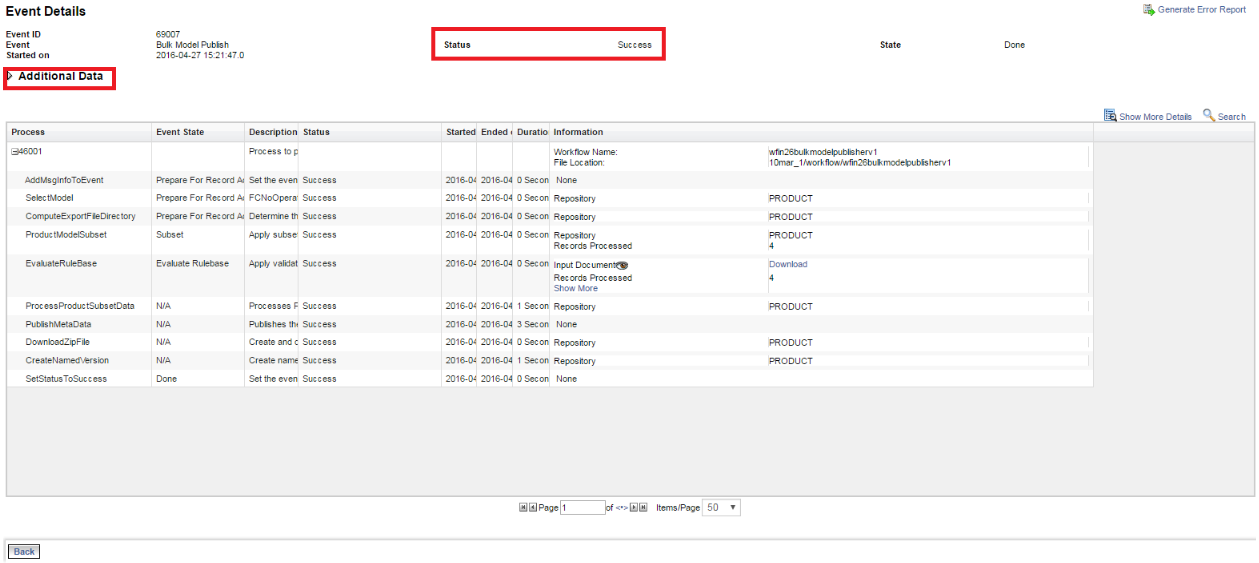
Bulk Model Publish Status

Please monitor Bulk Model Publish event log for PRODUCT MODEL with EventID = 69007 Check Progress

When event state is success, it means data published (data file created) as per worflow parameter set

8. Click **Check Progress** link to see the status of the delta publish.
   The Event Details page is displayed and the status of the delta publish is displayed.

   Event Details for Delta Publish



Event Details

Event ID 69007
Event Bulk Model Publish
Started on 2016-04-27 15:21:47.0

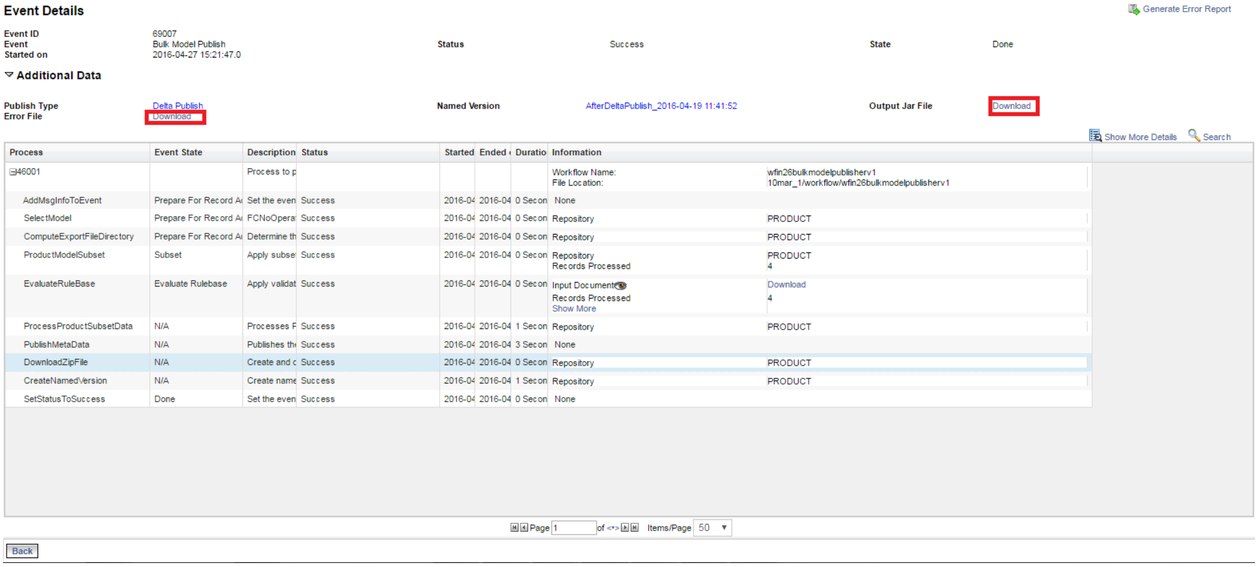
Status Success

State Done

Process	Event State	Description	Status	Started	Ended (	Duratio	Information	
46001		Process to p					Workflow Name: File Location:	wfin26bulkmodelpublisherv1 10mar_1/workflow/wfin26bulkmodelpublisherv1
AddMsgInfoToEvent	Prepare For Record A	Set the even	Success	2016-04	2016-04	0 Secon	None	
SelectModel	Prepare For Record A	FCNoOpera	Success	2016-04	2016-04	0 Secon	Repository	PRODUCT
ComputeExportFileDirectory	Prepare For Record A	Determine th	Success	2016-04	2016-04	0 Secon	Repository	PRODUCT
ProductModelSubset	Subset	Apply subse	Success	2016-04	2016-04	0 Secon	Repository Records Processed	PRODUCT 4
EvaluateRuleBase	Evaluate Rulebase	Apply validat	Success	2016-04	2016-04	0 Secon	Input Document Records Processed Show More	Download 4
ProcessProductSubsetData	N/A	Processes F	Success	2016-04	2016-04	1 Secon	Repository	PRODUCT
PublishMetaData	N/A	Publishes th	Success	2016-04	2016-04	3 Secon	None	
DownloadZipFile	N/A	Create and c	Success	2016-04	2016-04	0 Secon	Repository	PRODUCT
CreateNamedVersion	N/A	Create name	Success	2016-04	2016-04	1 Secon	Repository	PRODUCT
SetStatusToSuccess	Done	Set the even	Success	2016-04	2016-04	0 Secon	None	

9. Click the arrow preceding **Additional Data** to expand the details.
   Click the **Download** link following the *Output Jar File* field to download the published file. Click the **Download** link following the *Error File* field to download the error file to check for any errors, if any.

   Downloading Delta Publish Output



Event Details

Generate Error Report

Event ID 69007
Event Bulk Model Publish
Started on 2016-04-27 15:21:47.0
Status Success
State Done

▽ **Additional Data**

Publish Type Delta Publish
Error File Download
Named Version AfterDeltaPublish_2016-04-19 11:41:52
Output Jar File Download

Show More Details Search

Process	Event State	Description	Status	Started	Ended (	Duratio	Information	
⊟46001		Process to p					Workflow Name: File Location:	wfin26bulkmodelpublisherv1 10mar_1/workflow/wfin26bulkmodelpublisherv1
AddMsgInfoToEvent	Prepare For Record A	Set the even	Success	2016-04	2016-04	0 Secon	None	
SelectModel	Prepare For Record A	FCNoOpera	Success	2016-04	2016-04	0 Secon	Repository	PRODUCT
ComputeExportFileDirectory	Prepare For Record A	Determine th	Success	2016-04	2016-04	0 Secon	Repository	PRODUCT
ProductModelSubset	Subset	Apply subse	Success	2016-04	2016-04	0 Secon	Repository Records Processed	PRODUCT 4
EvaluateRuleBase	Evaluate Rulebase	Apply validat	Success	2016-04	2016-04	0 Secon	Input Document Records Processed Show More	Download 4
ProcessProductSubsetData	N/A	Processes F	Success	2016-04	2016-04	1 Secon	Repository	PRODUCT
PublishMetaData	N/A	Publishes th	Success	2016-04	2016-04	3 Secon	None	
DownloadZipFile	N/A	Create and c	Success	2016-04	2016-04	0 Secon	Repository	PRODUCT
CreateNamedVersion	N/A	Create name	Success	2016-04	2016-04	1 Secon	Repository	PRODUCT
SetStatusToSuccess	Done	Set the even	Success	2016-04	2016-04	0 Secon	None	

Relationship Record Filter for Delta Publish

If there is a product record named "Modem" having two characteristics "Speed" and "Color" and if you want to publish only the "Speed" characteristic and skip the "Color" characteristic during the publish, you have to specify the name of the relationship "characteristic" and the value of the Name attribute of the "Color" characteristic in the `$AC_HOME/samples/DeltaPublishRelationshipRecordFilter.csv` file. You will then have to upload the file in `REL_RECORD_FILTER_DS` data source.

You can also mention the above filter details in the `$AC_HOME/samples/DeltaPublishRelationshipRecordFilter.properties` file.

When executing delta publish, if you want to skip multiple relationship records, of the same relationship, you will have to specify the comma separated values of the name attribute of the relationship records.

Customization of Publish Catalog Workflow

The Workflow `wfin26bulkmodelpublisherv1.xml` can be modified to change the behavior in following ways:

- Publishing Catalog to XML Files
- Publishing Catalog to a JMS Channel
- Publishing Catalog to JMS and XML Channel
- Publishing Catalog at a Fixed Location

Publishing Catalog to XML Files

A parameter needs to be changed for the `Process<<Product/Customer/Segment/PlanFragment/Action/Price/ Discount>>SubsetData` activity in the `wfin26bulkmodelpublisherv1.xml` file. The workflow activity name will be dependent on the model name that will be published. For example, if the model being published is PRODUCT then the activity name will be `ProcessProductSubsetData`.

Any existing definition of above parameters may be commented if their definition does not match as defined above.

The published catalog ZIP file will be available for download, from the Event log, only if the publish event is successful.

```
<Parameter direction="in" name="Channel" type="string" eval="constant">FILE</
Parameter>
```

Publishing Catalog to a JMS Channel

A parameter needs to be changed for the `Process<<Product/Customer/Segment/PlanFragment/Action/Price/ Discount>>SubsetData` activity in the `wfin26bulkmodelpublisherv1.xml` file. The workflow activity name will be dependent on the model name that will be published. For example, if the model being published is PRODUCT then the activity name will be `ProcessProductSubsetData`.

```
<Parameter direction="in" name="Channel" type="string" eval="constant">JMS</
Parameter>
```

Publishing Catalog to JMS and XML Channel

A parameter needs to be changed for the `Process<<Product/Customer/Segment/PlanFragment/Action/Price/ Discount>>SubsetData` activity in the `wfin26bulkmodelpublisherv1.xml` file. The workflow activity name will be dependent on the model name that will be published. For example, if the model being published is PRODUCT then the activity name will be `ProcessProductSubsetData`.

```
<Parameter direction="in" name="Channel" type="string" eval="constant">BOTH</
Parameter>
```

The published catalog ZIP file will be available for download, from the Event log, only if the publish event is successful.

Publishing Catalog at a Fixed Location

1. Two parameters need to be changed for the `Process<<Product/Customer/Segment/PlanFragment/Action/Price/ Discount>>SubsetData` activity in the `wfin26bulkmodelpublisherv1.xml` file. The workflow activity name will be dependent on the model name that will be published. For example, if the model being published is PRODUCT then the activity name will be `ProcessProductSubsetData`.

   a. Confirm that the value of `Channel` parameter is `FILE` as shown in the following example:

   ```
   <Parameter direction="in" name="Channel" type="string" eval="constant">FILE</
   Parameter>
   ```

   b. Confirm that the value of `FolderLocation` parameter is *<absolute path location>* and its `eval` is `constant` as shown in the following example:

   ```
   <Parameter direction="in" name="FolderLocation" type="string"
   eval="constant"><absolute path></Parameter>
   ```

2. One parameter to change under `CreateAndDownloadZip` activity in `wfin26bulkmodelpublisherv1.xml` file:

   a. Confirm that the value for `DeleteSourceDir` parameter is `FALSE` as shown in the following example:

   ```
   <Parameter direction="in" name="DeleteSourceDir" type="boolean"
   eval="variable">FALSE</Parameter>
   ```

Verifying Input Parameters

Verify the validity of the input parameters as described in the following table:

Procedure

1. Set input parameter `Channel` = `FILE` in the *ACModelPublish* activity.
2. Specify the `folderLocation` with value as valid file path/directory which has a write access.
   File is created at the specified location and no message is published on the JMS topic.
3. Set input parameter `Channel` = `JMS` in the *ACModelPublish* activity.
   Message is published on the JMS topic.
4. Set input parameter `Channel` something other than the `JMS` or `FILE`.
   Activity throws an error as an invalid input for the input parameter `Channel`.
5. Set input parameter `Channel` = `BOTH` in the `ACModelPublish` activity.
   File is created at the specified location and message is published on the JMS topic.

Product Model Extension

You can execute the product model extension in the following two ways:

1. If new attribute is added in the product repository, and needs to be published in the product model. For details, see Product Model Extension for Additional Attribute.
2. If product model with more than one level of hierarchy needs to be published. For details, see Publish Data of Product Model with Hierarchy More Than One Level.

Attribute name should be preceded by "C". For example, to create an attribute "TEST", column name should be "CTEST", and the same should be specified in the `productmodelattribute.properties` file.

Product Model Extension For Additional Attribute

If a new attribute is added in the product repository, the following components are responsible to modify product model:

- **`productmodelattibute.properties`**: It has the list of all attribute names (column) required to generate product model. It is located at `<JBOSS_HOME>/modules/com/tibco/fulfillmentcatalog/main/acprop.jar`.
- **`mpfromcatalogitemtobulkproductmodelv1.xsl`**: This is a stylesheet file, which generates product model as per required structure. It is located at `MQ_COMMON_DIR/<ENT_NAME>/maps` directory.

Performing Product Model Extension

The steps to perform product model extension are as follows:

Procedure

1. Go to the `<JBOSS_HOME>/modules/com/tibco/fulfillmentcatalog/main/` directory.
2. Extract `acprop.jar`.
3. Open `productmodelattibute.properties` file.
4. Add column name for attribute, which needs to be added. For instance, `CPRODUCTCOUNT`.
5. Create jar once again.
6. Modify map (.XSL) file to publish `CPRODUCTCOUNT` value in product model.
7. Start JBOSS server.

8. Publish product model.

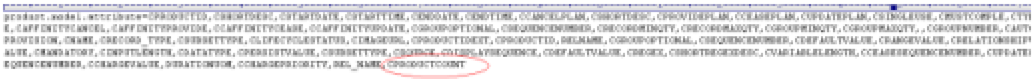
> When adding any attribute in the PRODUCT repository, do not provide column name because TIBCO MDM generates it.

Example for Product Model Extension

If you want to add a new attribute in the product model, for example, `PRODUCTCOUNT` in the product repository, perform the following steps:

1. Open the `productmodelattibute.properties` file located at `<JBOSS_HOME>/modules/com/tibco/fulfillmentcatalog/main/acprop.jar`.
2. Add column name in the end, for example, `CPRODUCTCOUNT` in the existing attribute list.

   Sample Product Model Attribute Properties 1

3. Modify the map file `mpfromcatalogitemtobulkproductmodelv1.xsl` to add mapping for the newly added attribute. The following snippet shows the sample map file.

```
<ns0:characteristics>
    <ns0:name>PRODUCTCOUNT</ns0:name>
    <ns0:value>
        <ns0:type>
        <xsl:value-of select="&quot;Feature&quot;"/>
        </ns0:type>
        <xsl:choose>
        <xsl:when test="count(./CPRODUCTCOUNT)>0 and string-length(./
CPRODUCTCOUNT) > 0">
            <ns0:discreteValue>
            <xsl:value-of select="./CPRODUCTCOUNT"/>
            </ns0:discreteValue>
        </xsl:when>
        <xsl:otherwise>
            <ns0:discreteValue>
            <xsl:value-of select="&quot;NULL&quot;"/>
            </ns0:discreteValue>
        </xsl:otherwise>
        </xsl:choose>
    </ns0:value>
    <ns0:simpleRule>
        <ns0:name>
        <xsl:value-of select="&quot;RULE&quot;"/>
        </ns0:name>
        <ns0:ruleSetOutcome>
        <xsl:value-of select="./CPRODUCTCOUNT"/>
        </ns0:ruleSetOutcome>
    </ns0:simpleRule>
</ns0:characteristics>
```

Publish Data of Product Model with Hierarchy More Than One Level

This feature provides the flexibility to publish the product model with more than one level hierarchy on demand.

You need to customize the product model map, and workflow file to publish product model with more than one level.

The following example explains how to publish product model with more than one level hierarchies.

Scenario: Record A at PRODUCT having relationship ProductPricedBy with record B at PRICE, which in turn has a relationship `PriceAlteredByDiscount` with record C at DISCOUNT and then record C has `DiscountRequiresProduct` with record D at PRODUCT.

In product model of record A, data related to record B will be published in the characteristic node. Similarly, data related to record C and D will be published same as B. Record B will have reference of record C, and record C will have reference of record D in its own simple rule.

Customizing the Product Model Map to Publish the Data with Hierarchy More Than One Level

The steps to customize the product model map to publish the data with hierarchy more than one level are as follows

Procedure

1. Pass all the relationship names explained in the scenario to activities mentioned as per the following table:

Workflow Name	Activity Name
wfin26bulkmodelpublisherv1.xml	ProductModelSubset

2. Modify product model map file to publish more than one level data in product model.

Sample workflow and map file is available in the sample folder `AC_HOME/samples/ProductModelCustomization` for above example. More than one level is supported other than ProductComprisedOf relationship.

TIBCO Fulfillment Catalog Relationships allow the definition of relationship types within and between Fulfillment Catalog repositories, conforming to SID entity relationship modeling.

Adding New Repository and Creating Relationship

You can extend the data model, for example, PRODUCT by adding a new repository, and creating a relationship between PRODUCT and the repository.

You can extend the data model, for example, PRODUCT by adding a new repository, and creating a relationship between PRODUCT and the repository.

To extend the Fulfillment Catalog data model, modify the following files:

File Name	Location	Description
`Productmodelattribute.properties`	`<JBOSS_HOME> / modules / com / tibco / fulfillmentcatalog / main / acprop.jar`	This file contains the list of all attribute names (column) required to generate product model.
`mpfromcatalogitemtobulkproductmodelv1.xsl`	`MQ_COMMON_DIR/<ENTNAME>/maps`	This is a stylesheet file, which generates product model according to the required structure.
`wfin26bulkmodelpublisherv1.xml`	`MQ_COMMON_DIR/<ENTNAME>/workflow`	Edit this file to add relationship name.

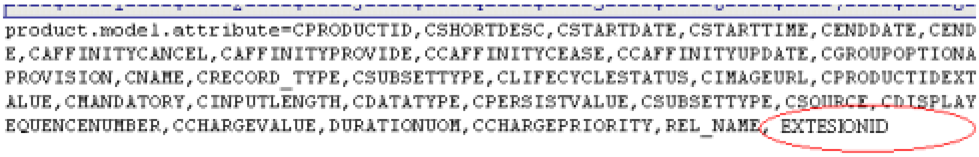
To extend the Fulfillment Catalog data model and add new repository, perform the following steps:

Procedure

1. Create the new EXTENSION repository.
2. Create an attribute for it, for instance, EXTENSIONID and EXTENSIONDESC. To add a new attribute in the product model, in the product repository, perform the following steps:
   a) Go to the `<JBOSS_HOME>/modules/com/tibco/fulfillmentcatalog/main/` directory.
   b) Extract `acprop.jar`.
   c) Using any text editor, open the `productmodelattibute.properties` file located at `<JBOSS_HOME>/modules/com/tibco/fulfillmentcatalog/main/acprop.jar`.
   d) Create relationship between PRODUCT and EXTENSION, for instance, *ProductHasExtension*.
   e) At the end of the `productmodelattibute.properties` file, add column name, `EXTESIONID` in the existing attribute list.

   Shut down the sever before you save the file.

   Sample Product Model Attribute Properties 2

```
product.model.attribute=CPRODUCTID,CSHORTDESC,CSTARTDATE,CSTARTTIME,CENDDATE,CEND
E,CAFFINITYCANCEL,CAFFINITYPROVIDE,CCAFFINITYCEASE,CCAFFINITYUPDATE,CGROUPOPTIONA
PROVISION,CNAME,CRECORD_TYPE,CSUBSETTYPE,CLIFECYCLESTATUS,CIMAGEURL,CPRODUCTIDEXT
ALUE,CMANDATORY,CINPUTLENGTH,CDATATYPE,CPERSISTVALUE,CSUBSETTYPE,CSOURCE,CDISPLAY
EQUENCENUMBER,CCHARGEVALUE,DURATIONUOM,CCHARGEPRIORITY,REL_NAME, EXTESIONID
```

3. Create the jar file, `acprop.jar` once again.
4. Add relationship name, *ProductHasExtension* in the `wfin26bulkmodelpublisherv1.xml` file for the following activities:
   a) ProductModelSubset
   b) ProcessProductSubsetData
5. Modify the map file `mpfromcatalogitemtobulkproductmodelv1.xsl` to include the new attribute in the product model. The following snippet shows the sample map file.

```
<xsl:for-each select="./ChildItems/ChildItem[REL_NAME ='ProductHasExtension']">
              <ns0:characteristics>
                <ns0:name>
                  <xsl:value-of select="./CEXTESIONID"/>
                </ns0:name>
                <ns0:description>
                  <xsl:value-of select="./CEXTESIONDESC"/>
                </ns0:description>
               </ns0:characteristics>
            </xsl:for-each>
```

6. Create a record in PRODUCT and EXTENSION repositories with relationship between them.
7. Start JBOSS server.
8. Publish Product model.

Modification of Rule Base

You can modify an existing rule-base according to your requirements using a text editor.

- Removing Mandatory Plan Fragment Attributes
- Removing Non-mandatory Plan Fragment Attributes
- Disabling Mandatory Plan Fragment Rules

Removing Mandatory Plan Fragment Attributes

To remove the mandatory Plan Fragment attributes, perform the following steps:

While creating the Plan Fragment record, the following are the mandatory values:

- Plan Fragment ID
- Plan Fragment Name
- Plan Fragment Version
- Description

Procedure

1. Go to `$MQ_COMMON_DIR/<enterprise>/catalog/master/<Catalog_ID_PLANFRAGMENT>`. Where:
   a) `enterprise` is the actual enterprise name for which the rule can be disabled.
   b) `Catalog_ID_PLANFRAGMENT` is the actual PLANFRAGMENT repository ID. It can be seen from the UI in the repository list.
2. Open the `catalogvalidation.xml` file with a text editor.
3. Change the given constraint and remove the mandatory attribute according to your requirement.

```
<constraint>
<name>Mandatory</name>
<description>Mandatory attributes</description>
<usefor>
   <var>PLANFRAGMENTNAME</var>
   <var>PLANFRAGMENTVERSION</var>
   <var>SHORTDESC</var>
</usefor>
<action>
<check>
       <explanation> Plan Fragment Name, Plan Fragment Version and Plan Fragment
Type, Plan Fragment Description are mandatory attributes</explanation>
     <defined>
           <var/>
     </defined>
</check>
</action>
</constraint>
```

Removing Non-mandatory Plan Fragment Attributes

You can modify the rule-base to remove the non-mandatory Plan Fragment attributes from the drop-down displayed on PRODUCT edit on TIBCO MDM native UI, or the Graphical Edit screen.

You can modify the rule-base to remove the non-mandatory Plan Fragment attributes from the drop-down displayed on PRODUCT edit on TIBCO MDM native UI, or the Graphical Edit screen.

By default, Plan Fragments drop-down shows a combination of the following attributes:

- Plan Fragment Name
- Plan Fragment Version
- Plan Fragment Type

To select the attributes to display on drop-down, perform the following steps:

Procedure

1. Go to `$MQ_COMMON_DIR/<enterprise>/catalog/master/<Catalog_ID_PRODUCT>`. Where:

   a) `enterprise` is the actual enterprise name for which the rule can be disabled.
   b) `Catalog_ID_PRODUCT` is the actual PRODUCT repository ID. It can be seen from the UI in the repository list.

2. Open the `catalogvalidation.xml` file with a text editor.

3. Change the given constraint to remove the values, which are not required.

```
<constraint>
<name>PlanFragments</name>
<description>Displays available PlanFragments for selection</description>
<usefor>
<var>PROVIDEPLAN</var>
     <var>CEASEPLAN</var>
     <var>UPDATEPLAN</var>
     <var>CANCELPLAN</var>
     <var>AFFINITYPROVIDE</var>
     <var>AFFINITYCEASE</var>
     <var>AFFINITYUPDATE</var>
     <var>AFFINITYCANCEL</var>
</usefor>
<action>
<select novalue="default" showoninput="2,3,4">
             <!-- 2nd, 3rd and 4th below will be displayed on UI dropdown -->
<table source="sql">
             <literal>PLANFRAGMENT/PRODUCTID</literal>
                 <literal>PLANFRAGMENT/PLANFRAGMENTNAME</literal><!-- This is 2
-->
                 <literal>PLANFRAGMENT/PLANFRAGMENTVERSION</literal><!-- This is
3 -->
                 <literal>PLANFRAGMENT/PLANFRAGMENTTYPE</literal><!-- This is 4
-->
                 <where type="SQL">
                     <sql>
                     <neq>
        <literal>PLANFRAGMENT/PLANFRAGMENTNAME</literal>
         <const type="string">NO_RECIPROCAL_ACTION</const>
                     </neq>
                     </sql>
</where>
</table>
     </select>
</action>
</constraint>
```

Disabling Mandatory Plan Fragment Rules

To disable mandatory Plan Fragment rules, perform the following steps:

Procedure

1. Go to `$MQ_COMMON_DIR/<enterprise>/catalog/master/<Catalog_ID_PLANFRAGMENT>`. Where:
   a) `enterprise` is the actual enterprise name for which the rule can be disabled.
   b) `Catalog_ID_PLANFRAGMENT` is the actual PLANFRAGMENT repository ID. It can be seen from the UI in the repository list.

2. Take the back up of the `catalogvalidation.xml` file.

3. Open the `catalogvalidation.xml` file with a text editor.

4. Search for the `NoPlan` constraint.

5. Comment the constraint to disable mandatory Plan Fragment for the Product record.

Data Modeling

Data Modeling describes the ways that TIBCO Fulfillment Catalog allows you to define, model and design products through action-based modeling, new affinity types, relationships, and group record.

Data Modeling also covers the following topics:

- Action-based Modeling
- Conditional Affinity
- ProductDependsOn and ProductRequiredFor Relationships
- Product Specification Field Decomposition
- Group Record Modeling

Action-based Modeling

TIBCO Fulfillment Catalog 3.0.0 allows you to define the set of actions, maintained in the ACTION repository. You can now define any number of unique fulfillment actions. Until this release, only four set of actions were supported.

The data modeled in the ACTION repository is published as the ACTION model.

The PRODUCT model has the `ProductHas<CustomAction>PlanFragment` relationship. The `ActionID` relationship attribute has a record from the ACTION repository.

These are the possible actions you can select:

- Provide
- Cease
- Update
- Cancel
- Custom Actions

Modeling Action-based Data

The following is an example of the action-based modeling. Perform the given steps to model action-based data:

Prerequisites

See Action-based Modeling for details related to Action-based Modeling.

Procedure

1. Create the `HomeMove` record in the ACTION repository.
2. Create the `P1` record in the PRODUCT repository.
3. Create the `ProductHasCustomPlanFragment` relationship for the `P1` record.
   In this relationship, the `ActionID` relationship attribute value `HomeMove` is selected from the ACTION repository.

   The existing PLANFRAGMENT record is assumed from the PLANFRAGMENT repository for the `ProductHasCustomPlanFragment` relationship. You can create a new PLANFRAGMENT record and then associate it.

Conditional Affinity

Fulfillment Catalog 2.0.0 introduced Conditional Affinity modeling, which combines InLink and CrossLink affinities in a single affinity type and provides additional flexibility. Affinity grouping enables different plan items to be grouped together based on the evaluation of the XPATH expression defined at the product catalog.

The following attributes are available at the record level in the AFFINITY GROUP of product or at the `ProductHasProvidePlanFragment`, `ProductHasUpdatePlanFragment`, `ProductHasCeasePlanFragment`, `ProductHasCancelPlanFragment`, and `ProductHasCustomPlanFragment` relationships. The additional configuration fields and rules in conditional affinity are :

AffinityType	Determines the type of affinity implemented. • InLink • CrossLink • Sequenced Affinity • ConditionalAffinity
AffinityCondition	Valid for Conditional type only. A String field containing an XPATH expression that evaluates to true or false based on data is in the order: • If the expression is true, the product plan item is affinity-grouped • If the expression is false, then the product plan item is not affinity-grouped • If the field is blank, assume that the value is true • If the XPATH expression evaluates to anything other than the true or false, AOPD fails, and returns an exception The XPATH expression evaluates against the following data fields on the order: • Order Header UDF Name and Value • Order Line ProductID • Order Line Action and ActionMode • Order Line UDF Name and Value The XPATH expression can also be defined against the following plan data fields: • planItem productID • planItem UDF name value • planItem Action

AffinityCorrelation	Valid for Conditional type only. The XPATH is evaluated on the Plan data and the order data. A String field containing an xpath expression based on a data is in the following order: • All plan items that evaluate to the same AffinityCorrelation are grouped together • The field is functionally similar to the LinkID method of correlating plan items in the InLink affinity. However, it allows correlation based on complex conditions without a restriction on the UDF names • If the field is blank, a default LinkID value is shared by all other blank configurations • If the XPATH expression evaluates to an empty string, the XPATH expression is blank, or assume a default LinkID The XPATH expression evaluates against the following order data fields: • Order Header UDF Name and Value • Order Line ProductID • Order Line Action and ActionMode • Order Line UDF Name and Value The XPATH expression can also be defined against the following plan data fields: • planItem productID • planItem UDF name value • planItem Action
AffinityParentGroup	Valid for Conditional type only. A boolean field containing the value true or false: • If set to true, the plan items with products sharing the same immediate parent product are grouped together • If set to false, the parent product is not considered for grouping
AffinityActionGroup	Valid for Conditional type only. A boolean field containing the value true or false: • If set to true, then only plan items with products that share the same action are grouped together • If set to false, then action is not considered for grouping

AffinityActionValue	AffinityActionValue is considered for grouping when AffinityActionGroup is set to true. This is valid for Conditional type only. String field containing an XPATH expression that evaluates to a String based on data is in the following order: The XPATH expression must evaluate to one of the following: • PROVIDE • UPDATE • CEASE • Empty String If the XPATH expression evaluates to anything other than these actions, then AOPD fails and returns an exception. • If the field is blank, or the return value from the XPATH expression is an empty string, the remaining action rules must be applied. The XPATH expression is able to evaluate against the following data fields on the order: • Order Header UDF Name and Value • Order Line Action and ActionMode • Order Line UDF Name and Value The XPATH expression can also be defined against the following plan data fields: • planItem productID • planItem UDF name value • planItem Action
AffinityProvide	Provide plan fragment name for affinity grouped plan item. Only plan items with the Provide action and the same value in this field are grouped together
AffinityUpdate	Update plan fragment name for affinity grouped plan item. Only plan items with the Update action and the same value in this field are grouped together
AffinityCease	Cease plan fragment name for affinity grouped plan item. Only plan items with the Cease action and the same value in this field are grouped together

AffinityCancel	Cancel plan fragment name for affinity grouped plan item. Only plan items with the Cancel action and the same value in this field are grouped together

See *TIBCO Fulfillment Catalog Product Catalog Guide* for more details.

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

XPath evaluates data from the planItem. See Sample Plan Item XML for more details.

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

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

See Sample XPATHs for XPATH definitions.

ProductDependsOn and ProductRequiredFor Relationships

The ProductDependsOn (PDO) and ProductRequiredFor (PRF) relationships helps you to create product offers without defining sequencing for the products. You can create ProductDependsOn relationship to lower level products instead of using ProductComprisedOf links.

ProductDependsOn Relationship	ProductRequiredFor Relationship
The ProductDependsOn (PDO) is a product dependency relationship to sequence the associated target and source plan items. The PDO relationship allows flexible product decomposition. This establishes a relationship between two products and is evaluated during the decomposition process.	The ProductRequiredFor (PRF) relationship is a prerequisite relationship for a product to add a target plan item.
The PDO and PRF relationships have the following two relationship attributes: • Source Action • Target Action	
The PRF relationship also has the third relationship attribute named ocvValidationReq. This is a boolean flag for validation. Based on a validation flag, the Fulfillment engine can decide if a product should be added, or only considered for validation purpose.	The PDO relationship also has the third relationship attribute named 'sequenceDirection'. The valid values of this attribute are either 'AFTER' or 'BEFORE'. This attribute will be paired with the provided values of SourceAction and TargetAction. For each SourceAction and TargetAction, there will be a value defined for the sequenceDirection attribute.

<table>
<tr><th>ProductDependsOn Relationship</th><th>ProductRequiredFor Relationship</th></tr>
<tr><td colspan="2">• A 'BEFORE' sequence direction will create a dependency of the target product on the source product.
• An 'AFTER' sequencing direction will create a dependency of the source product on the target product. This is the default.</td></tr>
<tr><td colspan="2">If no value is provided in the sequenceDirection attribute, the attribute defaults to 'AFTER', and the functionality works as it did before the introduction of sequenceDirection relationship attribute. This allows backward compatibility.</td></tr>
<tr><td colspan="2">The value defined in the sequenceDirection attribute will create a dependency of the target product on the source product or it will create a dependency of the source product on the target product.</td></tr>
</table>

Source and Target Attribute Values

The following table describes the different possible combinations:

SourceAction	TargetAction
Provide	Provide
Provide	Update
Provide	Cease
Provide	Cancel
Update	Provide
Update	Update
Update	Cease
Update	Cancel
Cease	Provide
Cease	Update
Cease	Cease
Cease	Cancel
Cancel	Provide
Cancel	Update
Cancel	Cease
Cancel	Cancel

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

SourceAction: Provide, Provide, Update, Cease, Cancel, Cease

TargetAction: Update, Cancel, Provide, Update, Provide, Update

You can also define sequenceDirection to match the following combination using comma separated values. For example

SourceAction: Provide, Provide, Update, Cease, Cancel, Cease

TargetAction: Update, Cancel, Provide, Update, Provide, Update

SequnceDirection: After, Before, After, Before, Before, After

Dependency between planitems occurs when both the following occur:

- The sequenceDirection attribute has valid values, i.e. either 'AFTER' or 'BEFORE.'
- The number of sequenceDirection attributes match with the number of Source Actions and the number of Target Actions.

❗ There is only one target action for any given source action.

The following table explains the PDO and PRF relationships and their impact on orders and plans.

Product Configuration	Order	Plan
Product A has a PRF relationship with Product B having source action and target action PROVIDE and PROVIDE	OL1=ProductA	Two plan item (A and B) do not depend on each other

Product A has PRF and PDO relationship with B and PRF and PDO has source action and target action PROVIDE and PROVIDE	OL1=ProductA(Action=Provide)
	OL2=ProductB(Action=Provide)
		planItemA depends on planItemB

Product A has PRF and PDO relationship with B and PRF and PDO has source action and target action PROVIDE and PROVIDE	OL1=ProductA	planItemA depends on planItemB

Product A has PDO relationship with B having source action and target action PROVIDE and PROVIDE	OL1=ProductA(Action=Provide)
	OL2=ProductB(Action=Provide)
		planItemA depends on planItemB

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

Product Configuration	Order	Plan

Product A has PDO relationship with B having SA & TA as PROVIDE & PROVIDE. SequenceDirection is AFTER.	OL1=ProductA(Action=Provide)
	OL2=ProductB(Action=Provide)
		Two planitem having planItemA depends on planItemB

Product Configuration	Order	Plan

Product A has PDO relationship with B having SA & TA as PROVIDE & PROVIDE. SequenceDirection is BEFORE.	OL1=ProductA(Action=Provide)
	OL2=ProductB(Action=Provide)
		Two planitem having planItemB depends on planItemA

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

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

Product Specification Field Decomposition

Each product has a modeled set of characteristics within a product catalogue. When a product is decomposed to a plan item, the default and the instance characteristics are copied over into the User Defined Fields (UDFs) of every plan item. This allows the information to be reused later when the plan item is executed.

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

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

Custom Action based Product Decomposition and Characteristic Inclusion

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

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

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

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

Scenario for the Custom Action Based Product Decomposition

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

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

Data Model Configuration	Order	Plan

Action repository has record with ID as `HomeMove` and recordtype as PROVIDE. Product B has PCO relationship with P1, P2, P3 with autoprovision=true
P1.PCO.ActionID=null
P2.PCO.ActionID=PROVIDE
P3.PCO.ActionID=`HomeMove`
	OL=B(PROVIDE)	There are three planItems:
		• B
		• P1
		• P2
		B depends on P1 & P2

	OL=B(UPDATE)	There are two planItems: • B • P1 B depends on P1
	OL=B(HomeMove)	There are two planItems: • B • P3 B depends on P3

Group Record Modeling

The Fulfillment Catalog's Group Record feature provides better ways to define, model and design products. This is a design time feature. The following Group related relationship attributes are moved to the record level:

- GroupNumber
- GroupOptional
- GroupMinQty
- GroupMaxQty

The moved attributes are put under the new attribute group called `GROUPINFO` of the `PRODUCT` repository.

When you publish the data model from Fulfillment Catalog to the Fulfillment system, the group information within this record is re-de normalized back to the child relationships. For the modeling purpose, the group record is neither a saleable nor an actionable record. It is a container to express group rules. Therefore, when expressing the product model to an external system, this container group is removed from the model and expressed using the same product schema.

Record Status Attribute

The Record Status attribute is maintained against every record of Fulfillment Catalog that holds the status of the record.

Even though MDM maintains the CONFIRMED or UNCONFIRMED state for every record, the attribute Record Status will be available, and depending upon the 'Record Status' value it will be updated or assigned automatically. This means that every CONFIRMED record should have a recordstatus="Active" and every UNCONFIRMED records should have a recordstatus ="Testing".

You can add additional states depending on the workflow. The value of 'Record Status' attribute has to be assigned and it is only available in the View mode, which means an end user cannot modify the value of the 'Record Status'.

The record status values will be set based on the following parameters:

- `Testing`: Assign the value `Testing` when creating a record or editing a record.
- `Active`: Assign or update the value `Active` when confirming a record as a part of the workflow.
- `Inactive`: Assign or update the value `Testing` when deleting a record as a part of the workflow.

Workflow Changes

Fulfillment Catalog has customized the two existing MDM workflows `wfin26productaddapprovalv3.xml` and `wfin26producteditapprovalv3.xml` in the `$MQ_COMMON_DIR/<enterprisename>/workflow` directory.

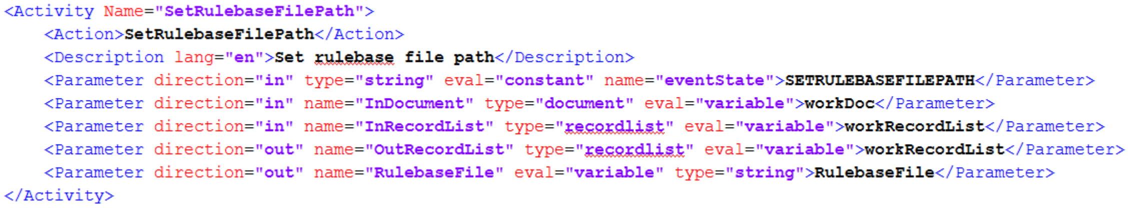
The `wfin26productaddapprovalv3.xml` workflow file is called when adding a new record or when approving an unconfirmed record. The `wfin26producteditapprovalv3.xml` workflow file is called when we edit an existing record or while deleting a record.

Customization on wfin26productaddapprovalv3.xml Workflow

Two new activities has been added in the workflow before the workflow calls the `UpdateRecordStateAsApproved` activity which will confirms the state of the record in MDM. They are `SetRulebaseFilePath` and `EvaluateRuleBase`.

Below is the `SetRulebaseFilePath` activity - which basically set the absolute rule base file path.

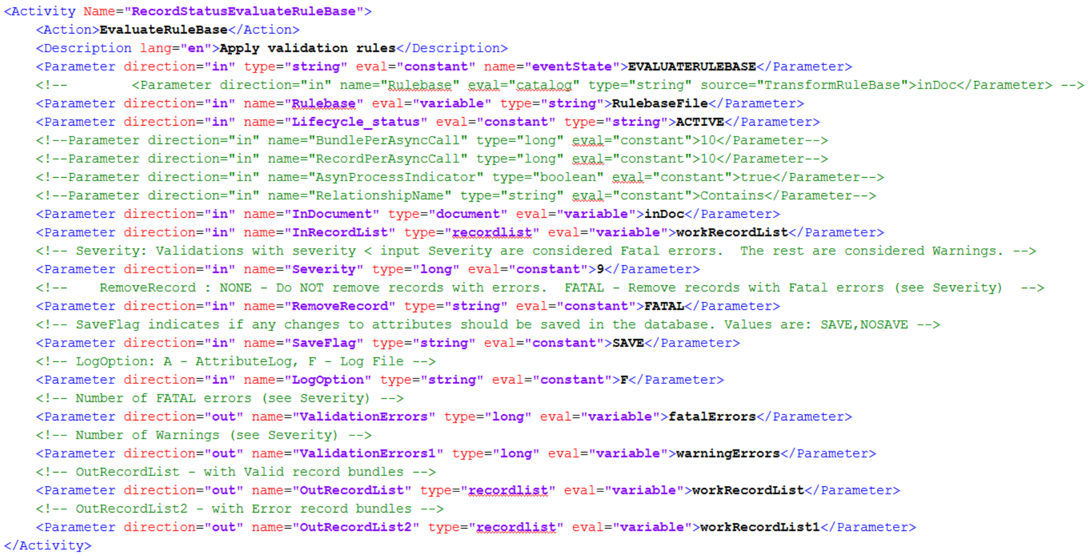
SetRulebaseFilePath Activity for wfin26productaddapprovalv3.xml Workflow



```
<Activity Name="SetRulebaseFilePath">
    <Action>SetRulebaseFilePath</Action>
    <Description lang="en">Set rulebase file path</Description>
    <Parameter direction="in" type="string" eval="constant" name="eventState">SETRULEBASEFILEPATH</Parameter>
    <Parameter direction="in" name="InDocument" type="document" eval="variable">workDoc</Parameter>
    <Parameter direction="in" name="InRecordList" type="recordlist" eval="variable">workRecordList</Parameter>
    <Parameter direction="out" name="OutRecordList" type="recordlist" eval="variable">workRecordList</Parameter>
    <Parameter direction="out" name="RulebaseFile" eval="variable" type="string">RulebaseFile</Parameter>
</Activity>
```

Below is the `EvaluateRuleBase` activity - which basically calls or executes the rule base file path.

EvaluateRulebase Activity for wfin26productaddapprovalv3.xml Workflow



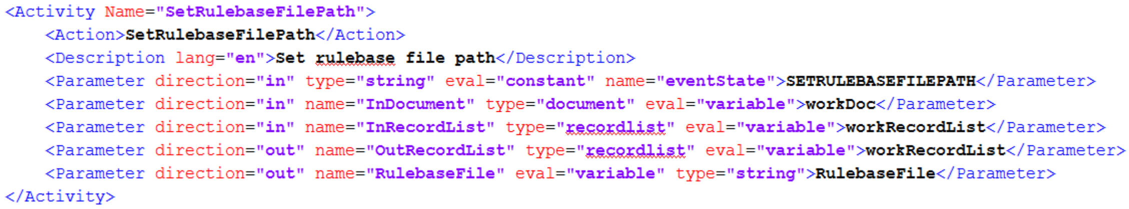
```xml
<Activity Name="RecordStatusEvaluateRuleBase">
    <Action>EvaluateRuleBase</Action>
    <Description lang="en">Apply validation rules</Description>
    <Parameter direction="in" type="string" eval="constant" name="eventState">EVALUATERULEBASE</Parameter>
    <!--        <Parameter direction="in" name="Rulebase" eval="catalog" type="string" source="TransformRuleBase">inDoc</Parameter> -->
    <Parameter direction="in" name="Rulebase" eval="variable" type="string">RulebaseFile</Parameter>
    <Parameter direction="in" name="Lifecycle_status" eval="constant" type="string">ACTIVE</Parameter>
    <!--Parameter direction="in" name="BundlePerAsyncCall" type="long" eval="constant">10</Parameter-->
    <!--Parameter direction="in" name="RecordPerAsyncCall" type="long" eval="constant">10</Parameter-->
    <!--Parameter direction="in" name="AsynProcessIndicator" type="boolean" eval="constant">true</Parameter-->
    <!--Parameter direction="in" name="RelationshipName" type="string" eval="constant">Contains</Parameter-->
    <Parameter direction="in" name="InDocument" type="document" eval="variable">inDoc</Parameter>
    <Parameter direction="in" name="InRecordList" type="recordlist" eval="variable">workRecordList</Parameter>
    <!-- Severity: Validations with severity < input Severity are considered Fatal errors.  The rest are considered Warnings. -->
    <Parameter direction="in" name="Severity" type="long" eval="constant">9</Parameter>
    <!--    RemoveRecord : NONE - Do NOT remove records with errors.  FATAL - Remove records with Fatal errors (see Severity)  -->
    <Parameter direction="in" name="RemoveRecord" type="string" eval="constant">FATAL</Parameter>
    <!-- SaveFlag indicates if any changes to attributes should be saved in the database. Values are: SAVE,NOSAVE -->
    <Parameter direction="in" name="SaveFlag" type="string" eval="constant">SAVE</Parameter>
    <!-- LogOption: A - AttributeLog, F - Log File -->
    <Parameter direction="in" name="LogOption" type="string" eval="constant">F</Parameter>
    <!-- Number of FATAL errors (see Severity) -->
    <Parameter direction="out" name="ValidationErrors" type="long" eval="variable">fatalErrors</Parameter>
    <!-- Number of Warnings (see Severity) -->
    <Parameter direction="out" name="ValidationErrors1" type="long" eval="variable">warningErrors</Parameter>
    <!-- OutRecordList - with Valid record bundles -->
    <Parameter direction="out" name="OutRecordList" type="recordlist" eval="variable">workRecordList</Parameter>
    <!-- OutRecordList2 - with Error record bundles -->
    <Parameter direction="out" name="OutRecordList2" type="recordlist" eval="variable">workRecordList1</Parameter>
</Activity>
```

Customization on wfin26producteditapprovalv3.xml Workflow

Two new activities has been added in the workflow before the workflow calls the `UpdateRecordStateAsApproved` activity which will confirms the state of the record in MDM. They are `SetRulebaseFilePath` and `EvaluateRuleBase`.

Below is the `SetRulebaseFilePath` activity - which basically set the absolute rule base file path.

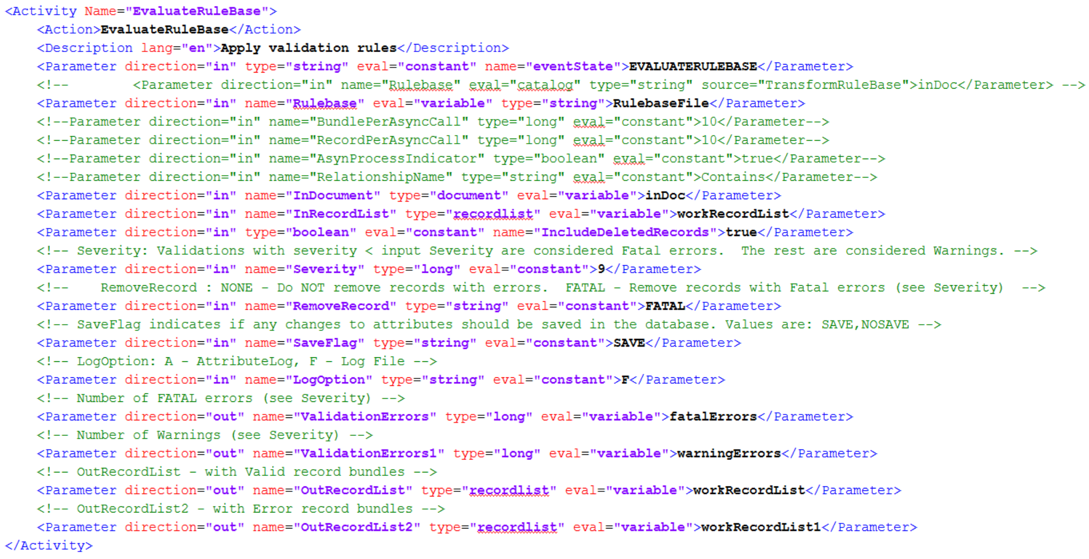
SetRulebaseFilePath Activity for wfin26producteditapprovalv3.xml Workflow



```xml
<Activity Name="SetRulebaseFilePath">
    <Action>SetRulebaseFilePath</Action>
    <Description lang="en">Set rulebase file path</Description>
    <Parameter direction="in" type="string" eval="constant" name="eventState">SETRULEBASEFILEPATH</Parameter>
    <Parameter direction="in" name="InDocument" type="document" eval="variable">workDoc</Parameter>
    <Parameter direction="in" name="InRecordList" type="recordlist" eval="variable">workRecordList</Parameter>
    <Parameter direction="out" name="OutRecordList" type="recordlist" eval="variable">workRecordList</Parameter>
    <Parameter direction="out" name="RulebaseFile" eval="variable" type="string">RulebaseFile</Parameter>
</Activity>
```

Below is the `EvaluateRuleBase` activity - which basically calls or executes the rule base file path.

EvaluateRulebase Activity for wfin26producteditapprovalv3.xml Workflow



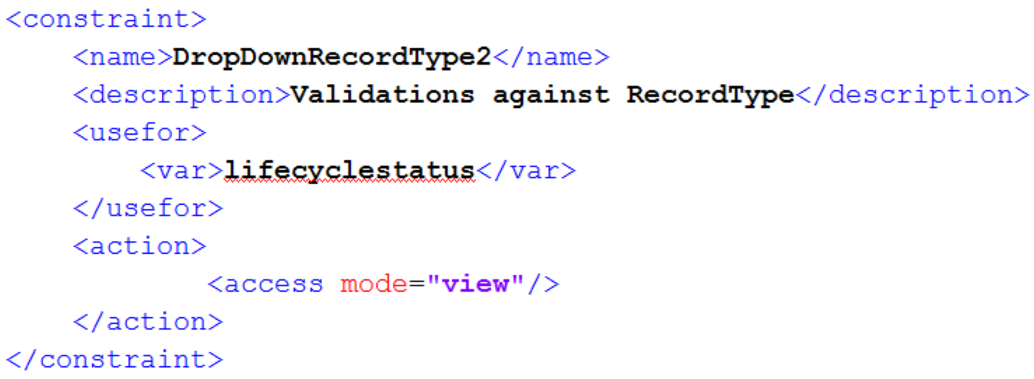
```xml
<Activity Name="EvaluateRuleBase">
    <Action>EvaluateRuleBase</Action>
    <Description lang="en">Apply validation rules</Description>
    <Parameter direction="in" type="string" eval="constant" name="eventState">EVALUATERULEBASE</Parameter>
    <!--        <Parameter direction="in" name="Rulebase" eval="catalog" type="string" source="TransformRuleBase">inDoc</Parameter> -->
    <Parameter direction="in" name="Rulebase" eval="variable" type="string">RulebaseFile</Parameter>
    <!--Parameter direction="in" name="BundlePerAsyncCall" type="long" eval="constant">10</Parameter-->
    <!--Parameter direction="in" name="RecordPerAsyncCall" type="long" eval="constant">10</Parameter-->
    <!--Parameter direction="in" name="AsynProcessIndicator" type="boolean" eval="constant">true</Parameter-->
    <!--Parameter direction="in" name="RelationshipName" type="string" eval="constant">Contains</Parameter-->
    <Parameter direction="in" name="InDocument" type="document" eval="variable">inDoc</Parameter>
    <Parameter direction="in" name="InRecordList" type="recordlist" eval="variable">workRecordList</Parameter>
    <Parameter direction="in" type="boolean" eval="constant" name="IncludeDeletedRecords">true</Parameter>
    <!-- Severity: Validations with severity < input Severity are considered Fatal errors.  The rest are considered Warnings. -->
    <Parameter direction="in" name="Severity" type="long" eval="constant">9</Parameter>
    <!--    RemoveRecord : NONE - Do NOT remove records with errors.  FATAL - Remove records with Fatal errors (see Severity)  -->
    <Parameter direction="in" name="RemoveRecord" type="string" eval="constant">FATAL</Parameter>
    <!-- SaveFlag indicates if any changes to attributes should be saved in the database. Values are: SAVE,NOSAVE -->
    <Parameter direction="in" name="SaveFlag" type="string" eval="constant">SAVE</Parameter>
    <!-- LogOption: A - AttributeLog, F - Log File -->
    <Parameter direction="in" name="LogOption" type="string" eval="constant">F</Parameter>
    <!-- Number of FATAL errors (see Severity) -->
    <Parameter direction="out" name="ValidationErrors" type="long" eval="variable">fatalErrors</Parameter>
    <!-- Number of Warnings (see Severity) -->
    <Parameter direction="out" name="ValidationErrors1" type="long" eval="variable">warningErrors</Parameter>
    <!-- OutRecordList - with Valid record bundles -->
    <Parameter direction="out" name="OutRecordList" type="recordlist" eval="variable">workRecordList</Parameter>
    <!-- OutRecordList2 - with Error record bundles -->
    <Parameter direction="out" name="OutRecordList2" type="recordlist" eval="variable">workRecordList1</Parameter>
</Activity>
```

Uneditable Record Status

The value of record status is uneditable and the value will be assigned, automatically, depending on the state of record.

This non-editability is achieved by having `VIEW` mode constraint in the catalog validation file for the repositories which has record status as attribute.

Uneditable Record Status



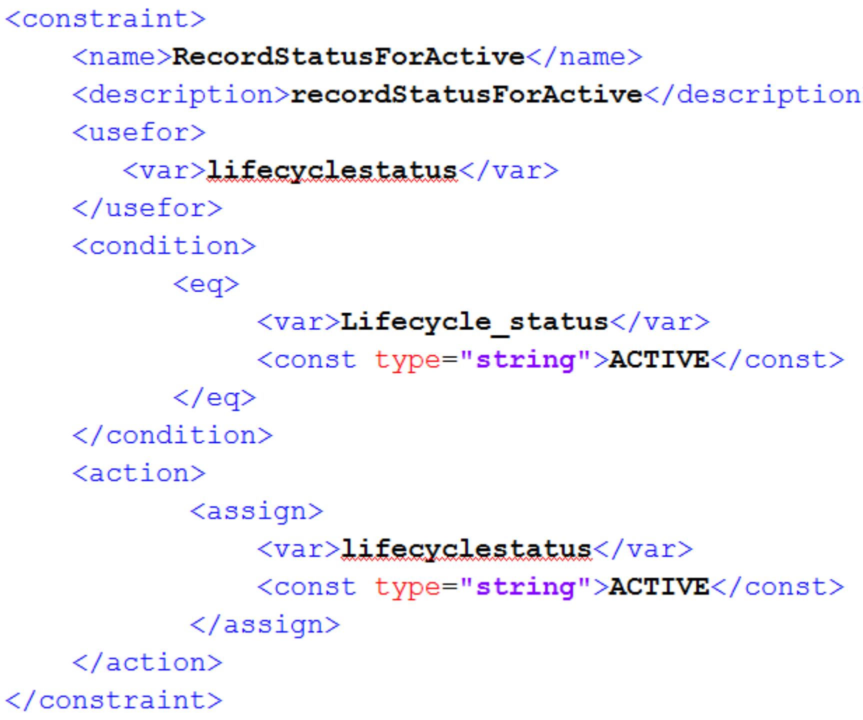
```
<constraint>
    <name>DropDownRecordType2</name>
    <description>Validations against RecordType</description>
    <usefor>
        <var>lifecyclestatus</var>
    </usefor>
    <action>
            <access mode="view"/>
    </action>
</constraint>
```

Assign ACTIVE to Confirmed Record

As a part of the workflow that confirms a record, update the record status to `ACTIVE`.

When a record is approved while creating or adding, `wfin26productaddapprovalv3.xml` will be invoked which has evaluate rule base activity to run the catalog validation file where the following constraint is executed to assign or update the record status to `ACTIVE`.

Assign ACTIVE to Confirmed Record



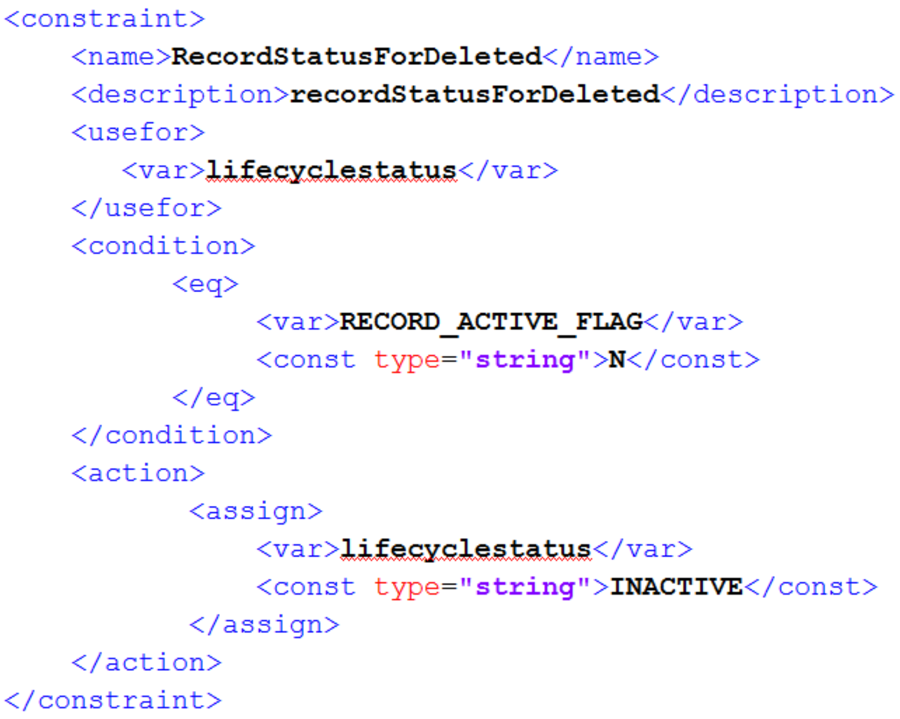
```
<constraint>
    <name>RecordStatusForActive</name>
    <description>recordStatusForActive</description>
    <usefor>
       <var>lifecyclestatus</var>
    </usefor>
    <condition>
          <eq>
               <var>Lifecycle_status</var>
               <const type="string">ACTIVE</const>
          </eq>
    </condition>
    <action>
           <assign>
               <var>lifecyclestatus</var>
               <const type="string">ACTIVE</const>
           </assign>
    </action>
</constraint>
```

Assign INACTIVE to a Deleted Record

As a part of the workflow that deletes a record, update the record status to `INACTIVE`.

When a record is deleted then `wfin26producteditapprovalv3.xml` will be invoked which has evaluate rule base activity to run the catalog validation file where below constraint is executed to assign or update the record status to `INACTIVE`.

Assign INACTIVE to a Deleted Record



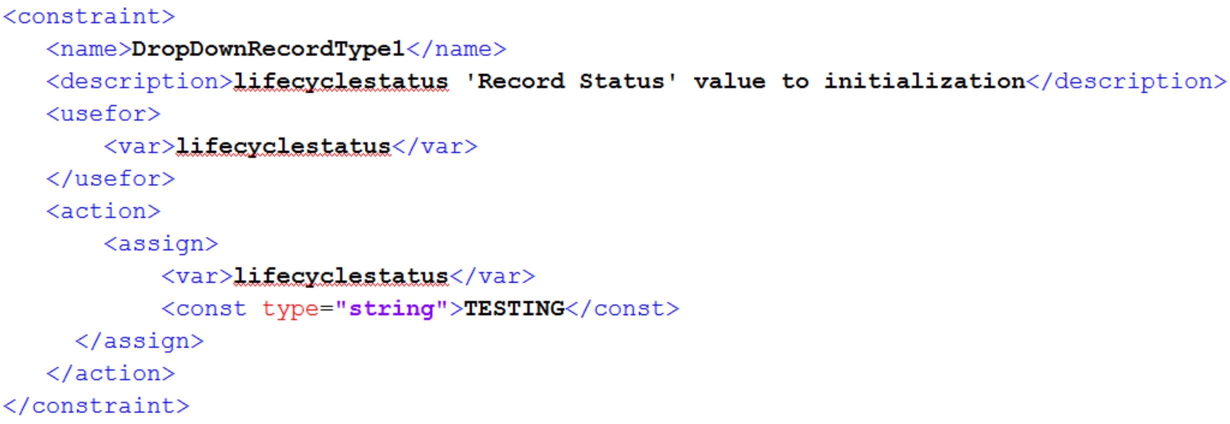
```
<constraint>
    <name>RecordStatusForDeleted</name>
    <description>recordStatusForDeleted</description>
    <usefor>
       <var>lifecyclestatus</var>
    </usefor>
    <condition>
          <eq>
               <var>RECORD_ACTIVE_FLAG</var>
               <const type="string">N</const>
          </eq>
    </condition>
    <action>
           <assign>
               <var>lifecyclestatus</var>
               <const type="string">INACTIVE</const>
           </assign>
    </action>
</constraint>
```

Assign TESTING to a Creating Record or Unconfirmed Record

At the time of creating a record or editing a record, assigning the record status field to `TESTING`.

When a record is created which means in unconfirmed state then using the new record configuration file we assign record status to `TESTING` where below constraint is executed. Till the record is confirmed it will be in `TESTING`. Once the record is approved then `wfin26productaddapprovalv3.xml` will be invoked which assign from `TESTING` to `ACTIVE`.

Assign TESTING to a Creating Record or Unconfirmed Record



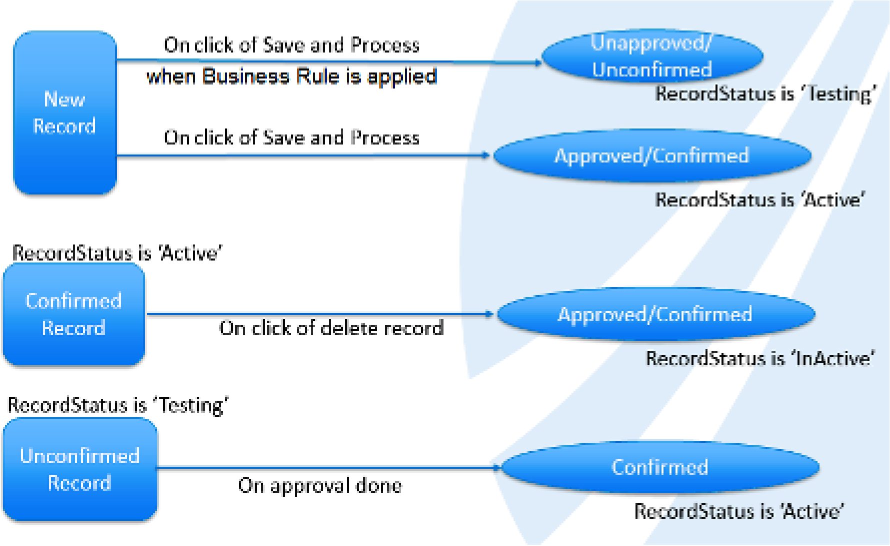
```
<constraint>
   <name>DropDownRecordType1</name>
   <description>lifecyclestatus 'Record Status' value to initialization</description>
   <usefor>
       <var>lifecyclestatus</var>
   </usefor>
   <action>
       <assign>
           <var>lifecyclestatus</var>
           <const type="string">TESTING</const>
     </assign>
   </action>
</constraint>
```

Record Status Value Transitions in Normal Use Cases

The normal use cases considered are record creation, approval, and deletion.

The following diagram shows how value transitions occur in normal use cases:

Record Status Value Transitions in Normal Use Cases



Metadata Repositories Holding the Record Status Attribute

The list of metadata repositories that holds the record status attribute are as follows:

- `ACTION`
- `ALLOWANCE`
- `CATALOG`
- `CATEGORY`
- `CHARATERISTIC`
- `CHARACTERISTICDATATYPE`
- `DISCOUNT`
- `MIGRATION`
- `MILESTONE`
- `PARTY`
- `PARTYEXTENSION`
- `PLANFRAGMENT`
- `POLICYRULE`
- `PRICE`
- `PRODUCT`
- `PROJECTTAG`
- `REQUIRES_PRODUCT`
- `RULE`
- `RULECONDITION`
- `RULEPARAMETER`

- `SEGMENT`

Building Blocks

The Building Block feature is a catalog structure that defines the Product repository and its relationships. There are no limits on the hierarchy levels of the Building Block. The hierarchy is identified, in the Fulfillment Catalog, using the ProductComprisedOf relationship.

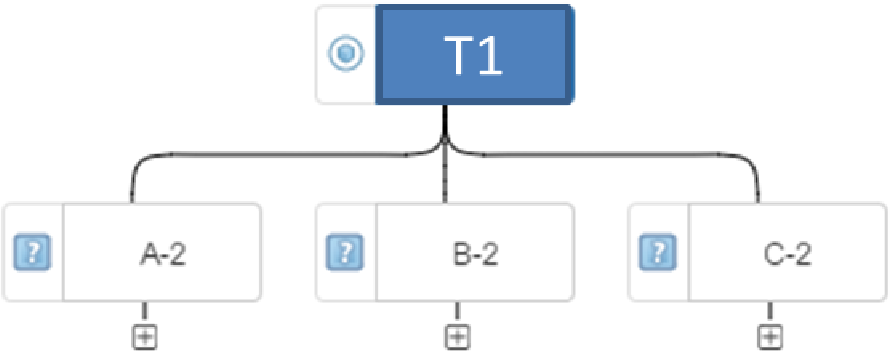
You can use the Building Block feature to define different characteristics for the parent and child product. You can also define the relationships between single products or group of products.

Use Case for Building Blocks

Consider a case of a building block where the ProductComprisedOf relationship with three records are defined. Since this is a building block, you will need to create multiple such structures with only a change in root the node.

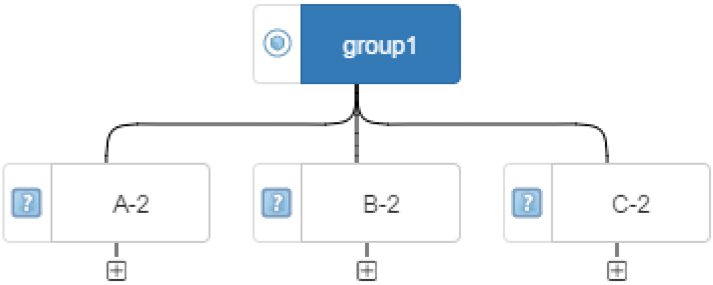
In absence of the Template feature, duplication effort will be required to create a new record and link those three children nodes with the ProductComprisedOf relationship.

Building Blocks Example using Templates



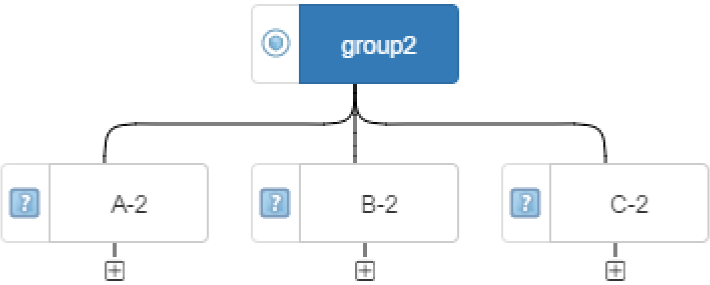
You can use the Template feature to create a pattern with one root template record and three non-template records. The root record then connects three children with a ProductComprisedOf relationship. This pattern can further be instantiated to create multiple hierarchies as shown in the following image:

Template Instance for Building Blocks - Group 1



In this instance the root template record T1 is instantiated and named as group1.

Template Instance for Building Blocks - Group 2



In this instance the root template record T1 is instantiated and named as group2.

Template-based Product Model

A Template is a record in the Product repository. A product is a template product only if the value of the attribute IsTemplate is TRUE.

The template attribute was introduced because there was a need to have a system where a pattern was created and then that pattern was re-used to create real products into the system. A template product reduces the effort in creating multiple similar bundles.

The template feature also integrates with the Building Blocks feature.

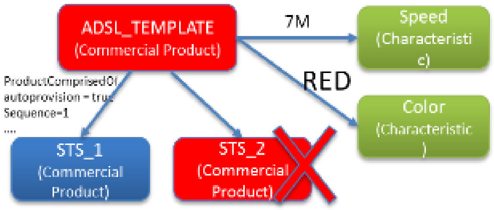
Difference between Template and Non-template Record

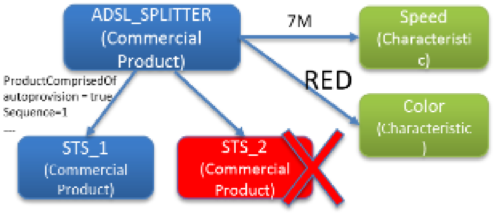
The differences between template and non-template record are as follows:

Template Record	Non-template Record
Can be instantiated.	Cannot be instantiated. If a non-template record is selected for instance creation, an alert will pop up.
Cannot be published during Publish Model as these are not real products which can be ordered.	Can be exported using Full or Partial Export feature. This flexibility is provided to enable copy of template records as well from one environment to another.

Validation Rules for Template Record

There are some constraints for introducing a template. They are as follows:

Rule	Example
A template cannot be added under another template. ADSL_TEMPLATE is a template. Therefore, another template STS_2 cannot be added as a child. The bundle becomes invalid.	 ADSL_TEMPLATE (Commercial Product) 7M Speed (Characteristic) RED Color (Characteristic) ProductComprisedOf autoprovision = true Sequence=1 .... STS_1 (Commercial Product) STS_2 (Commercial Product)

Rule	Example
A template cannot be added under a non-template record. ADSL_SPLITTER is a non-template. Therefore, a template STS_2 cannot be added as a child. The bundle becomes invalid.	

Template Filter during Publish Catalog

The Template functionality impacts Publish Catalog feature because the Publish Catalog does not publish any records, in XML, that are template records.

This has been achieved by a rule that is executed for each and every record during the Publish Catalog execution.

```
<constraint>
    <name>IsTemplateRecord</name>
    <description>Checks record is template </description>
    <condition>
        <and>
            <eq>
                <var>CATALOG_NAME</var>
                <const type="string">PRODUCT</const>
            </eq>
            <defined>
                <var>IsTemplate</var>
            </defined>
        </and>
    </condition>
    <action>
        <check>
            <explanation>Record is template.</explanation>
            <!--reject record when condition is evaluated to false-->
            <neq>
                <var>IsTemplate</var>
                <const type="boolean">true</const>
            </neq>
        </check>
    </action>
</constraint>
```

The rule is executed only for the PRODUCT repository record as only these records can be defined as a template.

Use Cases for Template Instances

The following are some key points before you refer the Use Cases:

- The instance creation screen as similar to that of Add Record screen except that the screen is pre-populated with data of the selected template.
- In all the use cases, when an instance of template is being created, the PRODUCTID and PRODUCTIDEXT are pre-populated with COPY-<template's PRODUCTID> and COPY-<template's PRODUCTIDEXT> respectively.
- Only a new copy of only template record is created. If template record has any relationship (implicit or explicit), only those relationships are copied to the instance. Child records are never copied. This means, the instance will have new relationships with the old child records.
- While creating an instance, you can add new relationship or relationships in addition to existing relationship or relationships, which were defined for the template, and will be added by default to

the instance, remove any such default relationship(s) or modify the attributes of such default relationship.

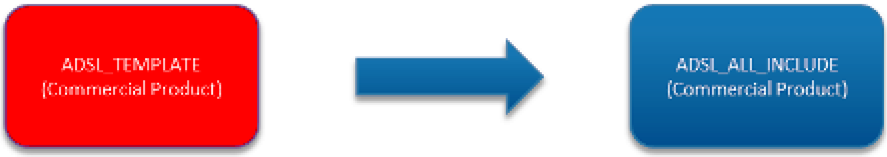
When a template record is instantiated, the new root record will have the value of IsTemplate record attribute as FALSE by default.

Simple Template with No Relationships

There is a template record ADSL_TEMPLATE which has no relationship. When this template is selected for instance creation, a screen is displayed, which is pre-filled with data of ADSL_TEMPLATE.

You can change any attribute. The following image indicates that the PRODUCTID of the instance is changed to ADSL_ALL_INCLUDE:

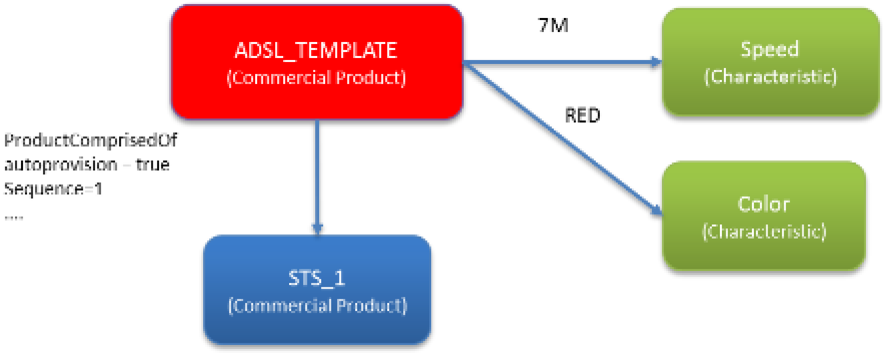
Simple Template with No Relationships



Template with Relationships

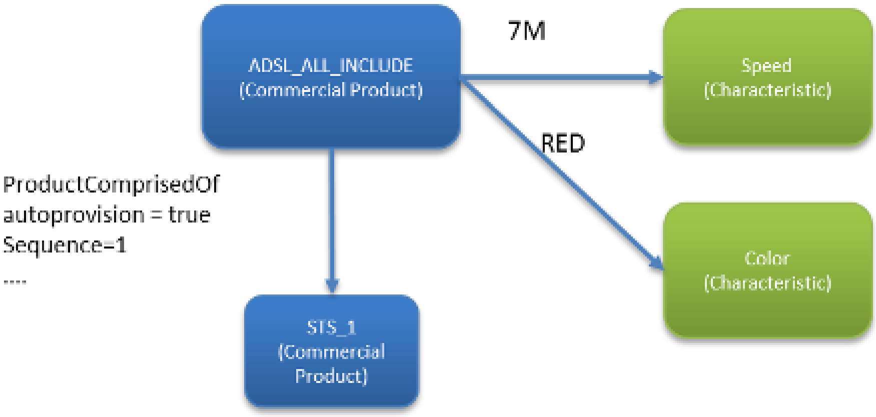
The following image is an example of Template with Relationship:

Template with Relationships



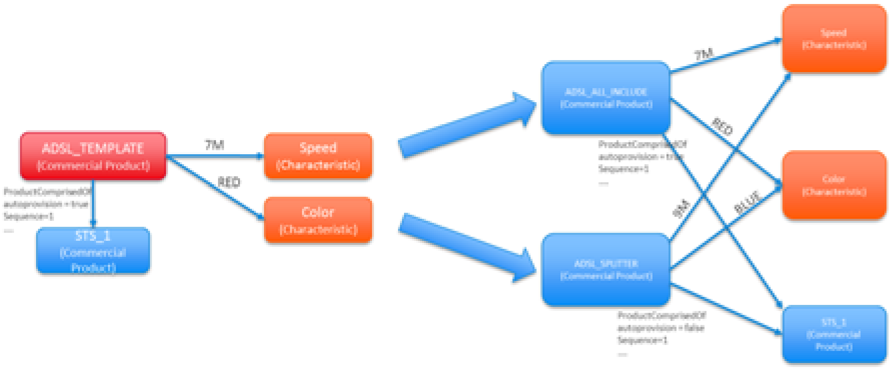
A template ADSL_TEMPLATE has two Characteristics and one ProductComprisedOf relationship. When this template is instantiated without any changes in relationship (addition, modification, or deletion), the instance model will look like the following image:

Template with Relationship when Template is Instantiated



The following image shows the representation of the result of the template being instantiated twice:

Template with Relationship when Template is Instantiated Twice



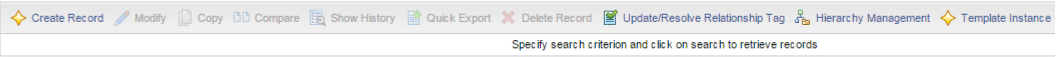
Update/Resolve Relationship Tag

This feature helps in automating the tagging service for relationships. This feature is also termed as resolving relationship tags because of the fact that it does tagging of the relationship based on parent and child record tags. It works only when one or more records are selected using the check boxes.

When clicked on Update Tag(s), it update/resolve the tags of all relationships which either starts from or ends at the selected record. While updating the tags, the previous tag would be appended with the common tag(s) between the parent and child record, ensuring that the tags do not duplicate. And this cycle continues for all selected record.

This feature is added on the MDM search result page.

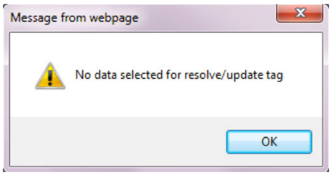
Update/Resolve Relationship Tag Button



Specify search criterion and click on search to retrieve records

The button is always enabled. But if clicked without selecting any record, it pops up an alert message.

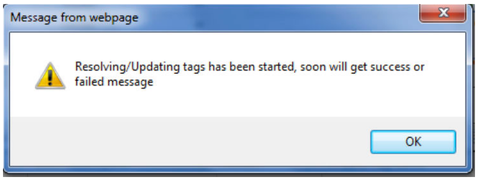
Update/Resolve Relationship Tag - Alert Popup



No data selected for resolve/update tag

When click on Update/Resolve Relationship Tag after selecting one or more records:

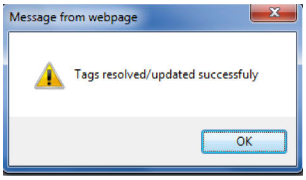
Update/Resolve Relationship Tag - In Progress



Resolving/Updating tags has been started, soon will get success or failed message

When the update/resolve process is successful:

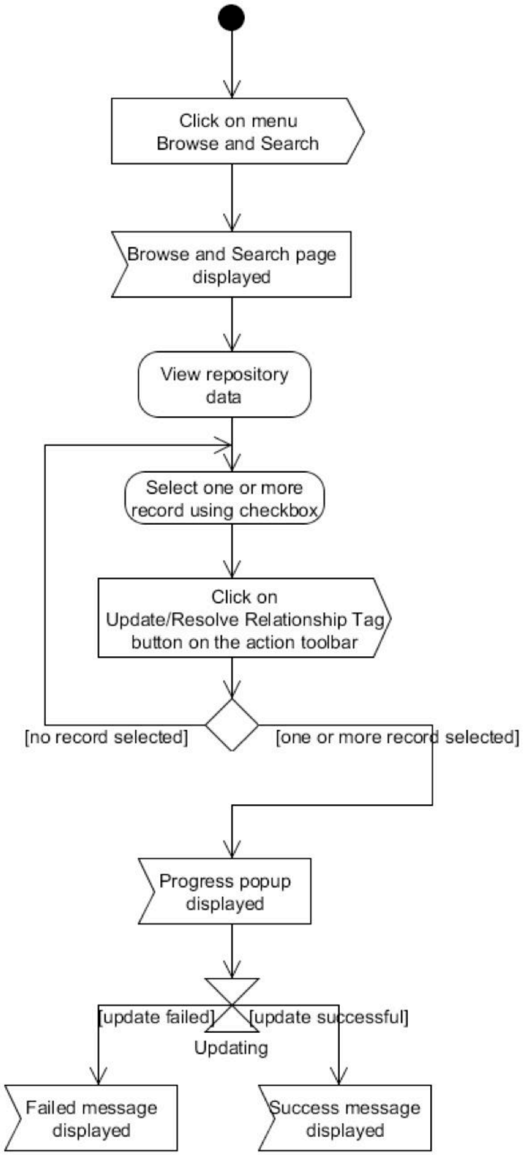
Update/Resolve Relationship Tag - Success



Tags resolved/updated successfuly

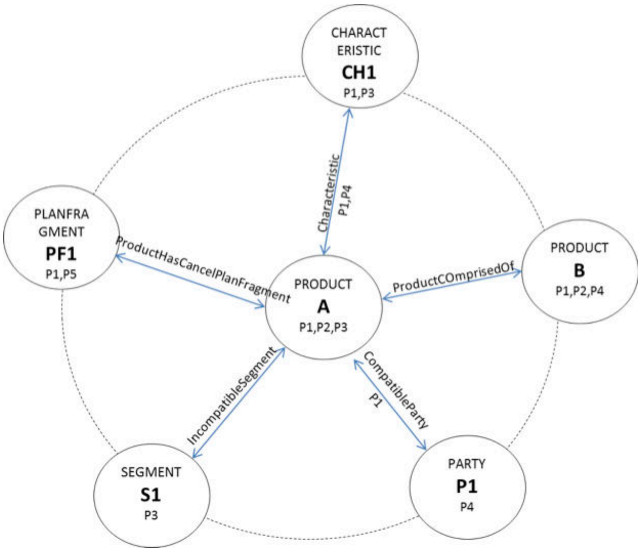
A normal flow is described below:

Update/Resolve Relationship Tag - Flow

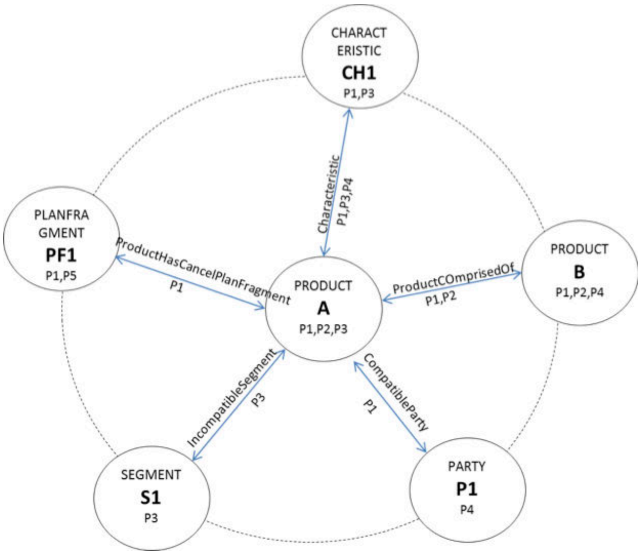

Click on menu Browse and Search
Browse and Search page displayed
View repository data
Select one or more record using checkbox
Click on Update/Resolve Relationship Tag button on the action toolbar
[no record selected]
[one or more record selected]
Progress popup displayed
[update failed]
[update successful]
Updating
Failed message displayed
Success message displayed

The feature behavior can be described with following diagrams:

Before Update/Resolve Relationship Tag



After Update/Resolve Relationship Tag



Administrator Options

You must have administrator privileges to perform the tasks listed in this topic.

The bulk model publisher does not require administrator privileges.

- Import from TIBCO Provisioning
- Export of Fulfillment Catalog Data
- Import of Fulfillment Catalog Data

Creating Users for Offer and Price Designer

A user can only access Offer and Price Designer if the user is assigned the role of Offer Designer in the Fulfillment Catalog enterprise. Only the user having the administrator role in the Fulfillment Catalog enterprise can create users who can access the Offer and Price Designer system.

The user having an administrator role only approves the offer using the Fulfillment Catalog enterprise. The user with the administrator role cannot access the Offer and Price Designer system. The user with the Offer Designer role can access the Offer and Price Designer system to create offers.

Prerequisites

To perform the following steps it is mandatory that the user has administrator role in the Fulfillment Catalog enterprise:

Procedure

1. Access the Fulfillment Catalog system and login using administrator credentials.
2. Click **Administration** > **User Accounts**.
   The User Accounts page opens.
3. Click **Create**.
   The Add User page opens.
4. Provide values for the following fields:

Fields	Description
User Name	Enter an appropriate user name. This will be used for logging in to Offer and Price Designer.
First Name	Enter the first name of the user.
Middle Name	Enter the middle name of the user.
Last Name	Enter the last name of the user.
Password	Enter a temporary password for the user.
Re-enter Password	Re-enter the temporary password for the user.

5. In the **Roles** panel, highlight the **Offer Designer** option from the **Available Roles** list and click the > button.
   The action performed will add **Offer Designer** role in the **Selected Roles** list.
6. Click the **Save** button.
   The new user is created and the user can access and use the *Offer and Price Designer* system.

User Role Matrix

The following table shows a comparison of the access rights, in the Fulfillment Catalog UI, between user having an Administrator role and user having Offer Designer role.

Menu/Role	Administrator	Offer Designer
Inbox	X	X
Administration	X	
Administration > My Company Profile	X	
Administration > User Accounts	X	
Administration > Roles	X	
Administration > Backend System Profiles	X	
Administration > Resource Security	X	
System Operations	X	
System Operations > Import Metadata	X	
System Operations > Export Metadata (Wizard)	X	
System Operations > Export Metadata (File)	X	
System Operations > Data Transfer	X	
Master Data	X	X
Master Data > Data Sources	X	X
Master Data > Repositories	X	X
Master Data > Mass Update	X	X
Master Data > Subset Rules	X	X
Master Data > Synchronization Profiles	X	X
Master Data > Synchronization Formats	X	X
Browse and Search	X	X
Fulfillment Catalog Operation	X	X
Fulfillment Catalog Operation > Import from TIBCO Provisioning	X	

Menu/Role	Administrator	Offer Designer
Fulfillment Catalog Operation > Publish Catalog	X	
Fulfillment Catalog Operation > Export FC Data	X	
Fulfillment Catalog Operation > Import FC Data	X	
Fulfillment Catalog Operation > Hierarchy Management	X	X
Business Processes	X	
Event Log	X	X
"X" denotes the availability of the menu for the role. In case an "X" is marked for the menu, it is not necessary that all menu items are available for the particular role. For example, the Administrator role has access to all menu items of the **Fulfillment Catalog Operation** menu but the Offer Designer role has access only to the **Hierarchy Management** menu item of the **Fulfillment Catalog Operation** menu.		

The user with the Administrator role cannot access the Offer and Price Designer system at all. The access is only granted to the user with the Offer Designer role. The user with the Administrator role can only create users who can access Offer and Price Designer system, and approve or activate the offers created by them.

Import from TIBCO Provisioning

Import from TIBCO Provisioning is a process of importing Customer Facing Services (CFS) record type from TIBCO Fulfillment Provisioning (FP) into TIBCO® Fulfillment Catalog (FC) on demand. It is a request-reply sequence.

TIBCO® Fulfillment Catalog is used to design various bundles of product or service offering and various offers. TIBCO Fulfillment Provisioning has its own catalog, called *FP Catalog* that allows you to configure the list of available CFS and all of them are diverted into Resource Facing Services (RFS). Fulfillment Provisioning supports different catalogues as part of the implementation of the Product and Service Order Management components maintaining Telecom provider products (product catalogue) and services (service catalogue).

For each order management component a Catalog defines the following information:

- managing order item instances such as services, products or resources
- authorized actions for these order items instances, such as create, remove, and modify

The synchronization is primarily a one-way process, where catalog data from Fulfillment Provisioning is extracted and imported into the TIBCO Fulfillment Catalog under the pre-defined rule set. TIBCO Fulfillment Provisioning responds to any incoming CFS Import Request messages by generating a `.csv` file of all the top-level records in its default catalog. At the Fulfillment Provisioning level, the catalog has the CFS and RFS data, but only the top level CFS record data is synchronized. The child records are not synchronized.

Configurations Before Starting the Fulfillment Provisioning Catalog Synchronization

You must set the following properties before starting the Fulfillment Provisioning Catalog Synchronization process.

These properties are added to the `$MQ_HOME/config/ConfigValues.xml` file after Fulfillment Catalog is configured as the TIBCO MDM plugin. For details, see the *Configuring Fulfillment Catalog* topic in the *TIBCO Fulfillment Catalog Installation and Configuration Guide*.

Property Name	Description	Value
FulfillmentProvisioning messaging initial context	FulfillmentProvisioning messaging initial context	`com.tibco.catalog.fulfillmentProvisioning.messaging.initialcontext`
FulfillmentProvisioning messaging endpoint	FulfillmentProvisioning messaging endpoint	`com.tibco.catalog.fulfillmentProvisioning.messaging.endpoint`
FulfillmentProvisioning Catalog messaging user name	FulfillmentProvisioning messaging user name	`com.tibco.catalog.fulfillmentProvisioning.messaging.username`
FulfillmentProvisioning Catalog messaging password	FulfillmentProvisioning messaging password	`com.tibco.catalog.fulfillmentProvisioning.messaging.password`

To configure the context factory of Fulfillment Provisioning messaging service, use the `com.tibco.catalog.fulfillmentProvisioning.messaging.initialcontext` property. The value of this property is described in the following table:

Message Service Provider	Context factory name
TIBCO EMS	`com.tibco.tibjms.naming.TibjmsInitialContextFactory`

Integration Between Fulfillment Catalog and Fulfillment Provisioning

The following configuration is required on TIBCO Enterprise Message Service:

1. Add Factories:
    a. Topic Connection Factory - `System/JSR264/ApplicationType/OrderManagement/Application/1-0;1-0-0;TIBCO-FOS-FP/Comp/TopicConnectionFactory`
    b. Queue Connection Factory - `System/JSR264/ApplicationType/OrderManagement/Application/1-0;1-0-0;TIBCO-FOS-FP/Comp/QueueConnectionFactory`
2. Create Queue:
    a. `System/JSR264/ApplicationType/OrderManagement/Application/1-0;1-0-0;TIBCO-FOS-FP/Comp/MessageQueue`
3. Create Topic:
    a. `System/JSR264/ApplicationType/OrderManagement/Application/1-0;1-0-0;TIBCO-FOS-FP/Comp/XVTEventTopic`

Restart Enterprise Message Service if factories are added in the `factories.conf` configuration file.

Synchronizing Fulfillment Provisioning Catalog

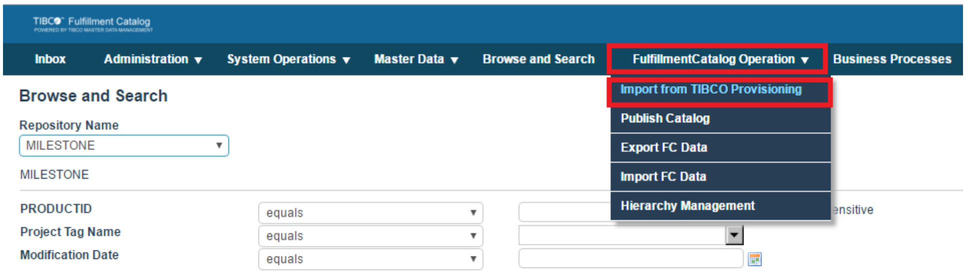
The Fulfillment Provisioning Catalog Synchronization process is initiated through a menu item **FulfillmentCatalog Operation** > **Import from TIBCO Provisioning**. This menu item is added to the menu only after the Fulfillment Catalog is configured as a plugin for TIBCO MDM. For security reasons, the menu item is visible only to the user with the `Administrator` privileges. For any change in the access rule, refer to the `Customizing Roles, Menus and Access Rights` in the *TIBCO MDM Customization Guide* document.

The nominal scenario is as follows (Service Order Data from Fulfillment Provisioning is imported into the Fulfillment Catalog PRODUCT repository. Additionally, the basic data is created into other repositories such as, ACTION, PLANFRAGMENT, MILESTONE, and CHARACTERISTIC, in order that service order data conforms to the Fulfillment Catalog rule):

Procedure

1. Got to **FulfillmentCatalog Operation** > **Import from TIBCO Provisioning**.

   Catalog Synchronization

   The application launches the Fulfillment Provisioning Catalog Synchronization process. In this process, Fulfillment Catalog makes a request for CFS data to Fulfillment Provisioning through a java message sent over a designated queue and wait for the response. The waiting time is set to 2 seconds. If reply is not received in time, Fulfillment Catalog terminates the synchronization process with a message `TIBCO Provisioning did not respond`. The delay can occur in the following two scenarios:

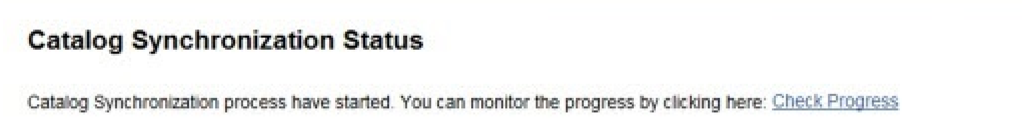
   - Fulfillment Provisioning is not running. Check with your administrator if Fulfillment Provisioning is deployed or not.
   - The Fulfillment Provisioning message queue listener is busy. Try again after some time.

   After Fulfillment Catalog receives the reply, it launches the `LoadImportAction` process of TIBCO MDM to push that data in various repositories. The affected repositories are:

   1. ACTION
   2. PRODUCT
   3. PLANFRAGMENT
   4. MILESTONE
   5. CHARACTERISTIC

2. Click **Check Progress** to open the log event where the progress or a status of import process is displayed. This event logging is similar to the MDM event logging.

Catalog Synchronization Status



Catalog Synchronization Status

Catalog Synchronization process have started. You can monitor the progress by clicking here: Check Progress

You may see an error or an unexpected screen, if you do not have Fulfillment Provisioning installed and configured on your machine.

Cases When Fulfillment Provisioning Synchronization Fails

The synchronization process can fail in the following cases:

1. If the required properties are not available in the `ConfigValues.xml` file or not set correctly. In this case, the process is terminated with the error message on the UI. For details on the configurable properties, see the Configuration before Starting the Fulfillment Provisioning Catalog Synchronization topic.
2. If Fulfillment Provisioning application is not running or busy.

Data to Synchronize

You can synchronize data by:

- ACTION Repository Data
- PRODUCT Repository Data

ACTION Repository Data

The details related to ACTION Repository Data are as follows:

TIBCO Fulfillment Catalog Attribute	TIBCO Fulfillment Provisioning Value	Comments
PRODUCTID	The Verb portion	create, modify, cancel
RECORD_TYPE	Update	hard-coded
SHORTDESC	FP managed Action	hard-coded

PRODUCT Repository Data

The details related to PRODUCT repository data are as follows:

TIBCO Fulfillment Catalog Attribute	TIBCO Fulfillment Provisioning Value	Comments
PRODUCTID	FP ID	
PRODUCTIDEXT		Fulfillment Provisioning Catalog Version
RECORD_TYPE	"CFS"	Hard-coded

TIBCO Fulfillment Catalog Attribute	TIBCO Fulfillment Provisioning Value	Comments
Name		same as PRODUCTID
Owner	"FP"	Hard-coded
SHORTDESC		
LONGDESC		
SingleUse		
MustComplete		
ConcurrentOrder		

Message Logs

All the messages are logged as per TIBCO MDM standards.

Export of Fulfillment Catalog Data

Export of Fulfillment Catalog data deals with:

- Export of Blank Template
- Export of Fulfillment Catalog Data using Graphical User Interface
- Customization Workflow for Export

Export of Blank Template

Fulfillment Catalog provides you the facility to export a blank template so that you can use the template to model your enterprise data.

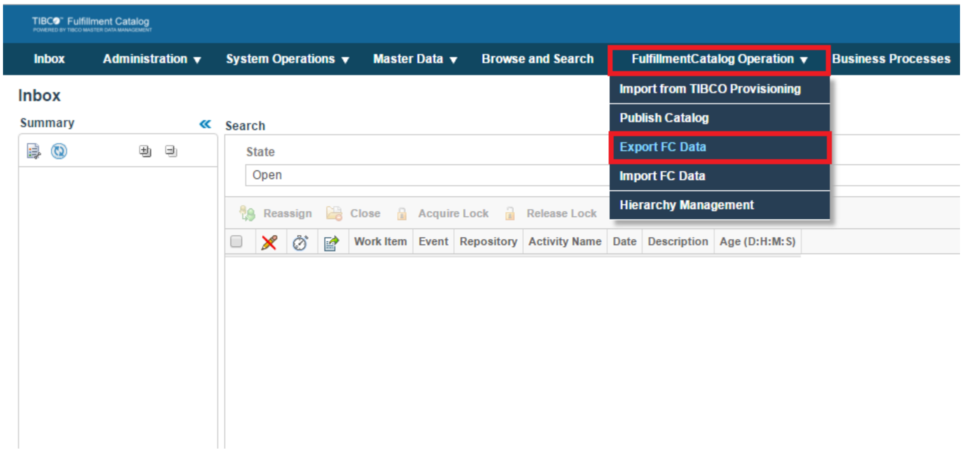
Accessing User Interface to Export a Blank Template

To export a blank template using Fulfillment Catalog user interface, perform the following steps:

Procedure

1. Click **FulfillmentCatalog Operation** > **Export FC Data**.

Accessing Export FC Data



2. The **Blank Template** option will be selected by default. Click the **Export** button. The blank template will be downloaded to your system.

Types of Export of Fulfillment Catalog Data

The following types of export can be performed using the Fulfillment Catalog user interface:

- Activating the BackwardCompatibleCsv Flag to Generate Backward Compatible CSVs
- Full Export of Fulfillment Catalog Data
- Partial Export of Fulfillment Catalog Data
- Enterprise Delta Export of Fulfillment Catalog Data

Activating the BackwardCompatibleCsv Flag to Generate Backward Compatible CSVs

A new parameter, named BackwardCompatibleCsv, has been introduced in the export workflow wfin26enterprisedataexportv1.xml.

By default the BackwardCompatibleCsv parameter is set to the value of FALSE for good performance. The BackwardCompatibleCsv parameter should be set to TRUE if backward compatible CSV files are to be generated. When BackwardCompatibleCsv is set to FALSE during export, there will not be any double quotation under relationship attribute CSVs.

Additionally, reverse relationships will not be exported. When the BackwardCompatibleCsv parameter is set to TRUE, during the export, there will be double quotation under relationship attribute CSVs and reverse relationships will also be exported.

Procedure

1. Access the `<MQ_COMMON_DIR>/<ENT_NAME>/workflow/` directory.
2. Open the file `wfin26enterprisedataexportv1.xml`.
3. Within the activity named **`SetParameters`**, locate the parameter with the name **`BackwardCompatibleCsv`**.

```
<Activity Name="SetParameters">
<Action>FCNoOperation</Action>
<Execution>SYNCHR</Execution>
<Parameter direction="out" eval="variable" type="string"
name="EnterpriseExportType">ExportType</Parameter>
<Parameter direction="out" eval="variable" type="string"
name="ProjectTags">Tags</Parameter>
<Parameter direction="in" type="string" eval="constant"
name="FeatureName">DataExport</Parameter>
<Parameter direction="in" name="BackwardCompatibleCsv" type="boolean"
```

```
eval="constant">false</Parameter>
</Activity>
```

4. By default the value of the `BackwardCompatibleCsv` parameter will be `FALSE`. Change the value of the flag to `TRUE`.

Full Export of Fulfillment Catalog Data

Full Export is a process of exporting all repository instances and relationship instances data into a csv format file. The Full Export feature is primarily used when there is a need to copy data from one enterprise to another.

The repositories involved in this process would be all the repositories that are present in the enterprise where from the process is invoked.

Full Export helps to replicate data in one environment in another environment. For example, if data is populated in a Development environment, and you are required to have the same data in a Test / Staging environment, you can use this feature to export the data from the Development environment and import it to your target environment.

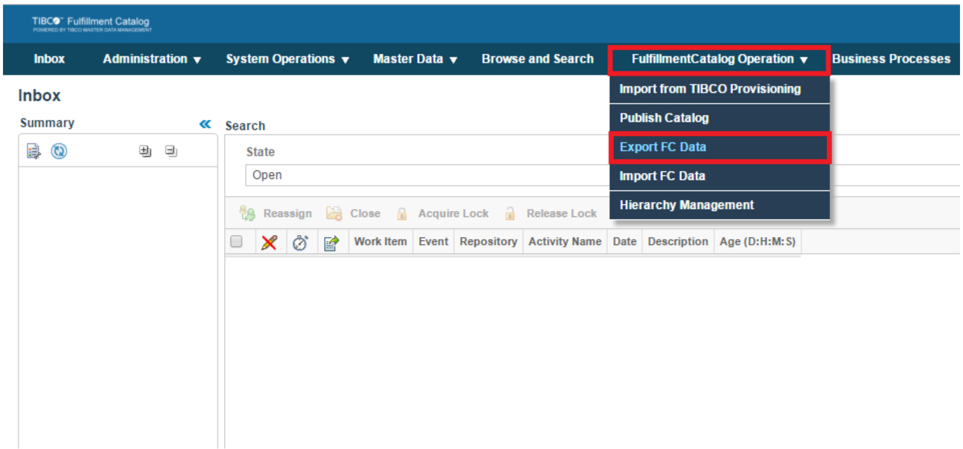
Accessing User Interface for Full Export of Fulfillment Catalog Data

To execute Full Export using Fulfillment Catalog user interface, perform the following steps:

Procedure

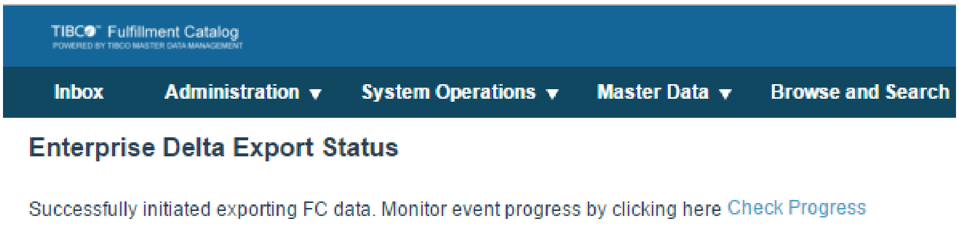
1. Click **FulfillmentCatalog Operation** > **Export FC Data**.

   Accessing Export FC Data



2. Select **Full/Partial Enterprise Data Export**.
3. Click **Export** button at the bottom of the screen.
   This internally initiates an export process on the enterprise. If the export process starts successfully, a success page is displayed with the message “Successfully initiated exporting FC data. Monitor event progress by clicking *Check Progress*".
4. Click *Check Progress* link to monitor the event progress.

Enterprise Data Export Status



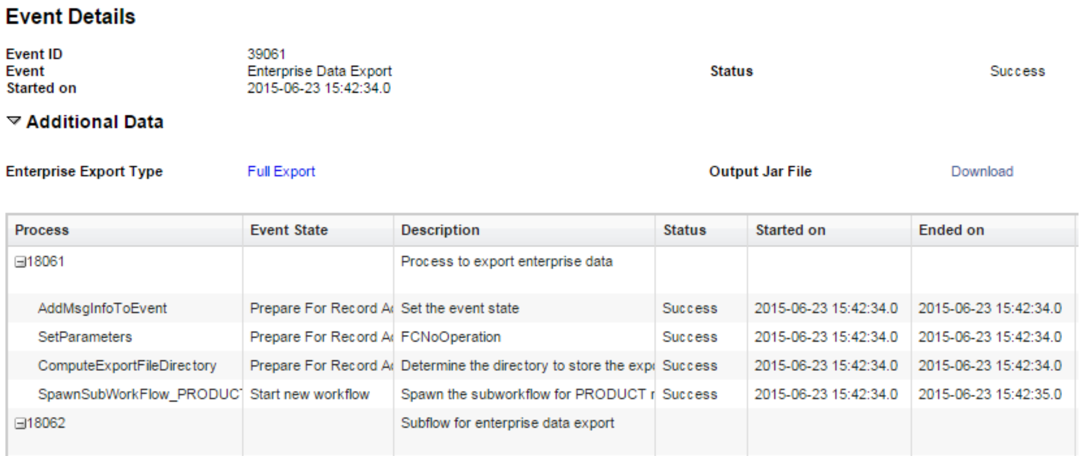
Enterprise Delta Export Status

Successfully initiated exporting FC data. Monitor event progress by clicking here Check Progress

> The success page (as above) is only an indication that the export process has been initiated successfully. The actual result of the process should be checked from the event log. If, for any reason, the process initiation fails, appropriate error message is displayed on the screen

The successful execution of the process creates a zip fie, containing one or more csv files inside, at the location represented by *FileName* attribute in the event log. The output file can also be downloaded from the event log.

Export Fulfillment Catalog Data Download



Event Details

Event ID Event Started on	39061 Enterprise Data Export 2015-06-23 15:42:34.0	Status	Success

▽ Additional Data

Enterprise Export Type	Full Export	Output Jar File	Download

Process	Event State	Description	Status	Started on	Ended on
⊟18061		Process to export enterprise data			
AddMsgInfoToEvent	Prepare For Record A	Set the event state	Success	2015-06-23 15:42:34.0	2015-06-23 15:42:34.0
SetParameters	Prepare For Record A	FCNoOperation	Success	2015-06-23 15:42:34.0	2015-06-23 15:42:34.0
ComputeExportFileDirectory	Prepare For Record A	Determine the directory to store the exp	Success	2015-06-23 15:42:34.0	2015-06-23 15:42:34.0
SpawnSubWorkFlow_PRODUC	Start new workflow	Spawn the subworkflow for PRODUCT r	Success	2015-06-23 15:42:34.0	2015-06-23 15:42:35.0
⊟18062		Subflow for enterprise data export			

Partial Export of Fulfillment Catalog Data

Partial Export is a process of exporting selective data from all the repository and relationships. The selection of data is based ProjectTagName attribute value.

Partial Export is primarily used when there is a need to copy one or more project data from one enterprise to another. The repositories involved in this process would be all the repositories that are present in the enterprise where from the process is invoked.

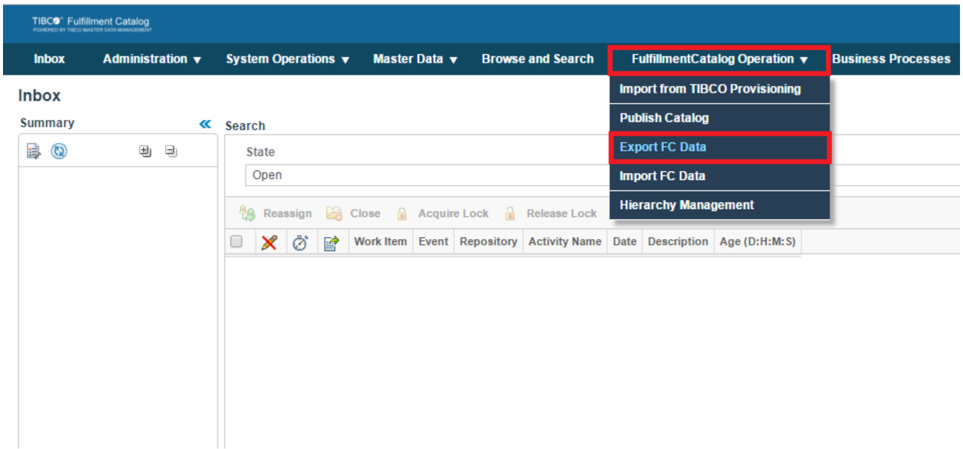
Accessing User Interface for Partial Export of Fulfillment Catalog Data

To execute Partial Export using Fulfillment Catalog user interface, perform the following steps:

Procedure

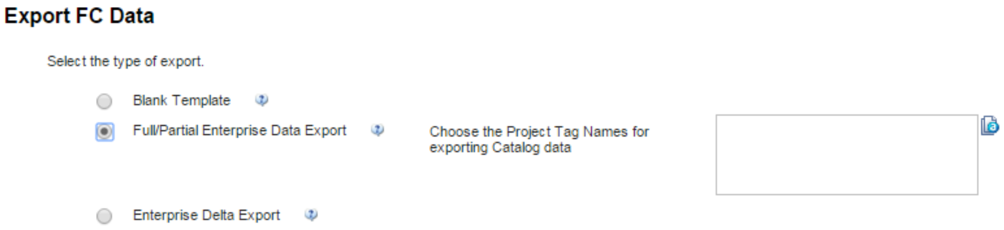
1. Click **FulfillmentCatalog Operation** > **Export FC Data**.

Accessing Export FC Data



2. Select **Full/Partial Enterprise Data Export**.

   Export FC Data Options



Export FC Data

Select the type of export.

○ Blank Template

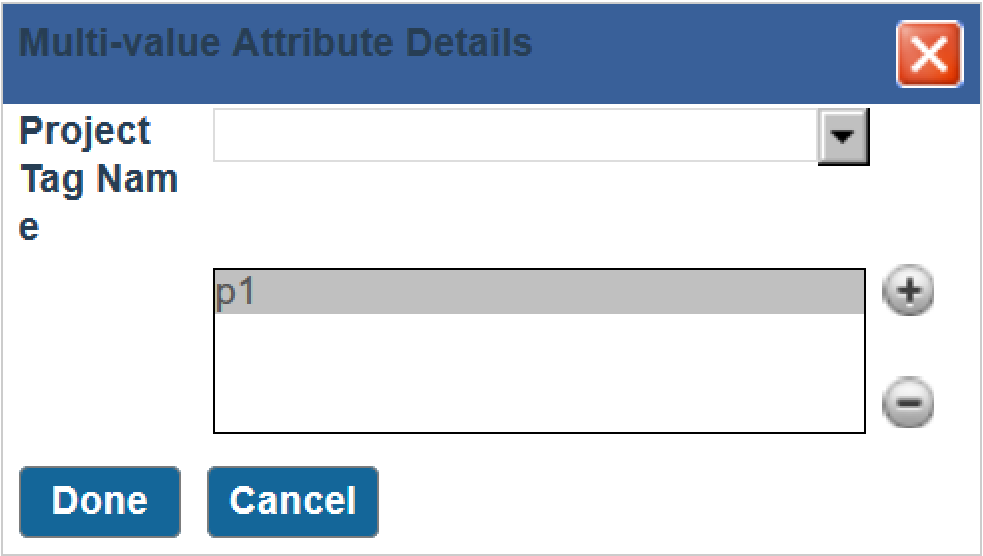
● Full/Partial Enterprise Data Export

Choose the Project Tag Names for exporting Catalog data

○ Enterprise Delta Export

3. Click the icon for the **Choose the Project Tag Names for exporting Catalog data** field. The **Multi-value Attribute Details** dialog opens.
4. Select the **Project Tag Name**, click the **+** sign and click the **Done** button.

   Multi-value Attribute Details Dialog Box

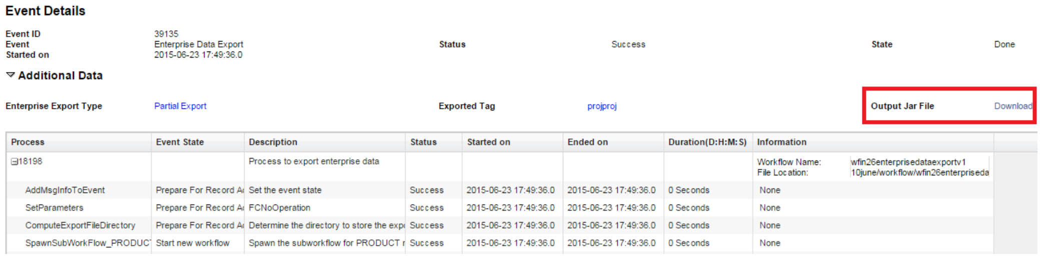


Project Tag Name

p1

5. When you click the *Export* label, the workflow is invoked. If the workflow executes successfully, it generates the csv files of the data (repository and relationship instance(s)) which matches the one or more selected tag names. If there are multiple tags selected, it is an OR between the tags while finding the match. The output export file is available for download using a link in the event log. The link to the output export file is illustrated as follows:

Link to Output Export File



Event Details

Event ID 39135
Event Enterprise Data Export
Started on 2015-06-23 17:49:36.0
Status Success
State Done

Additional Data

Enterprise Export Type Partial Export
Exported Tag projproj
Output Jar File Download

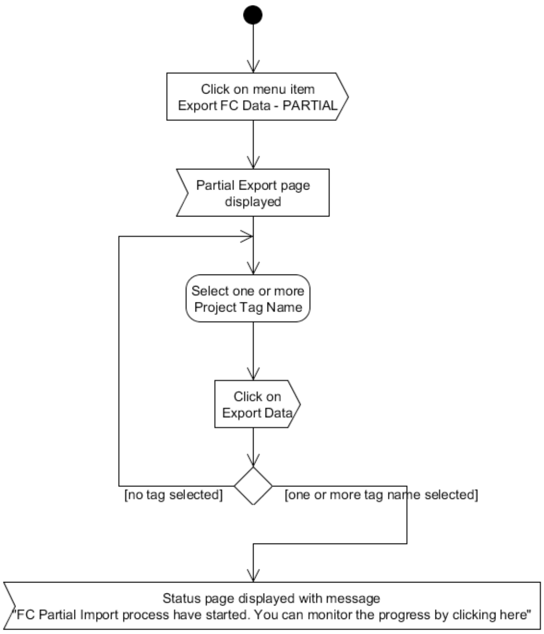
Process	Event State	Description	Status	Started on	Ended on	Duration(D:H:M:S)	Information
18198		Process to export enterprise data					Workflow Name: wfin26enterprisedataexportv1 File Location: 10june/workflow/wfin26enterpriseda
AddMsgInfoToEvent	Prepare For Record A	Set the event state	Success	2015-06-23 17:49:36.0	2015-06-23 17:49:36.0	0 Seconds	None
SetParameters	Prepare For Record A	FCNoOperation	Success	2015-06-23 17:49:36.0	2015-06-23 17:49:36.0	0 Seconds	None
ComputeExportFileDirectory	Prepare For Record A	Determine the directory to store the exp	Success	2015-06-23 17:49:36.0	2015-06-23 17:49:36.0	0 Seconds	None
SpawnSubWorkFlow_PRODUC	Start new workflow	Spawn the subworkflow for PRODUCT r	Success	2015-06-23 17:49:36.0	2015-06-23 17:49:36.0	0 Seconds	None

The download link for the partial export will be available only if the partial export is successful. Click the Download link corresponding to the Output JAR File field.

Partial Export of Fulfillment Catalog Data - Control Flow

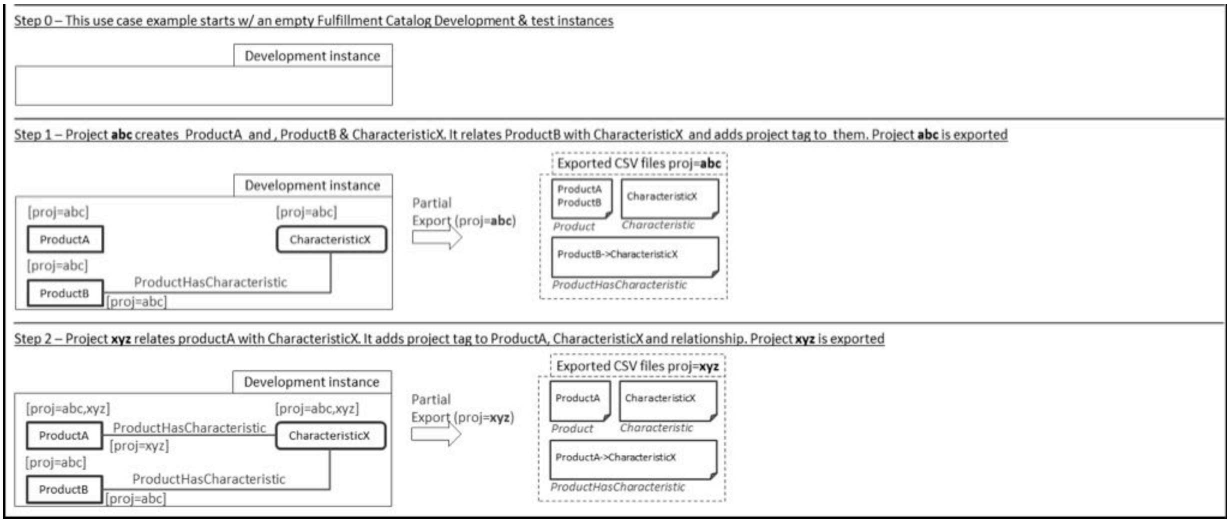
The control flow is described in the following diagram:

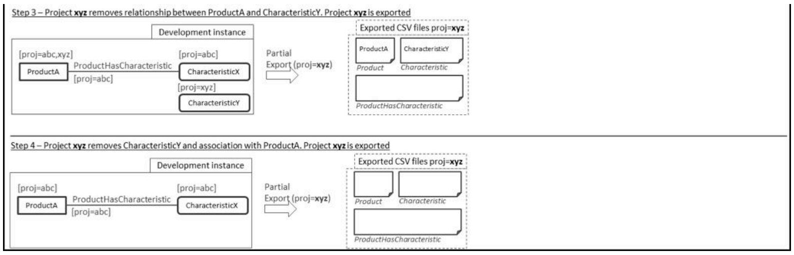
Partial Export of Fulfillment Catalog Data - Control Flow

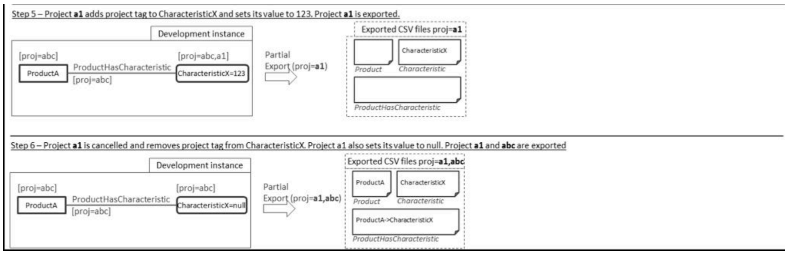

Click on menu item
Export FC Data - PARTIAL
Partial Export page
displayed
Select one or more
Project Tag Name
Click on
Export Data
[no tag selected]
[one or more tag name selected]
Status page displayed with message
"FC Partial Import process have started. You can monitor the progress by clicking here"

Partial Export of Fulfillment Catalog Data - Use Case 1

The following is the first use case for partial export of Fulfillment Catalog:

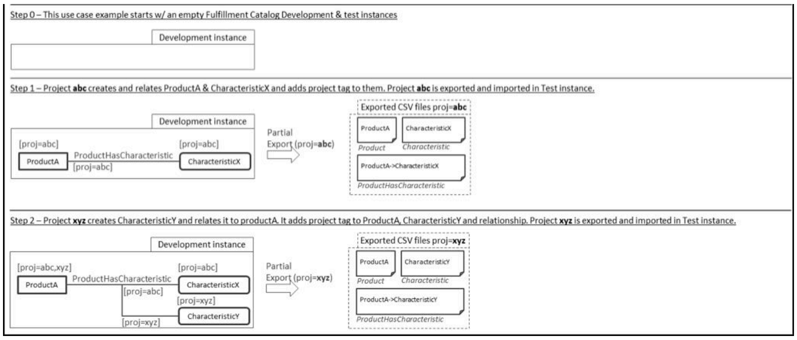

Step 0 – This use case example starts w/ an empty Fulfillment Catalog Development & test instances
Development instance
Step 1 – Project abc creates ProductA and , ProductB & CharacteristicX. It relates ProductB with CharacteristicX and adds project tag to them. Project abc is exported
Development instance
[proj=abc]
ProductA
[proj=abc]
CharacteristicX
[proj=abc]
ProductB
ProductHasCharacteristic
[proj=abc]
Partial Export (proj=abc)
Exported CSV files proj=abc
ProductA ProductB
Product
CharacteristicX
Characteristic
ProductB->CharacteristicX
ProductHasCharacteristic
Step 2 – Project xyz relates productA with CharacteristicX. It adds project tag to ProductA, CharacteristicX and relationship. Project xyz is exported
Development instance
[proj=abc,xyz]
ProductA
ProductHasCharacteristic
[proj=xyz]
[proj=abc,xyz]
CharacteristicX
[proj=abc]
ProductB
ProductHasCharacteristic
[proj=abc]
Partial Export (proj=xyz)
Exported CSV files proj=xyz
ProductA
Product
CharacteristicX
Characteristic
ProductA->CharacteristicX
ProductHasCharacteristic

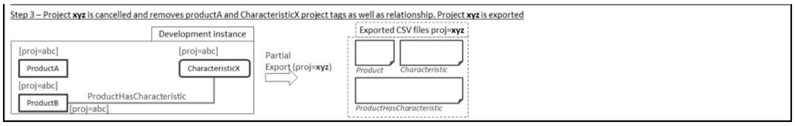



Step 5 – Project a1 adds project tag to CharacteristicX and sets its value to 123. Project a1 is exported.
Development instance
[proj=abc]
ProductA
ProductHasCharacteristic
[proj=abc]
[proj=abc,a1]
CharacteristicX=123
Partial Export (proj=a1)
Exported CSV files proj=a1
Product
CharacteristicX
Characteristic
ProductHasCharacteristic
Step 6 – Project a1 is cancelled and removes project tag from CharacteristicX. Project a1 also sets its value to null. Project a1 and abc are exported
Development instance
[proj=abc]
ProductA
ProductHasCharacteristic
[proj=abc]
[proj=abc]
CharacteristicX=null
Partial Export (proj=a1,abc)
Exported CSV files proj=a1,abc
ProductA
Product
CharacteristicX
Characteristic
ProductA->CharacteristicX
ProductHasCharacteristic

Partial Export of Fulfillment Catalog Data - Use Case 2

The following is the second use case for partial export of Fulfillment Catalog:

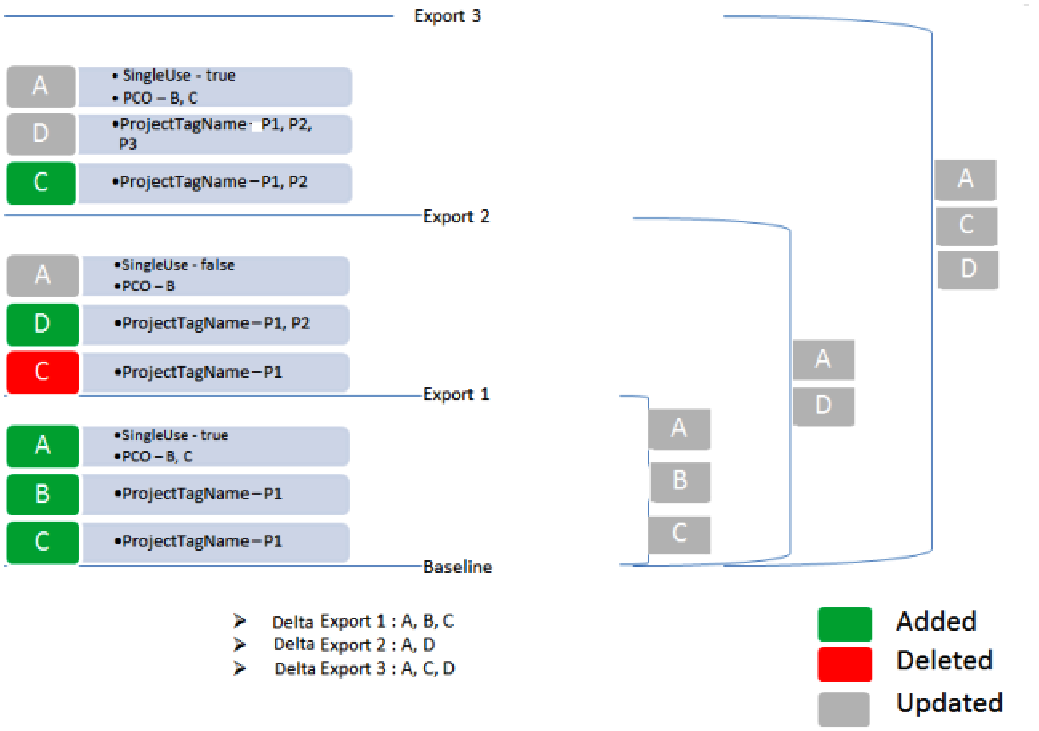

Step 0 – This use case example starts w/ an empty Fulfillment Catalog Development & test instances
Development instance
Step 1 – Project abc creates and relates ProductA & CharacteristicX and adds project tag to them. Project abc is exported and imported in Test instance.
Development instance
[proj=abc]
ProductA
ProductHasCharacteristic
[proj=abc]
[proj=abc]
CharacteristicX
Partial Export (proj=abc)
Exported CSV files proj=abc
ProductA
Product
CharacteristicX
Characteristic
ProductA->CharacteristicX
ProductHasCharacteristic
Step 2 – Project xyz creates CharacteristicY and relates it to productA. It adds project tag to ProductA, CharacteristicY and relationship. Project xyz is exported and imported in Test instance.
Development instance
[proj=abc,xyz]
ProductA
ProductHasCharacteristic
[proj=abc]
[proj=abc]
CharacteristicX
[proj=xyz]
CharacteristicY
[proj=xyz]
Partial Export (proj=xyz)
Exported CSV files proj=xyz
ProductA
Product
CharacteristicY
Characteristic
ProductA->CharacteristicY
ProductHasCharacteristic


Step 3 – Project xyz is cancelled and removes productA and CharacteristicX project tags as well as relationship. Project xyz is exported
Development instance
[proj=abc]
ProductA
[proj=abc]
CharacteristicX
[proj=abc]
ProductB
ProductHasCharacteristic
[proj=abc]
Partial Export (proj=xyz)
Exported CSV files proj=xyz
Product
Characteristic
ProductHasCharacteristic

Enterprise Delta Export of Fulfillment Catalog Data

You can use Delta Export to export records based on the last modified date of the record. The named version acts as the delta export point and provides information about the date and time of delta export.

Delta Export Flow



If you want to use the delta export, you should select the named version. If it is the first delta export, no named version will be available and all the delta export data, in the enterprise, will be exported. After each successful execution of delta export, a named version will be created for future delta export.

- Currently, the Delta Export feature exports only the right set of data and its relationships (executed using record add, modified using the user interface, or using the import feature) if the changes are performed using a forward relationship.
- If the data is updated or imported using the reverse relationship, you will be able to export the data and relationships, but you will not be able to import the same relationships for the data.
- It is recommended to to update the records or its relationship data using the forward relationship if you want to reuse or import the delta export data. If you do not wish to import delta export data then there will be no impact.

Delta Export creates a named version after each successful execution. This results in a big list of available named version for execution of delta export.

A new utility named `ClearNamedVersionData` is added to address the issue. The `ClearNamedVersionData` utility is present in `$AC_HOME/bin` folder and it clears the named versions from the enterprise.

The `ClearNamedVersionData` utility requests user input like enterprise name, database type, database user, database password, and database instance name.

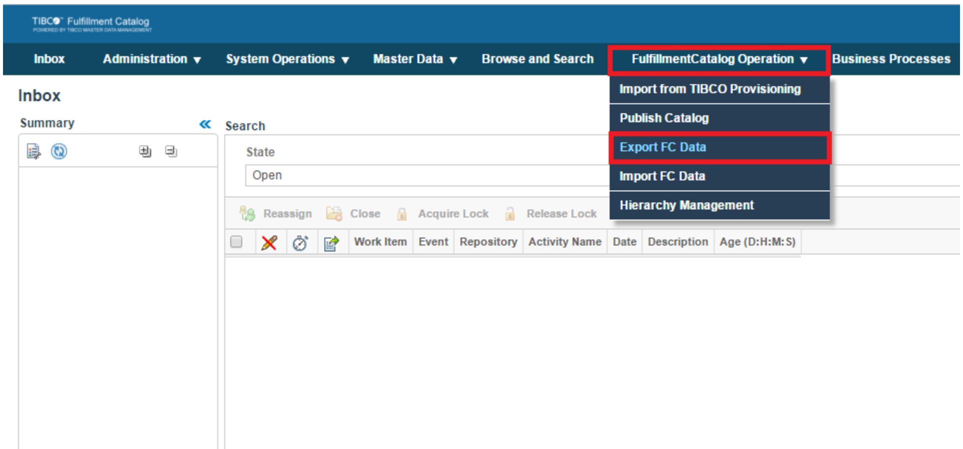
Accessing User Interface to Perform Enterprise Delta Export

To execute Enterprise Delta Export using Fulfillment Catalog user interface, perform the following steps:

Procedure

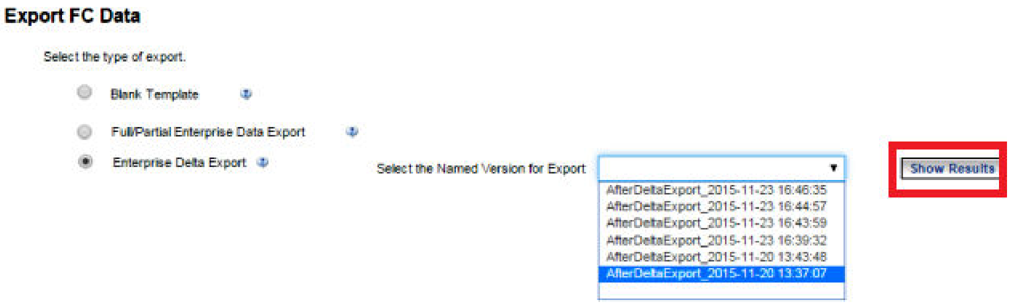
1. Click **FulfillmentCatalog Operation** > **Export FC Data**.

   Accessing Export FC Data

2. Select **Enterprise Delta Export**.
3. Select an appropriate value for the field **Select the Named Version for Export** using the drop-down box.

   Enterprise Delta Export Option Selection

   
   Export FC Data

   Select the type of export.

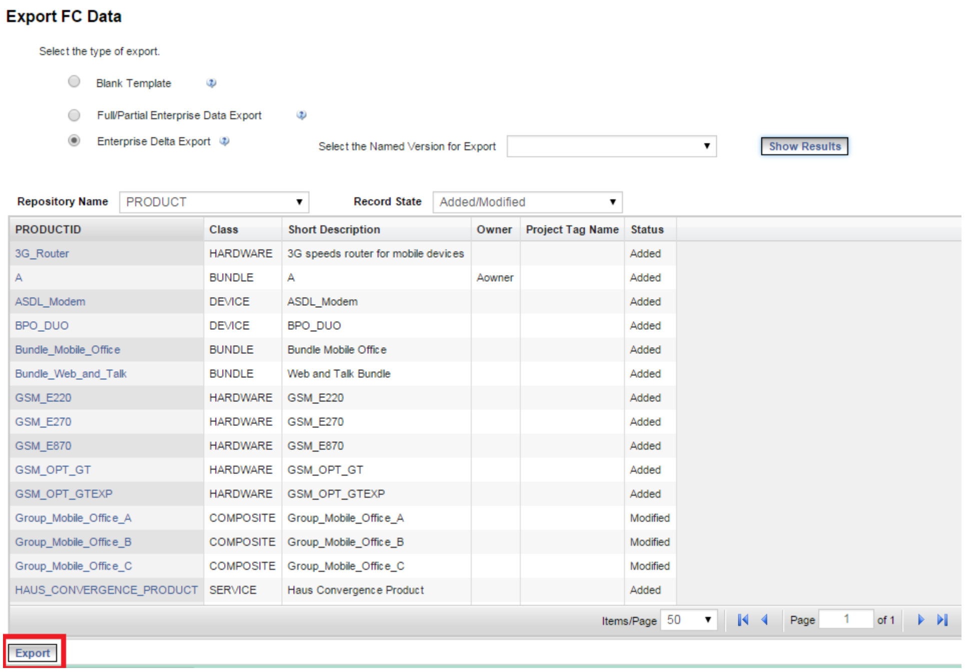
   ○ Blank Template
   ○ Full/Partial Enterprise Data Export
   ● Enterprise Delta Export

   Select the Named Version for Export

   AfterDeltaExport_2015-11-23 16:46:35
   AfterDeltaExport_2015-11-23 16:44:57
   AfterDeltaExport_2015-11-23 16:43:59
   AfterDeltaExport_2015-11-23 16:39:32
   AfterDeltaExport_2015-11-20 13:43:48
   AfterDeltaExport_2015-11-20 13:37:07

4. Click the **Show Results** button to view the data to be exported.

Enterprise Delta Export - Show Results



Export FC Data

Select the type of export.

○ Blank Template

○ Full/Partial Enterprise Data Export

● Enterprise Delta Export

Select the Named Version for Export

Repository Name: PRODUCT

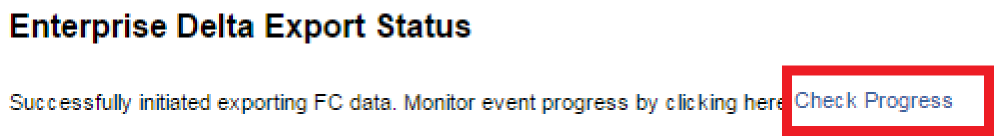
Record State: Added/Modified

PRODUCTID	Class	Short Description	Owner	Project Tag Name	Status
3G_Router	HARDWARE	3G speeds router for mobile devices			Added
A	BUNDLE	A	Aowner		Added
ASDL_Modem	DEVICE	ASDL_Modem			Added
BPO_DUO	DEVICE	BPO_DUO			Added
Bundle_Mobile_Office	BUNDLE	Bundle Mobile Office			Added
Bundle_Web_and_Talk	BUNDLE	Web and Talk Bundle			Added
GSM_E220	HARDWARE	GSM_E220			Added
GSM_E270	HARDWARE	GSM_E270			Added
GSM_E870	HARDWARE	GSM_E870			Added
GSM_OPT_GT	HARDWARE	GSM_OPT_GT			Added
GSM_OPT_GTEXP	HARDWARE	GSM_OPT_GTEXP			Added
Group_Mobile_Office_A	COMPOSITE	Group_Mobile_Office_A			Modified
Group_Mobile_Office_B	COMPOSITE	Group_Mobile_Office_B			Modified
Group_Mobile_Office_C	COMPOSITE	Group_Mobile_Office_C			Modified
HAUS_CONVERGENCE_PRODUCT	SERVICE	Haus Convergence Product			Added

Items/Page 50 Page 1 of 1

5. Click the **Export** button.
   The **Enterprise Delta Export Status** page opens. The enterprise data will be exported.
6. Click **Check Progress** to check the progress of the export.

   Enterprise Delta Export Status



Enterprise Delta Export Status

Successfully initiated exporting FC data. Monitor event progress by clicking here Check Progress

Customization Workflow for Export

The following customization can be performed on workflow for the export feature:

- Customizing Workflow for a New Repository for Full Export and Partial Export
- Customizing Workflow for a New Repository for Delta Export
- Customizing Workflow for a New Relationship
- Customizing Workflow for a New Attribute

Customizing Workflow for a New Repository for Full Export and Partial Export

To customize the workflow for a new repository for full and partial export, perform the following steps:

Procedure

1. Create a new repository. For example, NEWREPO. The prerequisite for NEWREPO is that it has to created in the Fulfillment Catalog Metadata Studio Project and should be successfully deployed into the Fulfillment Catalog enterprise.
2. Locate the `$MQ_COMMON_DIR/<ENTERPRISE_NAME>/workflow/` directory, edit the `wfin26enterprisedataexportv1.xml` file.

3. Succeeding the `SpawnSubWorkFlow_PROJECTTAG`, add another activity for the new repository as shown in the following example:

```
<Activity Name="SpawnSubWorkFlow_NEWREPO">
<Action>InitiateSubFlow</Action>
<Description>Spawn the subworkflow for NEWREPO repository</Description>
<Execution>SYNCHR</Execution>
<Parameter direction="in" type="string" eval="constant"
name="eventState">SPAWNWORKFLOW</Parameter>
<Parameter direction="in" type="string" eval="constant"
name="ProcessID">standard/workflow/wfin26enterprisedataexportsubflowv1</
Parameter>
<Parameter direction="in" type="document" eval="variable"
name="InDocument">inDoc</Parameter>
<Parameter name="RepositoryName" direction="in" eval="constant"
type="string">NEWREPO</Parameter>
<Parameter direction="in" eval="variable" type="string"
name="ProjectTagNames">ProjectTags</Parameter>
<Parameter direction="in" eval="variable" type="string"
name="FolderPath">TempFilePath</Parameter>
<Parameter name="OutDocument" direction="out" eval="variable"
type="document">inDoc</Parameter>
</Activity>
```

4. Comment out the `Script` tag by adding "<!--" at the beginning of the script and adding "-->" at the end of the script. Example for the script to be commented out is as follows:

```
<Transition FromActivity="SpawnSubWorkFlow_PROJECTTAG"
ToActivity="PostProcessExport"/>
```

5. Add activity transitions as shown in the following example:

```
<Transition FromActivity="SpawnSubWorkFlow_PROJECTTAG"
ToActivity="SpawnSubWorkFlow_NEWREPO"/>
<Transition FromActivity="SpawnSubWorkFlow_NEWREPO" ToActivity="
PostProcessExport "/>
```

Customizing Workflow for a New Repository for Delta Export

To customize the workflow for a new repository for delta export, perform the following steps:

Procedure

1. Create a new repository. For example, NEWREPO. The prerequisite for NEWREPO is that it has to created in the Fulfillment Catalog Metadata Studio Project and should be successfully deployed into the Fulfillment Catalog enterprise.
2. Locate the `$MQ_COMMON_DIR/<ENTERPRISE_NAME>/workflow/` directory, edit the `wfin26enterprisedataexport_increamentalv1.xml` file.
3. Succeeding the `SpawnSubWorkFlow_PROJECTTAG`, add another activity for the new repository as shown in the following example:

```
<Activity Name="SpawnSubWorkFlow_NEWREPO">
    <Action>InitiateSubFlow</Action>
    <Description>Spawn the subworkflow for NEWREPO repository</Description>
    <Execution>SYNCHR</Execution>
    <Parameter direction="in" type="string" eval="constant"
name="eventState">SPAWNWORKFLOW</Parameter>
    <Parameter direction="in" type="string" eval="constant"
name="ProcessID">standard/workflow/
wfin26enterprisedataexportsubflow_increamentalv1</Parameter>
    <Parameter direction="in" type="document" eval="variable"
name="InDocument">inDoc</Parameter>
    <Parameter name="RepositoryName" direction="in" eval="constant"
type="string">NEWREPO</Parameter>
    <Parameter name="SelectedNV" direction="in" eval="variable"
type="string">SelectedNamedVersionName</Parameter>
    <Parameter name="SelNVTimeStamp" direction="in" eval="variable"
type="string">fromTimeStamp</Parameter>
    <Parameter direction="in" eval="variable" type="string"
```

```
name="NamedVersionName">NamedVersionNameToCreate</Parameter>
    <Parameter name="NVTimeStamp" direction="in" eval="variable"
type="string">toTimeStamp</Parameter>
    <Parameter direction="in" eval="variable" type="string"
name="FolderPath">TempFilePath</Parameter>
    <Parameter name="OutDocument" direction="out" eval="variable"
type="document">inDoc</Parameter>
 </Activity>
```

4. Comment out the `Script` tag by adding "<!--" at the beginning of the script and adding "-->" at the end of the script. Example for the script to be commented out is as follows:

   ```
   <Transition FromActivity="SpawnSubWorkFlow_PROJECTTAG"
   ToActivity="PostProcessExport"/>
   ```

5. Add activity transitions as shown in the following example:

   ```
   <Transition FromActivity="SpawnSubWorkFlow_PROJECTTAG"
   ToActivity="SpawnSubWorkFlow_NEWREPO"/>
   <Transition FromActivity="SpawnSubWorkFlow_NEWREPO" ToActivity="
   PostProcessExport "/>
   ```

Customizing Workflow for a New Relationship

For a new relationship, no additional configuration is required.

New relationship must be created in the Fulfillment Catalog Metadata Studio Project and should be deployed into the Fulfillment Catalog enterprise successfully.

The new relationship would be exported to a CSV file with the name `RelationshipAttribute_<REL_NAME>.csv`.

Customizing Workflow for a New Attribute

For a new attribute (Repository or relationship), no additional configuration is required.

Newly added attribute must be created in Fulfillment Catalog Metadata Studio Project and successfully deployed into the Fulfillment Catalog instance.

The new attribute would be exported to CSV file of the corresponding repository or relationship.

Import of Fulfillment Catalog

Importing data into Fulfillment Catalog deals with:

- Types of Import of Fulfillment Catalog Data, and
- Customization Workflow for Import

Types of Import of Fulfillment Catalog Data

The following import types, of Fulfillment Catalog data, can be performed using the Fulfillment Catalog user interface:

- Full Import of Fulfillment Catalog Data
- Partial Import of Fulfillment Catalog Data

Full Import of Fulfillment Catalog Data

Full Import allows you to import data into the enterprise. You can import a `ZIP` file (`*.zip`) containing `CSV` files of data. To ensure that a valid `ZIP` file is selected for import, content files are checked for valid repository and relationship names. If unacceptable files are found, the process terminates with an error. Data will be added or updated in repositories and relationships based on the data availability in the imported `ZIP` file and data existing in the enterprise.

- Full Import works well with the out-of-the-box metadata provided by the TIBCO Fulfillment Catalog product. Any customization of the metadata, like adding attributes to the repository or to the relationship levels, will have an impact on the feature, resulting in incorrect data. It may also lead to the importing of incorrect data, or it might provide undesired results. If you are facing such a problem, contact TIBCO support for additional help.
- The `CSV` files, which will be imported, containing the Fulfillment Catalog data must conform to the structure required by the metadata. You can download a sample of the `CSV` file, for a repository or a relationship, from the respective data source.

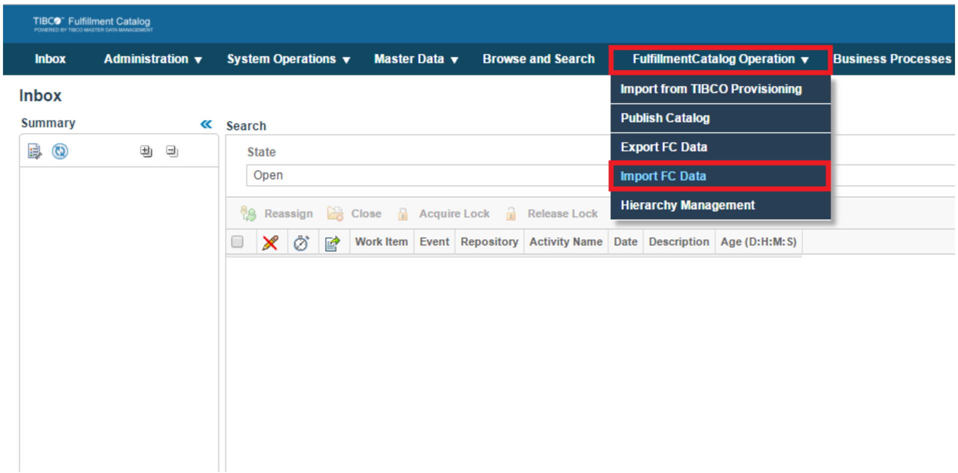
Accessing the User Interface to Fully Import Fulfillment Catalog Data

To execute Full Import of Fulfillment Catalog data using the Fulfillment Catalog user interface, perform the following steps:

Procedure

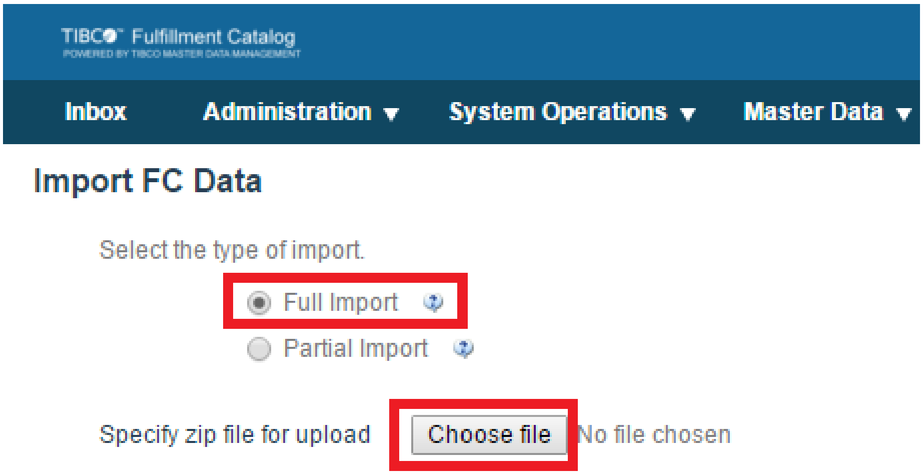
1. Click **FulfillmentCatalog Operation** > **Import FC Data**.

   Accessing Import FC Data



2. Select **Full Import**.
3. Click **Choose File** button and select the required file.

   Import FC Data Options


TIBCO Fulfillment Catalog
POWERED BY TIBCO MASTER DATA MANAGEMENT
Inbox
Administration
System Operations
Master Data
Import FC Data

Select the type of import.

● Full Import

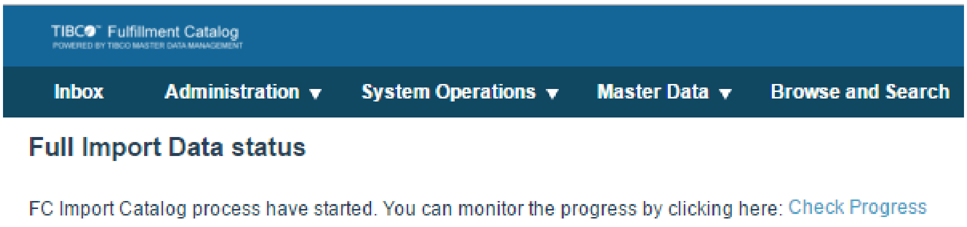
○ Partial Import

Specify zip file for upload Choose file No file chosen

Only a ZIP file can be selected for import.

4. When upload is clicked, business logic is invoked to import data. User is directed to a status page if the process has been initiated successfully. The status page contains the link to the event log where status of the process can be monitored.

   Full Import FC Data Status



Full Import Data status

FC Import Catalog process have started. You can monitor the progress by clicking here: Check Progress

Backward Compatibility

The Import FC Data feature is backward compatible with Fulfillment Catalog 3.0.1 because Fulfillment Catalog 3.0.1 data CSVs can still be imported without any messaging.

However, the same feature is not compatible with Fulfillment Catalog 3.0.0 data CSVs because of two new attributes, OfferID and IsTemplate, introduced to the `PRODUCT` repository. Importing the Fulfillment Catalog 3.0.0 compatible `PRODUCT_DATA.csv` (or any other csv name which is used for importing record in `PRODUCT` repository) would not be supported. You have to download the blank template from Fulfillment Catalog 3.0.2, fill the data, and then perform the import operation.

Partial Import of Fulfillment Catalog Data

This allows to import data into the enterprise where from the same has been initiated. It accepts a `ZIP` file (*.`ZIP`) only where the content should be `CSV` files containing the actual data. To ensure that a valid `ZIP` file is selected for import, the content files are checked for valid names for repository and relationships. If there is any unaccepted file found, the process terminates with the error. The data would be added/updated/deleted into repositories and relationships based on the exported tag(s), data in the imported `ZIP` file and already data availability in the enterprise.

- The feature works well with the out-of-the-box metadata provided by the TIBCO Fulfillment Catalog product. Any customization of the metadata, like adding attributes to the repository or to the relationship levels, will have an impact on the feature, resulting in incorrect data. It may also lead to the importing of incorrect data, or it might provide undesired results. If you are facing such a problem, contact TIBCO support for additional help.
- The `CSV` files, which will be imported, containing the Fulfillment Catalog data must conform to the structure required by the metadata. You can download a sample of the `CSV` file, for a repository or a relationship, from the respective data source.

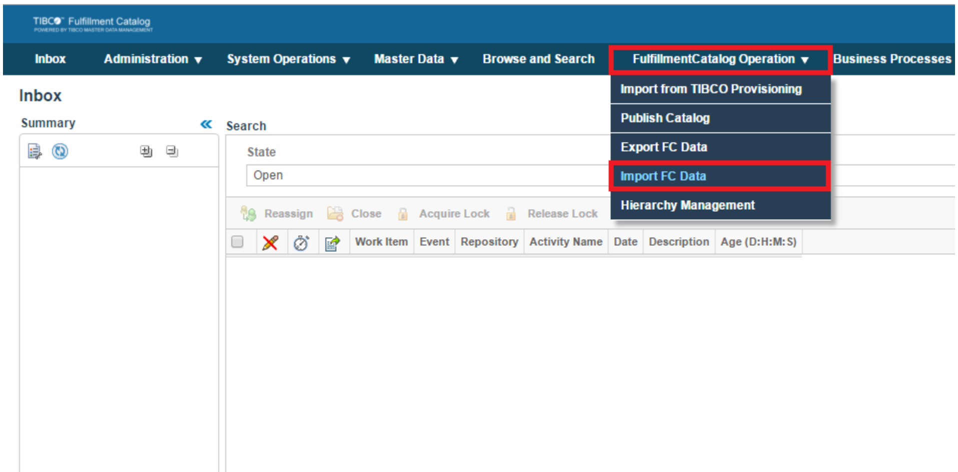
Accessing the User Interface to Partially Import Fulfillment Catalog Data

To execute the Partial Import of Fulfillment Catalog data using the Fulfillment Catalog user interface, perform the following steps:

Procedure

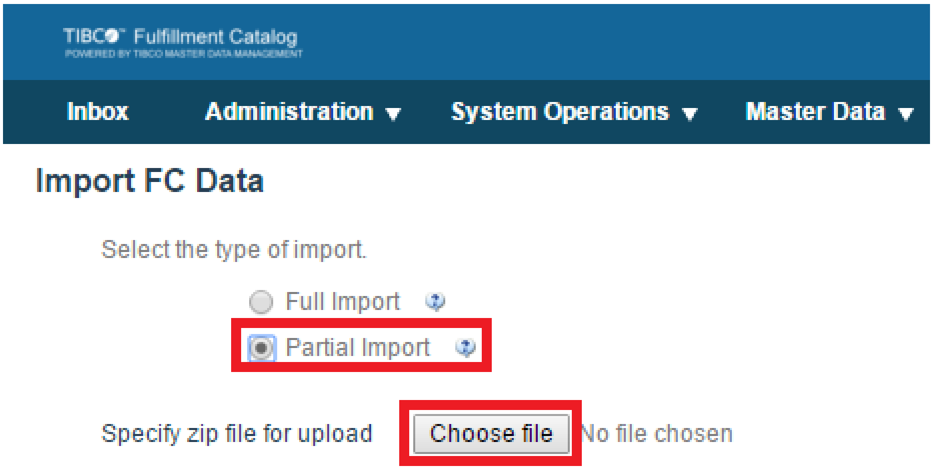
1. Click **FulfillmentCatalog Operation** > **Import FC Data**. The **Import FC Data** page opens.

Accessing Import FC Data



2. Select the type of import as **Partial Import**.
3. Click **Choose file** button to select the appropriate ZIP file.
4. Click **Upload** button to complete the import process.

   Import FC Data Page - Partial Upload Selected



Only a ZIP file can be selected for import.

When upload is clicked, business logic is invoked to import data. User is directed to a status page if the process has been initiated successfully. The status page contains the link to the event log where status of the process can be monitored.

Use Cases for Partial Import of Fulfillment Catalog Data

The topic covers use cases that are relevant to the Partial Import feature of Fulfillment Catalog.

Each use case is applicable for the Repository records as well as the Relationship records.

The list of use cases are as follows:

1. Partial Import of Fulfillment Catalog Data - Use Case 1
2. Partial Import of Fulfillment Catalog Data - Use Case 2
3. Partial Import of Fulfillment Catalog Data - Use Case 3

4. Partial Import of Fulfillment Catalog Data - Use Case 4
5. Partial Import of Fulfillment Catalog Data - Use Case 5
6. Partial Import of Fulfillment Catalog Data - Use Case 6
7. Partial Import of Fulfillment Catalog Data - Use Case 7
8. Partial Import of Fulfillment Catalog Data - Use Case 8
9. Partial Import of Fulfillment Catalog Data - Use Case 9
10. Partial Import of Fulfillment Catalog Data - Use Case 10

Partial Import of Fulfillment Catalog Data - Use Case 1

The following is the first use case for the Partial Import feature of Fulfillment Catalog:

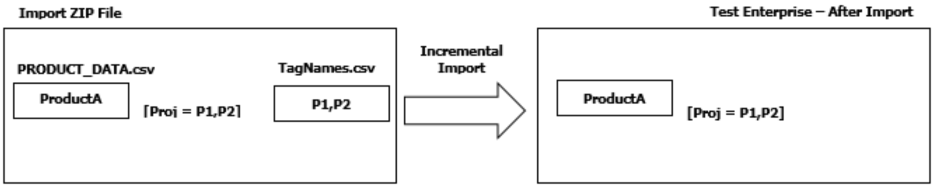
Step 1

The import zip file has repository records (for example, PRODUCT repository records) named ProductA with the project tag [P1,P2]. The `TagNames.csv` file, which exists within the import zip file, has [P1,P2]. **Import the zip file to a Clean or New Enterprise**.

Because [P1,P2] exists in the `TagNames.csv` file, only the records that are tagged with [P1,P2] are added or modified to the Test enterprise.

Because ProductA does not exist in the enterprise, but exists in the `PRODUCT_DATA.csv` file with the project tag [P1,P2], it is added to the Test enterprise with the project tag [P1,P2].

Partial Import of Fulfillment Catalog Data - Use Case 1 Step 1



Step 2

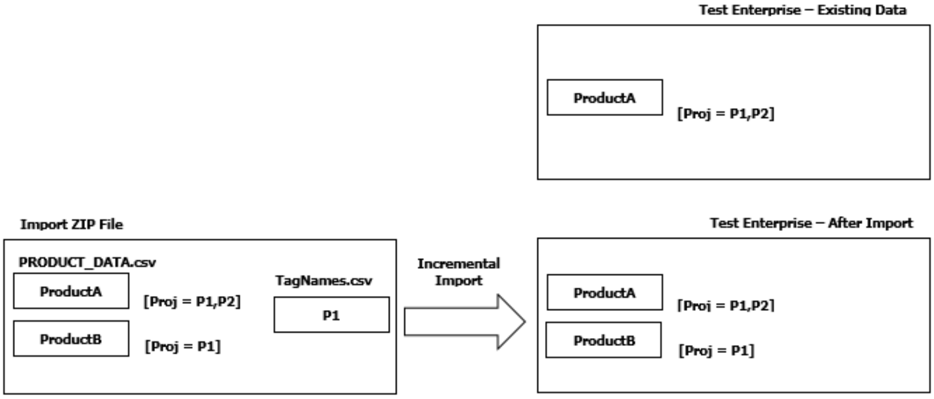
The import zip file has repository records (for example, PRODUCT repository records) named ProductA and ProductB with project tags as [P1,P2] and [P1] respectively. The `TagNames.csv` file, which exists within the import zip file, has [P1]. **Import the zip file to an enterprise on which Step 1 is performed**.

Because [P1] is present in the `TagNames.csv` file, only the records that are tagged with [P1] are added or modified to the Test enterprise.

Because ProductB does not exist in the enterprise, but exists in the `PRODUCT_DATA.csv` file, ProductB, with the project tag [P1], is loaded to the Test enterprise.

Because ProductA exists in the enterprise with the project tag [P1], and exists in the `PRODUCT_DATA.csv` file with the project tag [P1], it is loaded to the enterprise with no modifications to the project tag.

Partial Import of Fulfillment Catalog Data - Use Case 1 Step 2



Partial Import of Fulfillment Catalog Data - Use Case 2

The following is the second use case for the Partial Import feature of Fulfillment Catalog:

Step 1

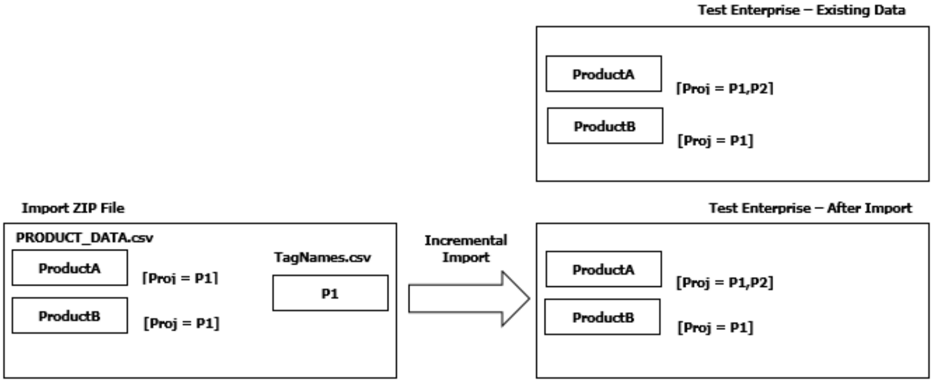
The Import zip file has repository records (for example, PRODUCT repository records) named ProductA and ProductB with the project tags [P1] and [P1] respectively. The `TagNames.csv` file, which exists within the import zip file, has [P1]. **Import the zip file to an existing enterprise that has data in it**.

Because [P1] exists in `TagNames.csv` file, only the records with the project tag [P1] are added or modified to the Test enterprise.

Because ProductA exists in the `PRODUCT_DATA.csv` file with the project tag [P1], and ProductA also exists in the enterprise with the project tag [P1], it is loaded to the enterprise without any change to the project tag.

Because ProductB exists in the `PRODUCT_DATA.csv` file with the project tag [P1], and ProductB also exists in the enterprise with the project tag [P1], it is loaded to the enterprise without any change to the project tag.

Partial Import of Fulfillment Catalog Data - Use Case 2 Step 1



Partial Import of Fulfillment Catalog Data - Use Case 3

The following is the third use case for the Partial Import feature of Fulfillment Catalog:

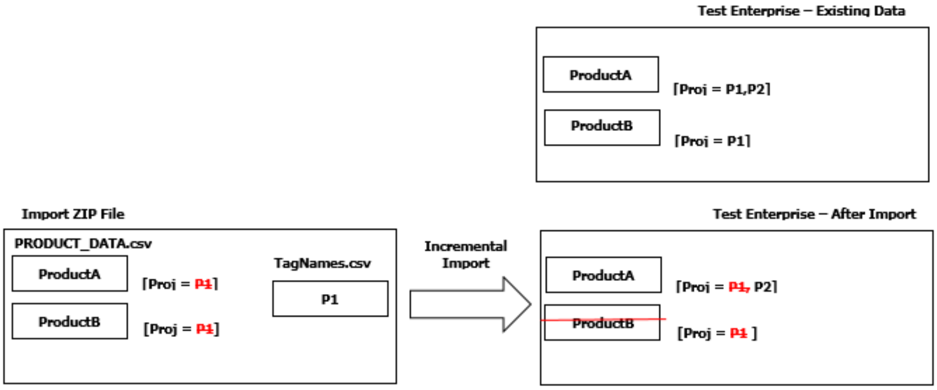
The Import zip file has repository records (for example, PRODUCT repository records) named ProductA and ProductB without the project tags. The `TagNames.csv` file, which exists within the import zip file, has [P1]. **Import the zip file to an existing enterprise that has data in it**.

Because [P1] exists in the `TagNames.csv` file, only the records that are tagged with [P1] are added or modified to the Test enterprise.

Because ProductA exists in the `PRODUCT_DATA.csv` file without the project tag, and ProductA also exists in the enterprise with the project tag [P1,P2], it is loaded to the enterprise with the project tag [P2]. The [P1] tag is removed for ProductA because it is not present in the `PRODUCT_DATA.csv` file.

Because ProductB exists in the `PRODUCT_DATA.csv` file without the project tag, and ProductB also exists in the enterprise with the project tag [P1], it is updated to the enterprise without the project tag. Because PorductB has no project tag it is removed from the enterprise.

Partial Import of Fulfillment Catalog Data - Use Case 3 Step 1



Partial Import of Fulfillment Catalog Data - Use Case 4

The following is the fourth use case for the Partial Import feature of Fulfillment Catalog:

Step 1

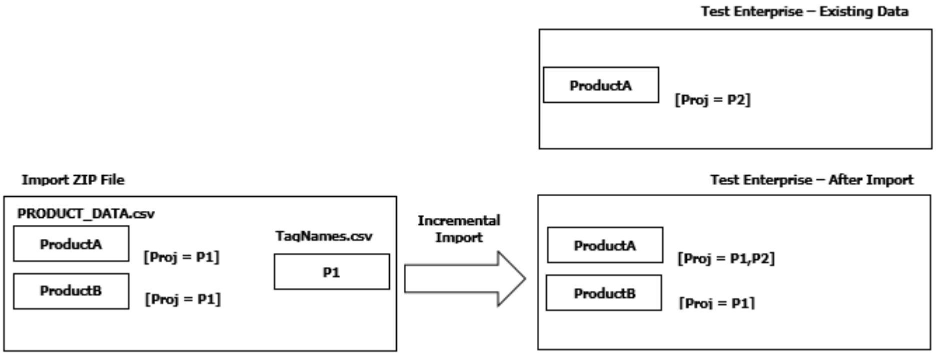
The Import zip file has repository records (for example, PRODUCT repository records) named ProductA and ProductB with project tags [P1]and [P1] respectively. The `TagNames.csv` file, which exists within the import zip file, has [P1]. **Import the zip file to an existing enterprise that has data present in it**.

Because [P1] exists in the `TagNames.csv` file, only the records with the project tag [P1] are added or modified to the Test enterprise.

Because ProductA exists in the `PRODUCT_DATA.csv` file with the project tag [P1], and ProductA also exists in the enterprise with the project tag [P2], it is loaded into the enterprise with the project tag [P1,P2].

Because ProductB exists in the `PRODUCT_DATA.csv` file with the project tag [P1], and ProductB does not exist in the enterprise, it is added to the Test enterprise with the project tag [P1].

Partial Import of Fulfillment Catalog Data - Use Case 4 Step 1

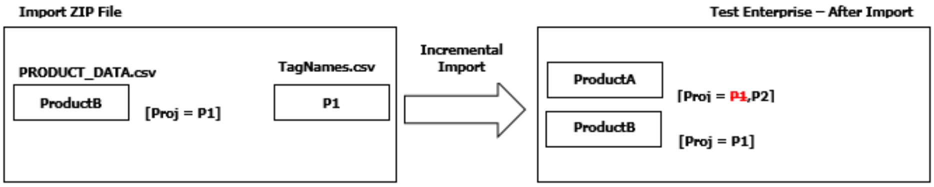


Step 2

The Import zip file has repository records (for example, PRODUCT repository records) named ProductB with the project tag [P1]. The `TagNames.csv` file, which exists within the import zip file, has [P1]. **Import the zip file to an existing enterprise on which Step 1 is performed**.

Because ProductA does not exist in the `PRODUCT_DATA.csv` file, and ProductA exists in the enterprise with the project tag [P1,P2], it is updated with the project tag [P2] and the project tag [P1] is be removed. If the ProductA existed in the enterprise, its project tag would have been modified or removed. The ProductB is loaded to the Test enterprise with the project tag [P1].

Partial Import of Fulfillment Catalog Data - Use Case 4 Step 2



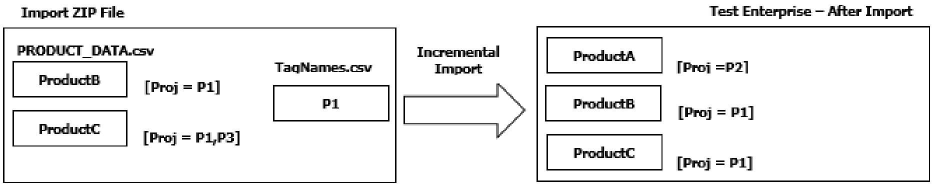
Step 3

The Import zip file has repository records (for example, PRODUCT repository records) named ProductB and Product C with project tags [P1], [P1,P3]. The `TagNames.csv` file, which exists within the import zip file, has [P1]. **Import the zip file to an existing enterprise on which Step 2 is performed**.

Because ProductB exists in the `PRODUCT_DATA.csv` file with the project tag [P1], and ProductB also exists in the enterprise, it is loaded to the Test enterprise with the project tag [P1].

Because ProductC exists in the `PRODUCT_DATA.csv` file with the project tag [P1], and ProductC does not exist in the enterprise, it is loaded to the Test enterprise with the project tag [P1].

Partial Import of Fulfillment Catalog Data - Use Case 4 Step 3



Partial Import of Fulfillment Catalog Data - Use Case 5

The following is the fifth use case for the Partial Import feature of Fulfillment Catalog:

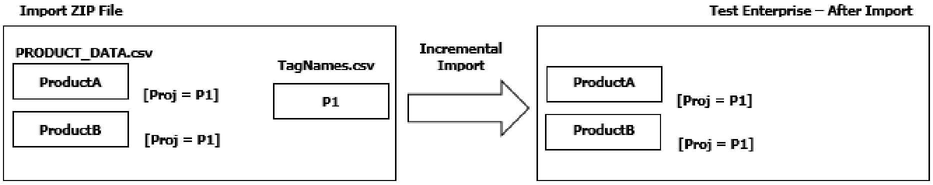
Step 1

The Import zip file has repository records (for example, PRODUCT repository records) named ProductA and ProductB with the project tags [P1] and [P1]. The `TagNames.csv` file, which exists within the import zip file, has [P1]. **Import the zip file to a Clean or New enterprise**.

Because [P1] exists in the `TagNames.csv` file, only the records that are tagged with [P1] are added or modified to the Test enterprise.

Because the import is performed on a Clean or New enterprise, ProductA and ProductB are loaded to the Test enterprise with the project tag [P1] and [P1] respectively.

Partial Import of Fulfillment Catalog Data - Use Case 5 Step 1

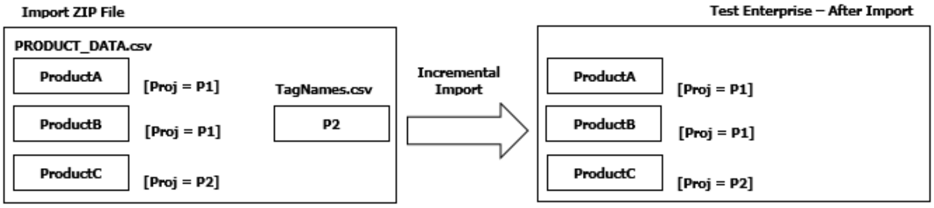


Step 2

The Import zip file has repository records (example PRODUCT repository records) named ProductA, ProductB, and ProductC with the project tags [P1], [P1], and [P2] respectively. The `TagNames.csv` file, which exists within the import zip file, has [P2]. **Import the zip file to an enterprise on which Step 1 is performed**.

Because [P2] exists in the `TagNames.csv` file, only the records that have the project tag [P2] are added or modified to the Test enterprise. ProductC with the project tag [P2] is loaded to the Test enterprise.

Partial Import of Fulfillment Catalog Data - Use Case 5 Step 2



Partial Import of Fulfillment Catalog Data - Use Case 6

The following is the sixth use case for the Partial Import feature of Fulfillment Catalog:

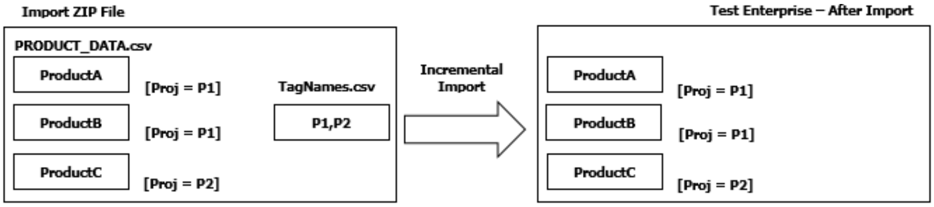
Step 1

The Import zip file has repository records (for example, PRODUCT repository records) named ProductA, ProductB, and ProductC with the project tags [P1], [P1], and [P2] respectively. The `TagNames.csv` file, which exists within the import zip file, has [P1,P2]. **Import the zip file to a Clean or New enterprise**.

Because [P1,P2] exists in the `TagNames.csv` file, only the records that have the project tag [P1] or [P2] are added or modified to the Test enterprise.

Because the import is performed on a Clean or New enterprise, ProductA, ProductB, and ProductC are loaded to the Test enterprise with the project tags [P1], [P1], and [P2] respectively.

Partial Import of Fulfillment Catalog Data - Use Case 6 Step 1



Step 2

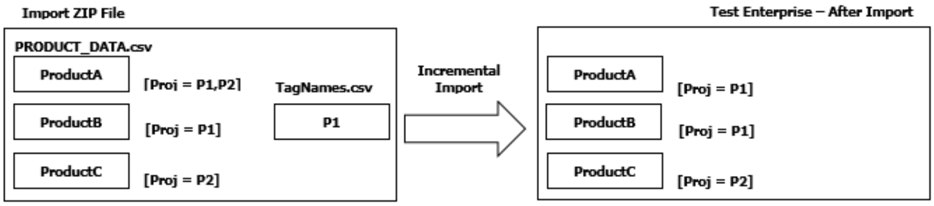
The Import zip file has repository records (for example, PRODUCT repository records) named ProductA, ProductB, and ProductC with the project taga [P1,P2], [P1], and [P2] respectively. The `TagNames.csv` file, which exists within the import zip file, has [P1]. **Import the zip file into an enterprise on which Step 1 is performed**.

Because [P1] exists in the `TagNames.csv` file, only the records that have the project tage [P1] are added or modified to the Test enterprise.

Because ProductA and ProductB exists in the `PRODUCT_DATA.csv` file with the project tag [P1,P2] and [P1] respectively, and ProductA and ProductB also exists in the enterprise with the project tag [P1], they are loaded to the enterprise without any changes to the project tags.

Because ProductC has the project tag [P2], and ProductC also exists in the enterprise with the project tag [P2], it is not loaded to the enterprise as the project tag did not match the exported project tag [P1].

Partial Import of Fulfillment Catalog Data - Use Case 6 Step 2



Partial Import of Fulfillment Catalog Data - Use Case 7

The following is the seventh use case for the Partial Import feature of Fulfillment Catalog:

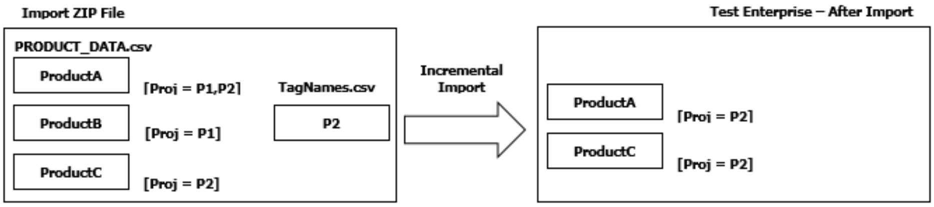
Step 1

The Import zip file has repository records (for example, PRODUCT repository records) named ProductA, ProductB, and ProductC with the project tags [P1,P2], [P1], and [P2] respectively. The `TagNames.csv` file, which exists within the import zip file, has [P2]. **Import the zip file to a Clean or New enterprise**.

Because [P2] exists in the `TagNames.csv` file, all the records with the project tag [P2] are added or modified to the Test enterprise.

Because ProductA has the project tag [P2], and ProductC has the project tag [P2], only ProductA and ProductC are added or loaded to the Test enterprise with the project tag [P2].

Partial Import of Fulfillment Catalog Data - Use Case 7 Step 1



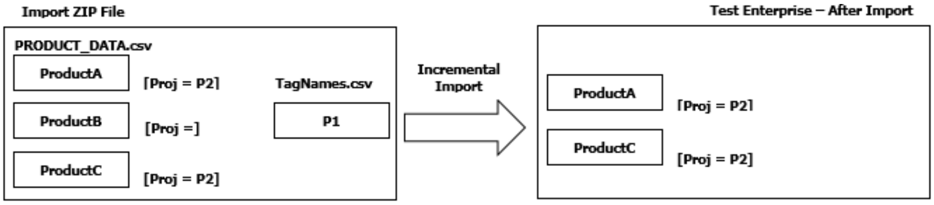
Step 2

The Import zip file has repository records (for example, PRODUCT repository records) named ProductA, ProductB, and ProductC with the project tags [P2], [], and [P2] respectively. The `TagNames.csv` file, which exists within the import zip file, has [P1]. **Import the zip file to an enterprise on which Step 1 is performed**.

Because [P1] exists in the `TagNames.csv` file, only the records with the project tag [P1] are added or modified to the Test enterprise.

Because there are no records in the import file with the project tag [P1], and there are no records in the enterprise with the project tag [P1], no data is added or modified to the Test enterprise.

Partial Import of Fulfillment Catalog Data - Use Case 7 Step 2



Partial Import of Fulfillment Catalog Data - Use Case 8

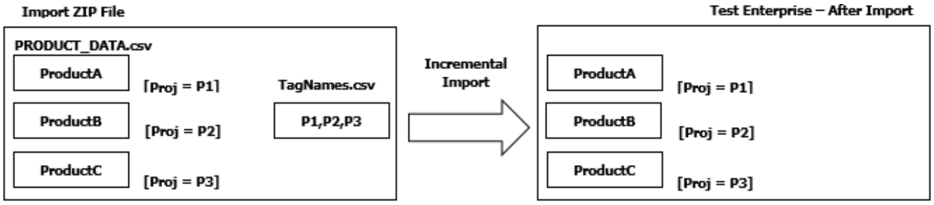
The following is the eighth use case for the Partial Import feature of Fulfillment Catalog:

Step 1

The Import zip file has repository records (for example, PRODUCT repository records) named ProductA, ProductB, and ProductC with the project tags [P1], [P2], and [P3] respectively. The `TagNames.csv` file, which exists within the import zip file, has [P1,P2,P3]. **Import the zip file to a Clean or New enterprise**.

Because [P1,P2,P3] exists in the `TagNames.csv` file, only the records that have the project tags [P1], [P2] or [P3] are added or modified to the Test enterprise.

Partial Import of Fulfillment Catalog Data - Use Case 8 Step 1



Step 2

The Import zip file has repository records (for example, PRODUCT repository records) named ProductA, ProductB, and ProductC with the project tags [P1,P2], [P2,P3], and [P3,P1] respectively. The `TagNames.csv` file, which exists within the import zip file, has [P1,P2,P3]. **Import the zip file to an enterprise on which Step 1 is performed**.

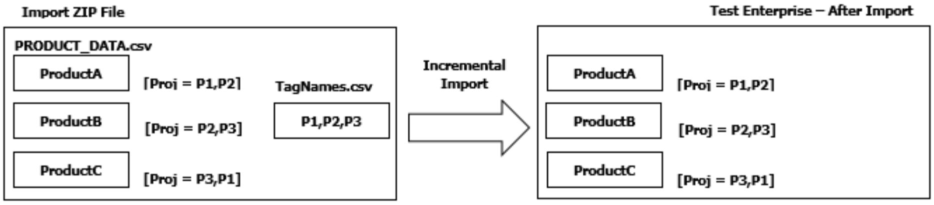
Because [P1], [P2], and [P3] exists in the `TagNames.csv` file, only the records with the project tags [P1], [P2] or [P3] are added or modified to the Test enterprise.

Because ProductA in the `PRODUCT_DATA.csv` file has the project tag [P1,P2], and the `TagNames.csv` file has the project tag [P1,P2,P3], it is loaded to the enterprise with the project tag [P1,P2].

Because ProductB in the `PRODUCT_DATA.csv` file has the project tag [P2,P3], and the `TagNames.csv` file has the project tag [P1,P2,P3], it is loaded to the enterprise with the project tag [P2,P3].

Because ProductC in the `PRODUCT_DATA.csv` file has the project tag [P3,P1], and the `TagNames.csv` file has the project tag [P1,P2,P3], it is loaded to the enterprise with the project tag [P3,P1].

Partial Import of Fulfillment Catalog Data - Use Case 8 Step 2



Partial Import of Fulfillment Catalog Data - Use Case 9

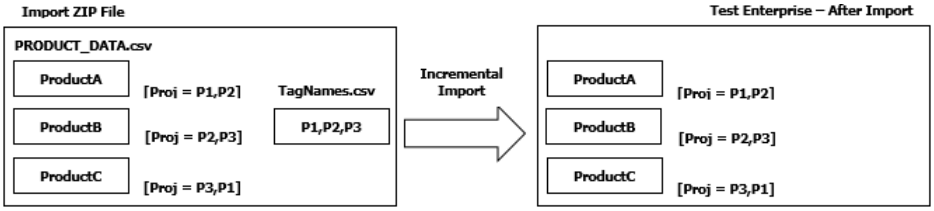
The following is the ninth use case for the Partial Import feature of Fulfillment Catalog:

Step 1

The Import zip file has repository records (for example, PRODUCT repository records) named ProductA, ProductB, and ProductC with the project tags [P1,P2], [P2,P3], and [P3,P1] respectively. The `TagNames.csv` file, which exists within the import zip file, has [P1,P2,P3]. **Import the zip file to a Clean or New enterprise**.

Because [P1,P2,P3] exists in the `TagNames.csv` file, only the records with the project tags [P1], [P2] or [P3] are added or modified to the Test enterprise.

Partial Import of Fulfillment Catalog Data - Use Case 9 Step 1



Step 2

The Import zip file has repository records (for example, PRODUCT repository records) named ProductA, ProductB, and ProductC with the project tags [P4], [P5], and [P6] respectively. The `TagNames.csv` file, which exists within the import zip file, has [P1,P2,P3]. **Import the zip file to an enterprise on which Step 1 has been performed**.

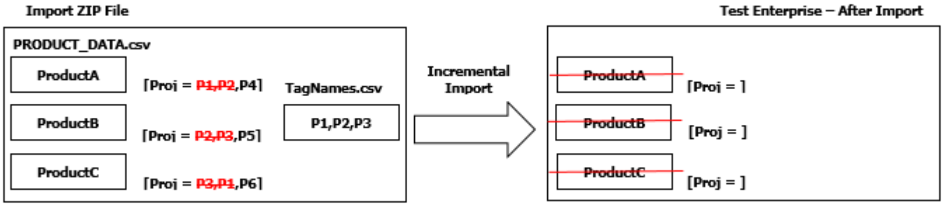
[P4], [P5], and [P6] does not exist in the `TagNames.csv` file and such project tags are not added to the Test enterprise.

Because ProductA exists in the `PRODUCT_DATA.csv` file without the project tag [P1,P2], and ProductA exists in the enterprise with the project tag [P1,P2], the import process removes the project tag [P1,P2] from ProductA. Because there is no project tag for ProductA, it is deleted.

Because ProductB exists in the `PRODUCT_DATA.csv` file without the project tag [P2,P3], and ProductB exists in the enterprise with the project tag [P2,P3], the import process removes the project tag [P2,P3] from ProductB. Because there is no project tag for ProductB, it is deleted.

Because ProductC exists in the `PRODUCT_DATA.csv` file without the project tag [P3,P1], and ProductC exists in the enterprise with the project tag [P3,P1], the import process removes the project tag [P3,P1] from ProductC. Because there is no project tag for ProductC, it is deleted.

Partial Import of Fulfillment Catalog Data - Use Case 9 Step 2



Partial Import of Fulfillment Catalog Data - Use Case 10

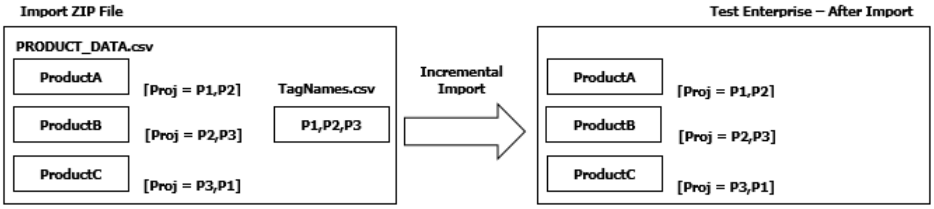
The following is the tenth use case for the Partial Import feature of Fulfillment Catalog:

Step 1

The import zip file has repository records (for example, PRODUCT repository records) named ProductA, ProductB, and ProductC with the project tags [P1,P2], [P2,P3], and [P3,P1] respectively. The `TagNames.csv` file, which exists within the import zip file, has [P1,P2,P3]. **Import the zip file to a Clean or New Enterprise.**

Because [P1], [P2], and [P3] exist in the `TagNames.csv` file, only the records having the project tag [P1], [P2] or [P3] are added or modified to the Test enterprise.

Partial Import of Fulfillment Catalog Data - Use Case 10 Step 1



Step 2

The import zip file has repository records (for example, PRODUCT repository records) named ProductA, ProductB, and ProductC with the project tags [P4], [P5], and [P6] respectively. The `TagNames.csv` file, which exists within the import zip file, has [P1,P6]. **Import the zip file to an enterprise on which Step 1 is performed.**

Because [P1,P6] exists in the `TagNames.csv` file, only the records that have the project tag [P1] or [P6] are added or modified to the Test enterprise.

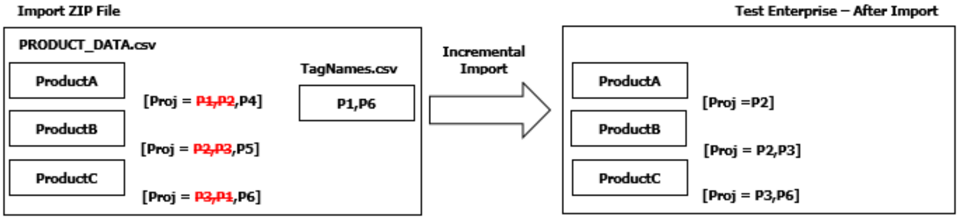
Because [P4,P5] does not exist in the `TagNames.csv` file, the records that have the project tag [P4] or [P5] are not added to the Test enterprise.

Because ProductA exists in the `PRODUCT_DATA.csv` file without the project tag [P1,P2], and ProductA exists in the `TagNames.csv` file with the project tag [P1,P6], ProductA is added to the Test enterprise without the project tag [P1].

Because ProductB exists in the `PRODUCT_DATA.csv` file with the project tag [P5], and the `TagNames.csv` file has [P1,P6], ProductB is not imported into the Test enterprise.

Because ProductC exists in the `PRODUCT_DATA.csv` file without the project tag [P3,P1], and the `TagNames.csv` file has [P1,P6], ProductC is added to the Test enterprise with the project tag [P6] and without the project tag [P1].

Partial Import of Fulfillment Catalog Data - Use Case 10 Step 2



Customization Workflow for Import

The following customization can be performed on the workflow for the import feature:

- Customizing Workflow for a New Repository

- Customizing Workflow for a Newly Added Relationship
- Customizing Workflow for a Newly Added Attribute to Existing Repository
- Customizing Workflow for Newly Added Relationship Attribute to Existing Relationship

Customizing Workflow for a New Repository

To customize the workflow for a new repository, perform the following steps:

- NEWREPO (newly created repository).
- The corresponding data source and input map should be created in the Fulfillment Catalog Metadata Studio Project and successfully deployed into the Fulfillment Catalog enterprise.
- Follow the naming convention of Repository name, its data source name, and the input map name as seen in the `CatalogDataMap.csv` file present in the `$AC_HOME/samples/` directory. Ensure that there are corresponding CSV file as per the created datasource to import.

Procedure

1. Open the `$AC_HOME/samples/ CatalogDataMap.csv` file.
2. After the line `RULEPARAMETER,RULEPARAMETER_DATA_IMAP,RULEPARAMETER_DATA_DS,RULEPARAMETER_DATA.csv` add the new line for every new repository `NEWREPO,<NEWREPO>_DATA_IMAP,<NEWREPO>_DATA_DS,NEWREPO_DATA.csv`.
    - New lines in `CatalogDataMap.csv` should be added as per the heading.
    - Adhere to the `<NEWREPO>_DATA_IMAP` format for the input map when creating it in MDM Studio.
    - Adhere to the `<NEWREPO>_DATA_DS` format for the data source when creating it in MDM Studio.
    - Adhere to the `<NEWREPO>_DATA.csv` file name format for the CSV file to be imported.
3. For just a partial import perform the following steps:
    a) Locate the `$MQ_COMMON_DIR/<ENTERPRISE_NAME>/workflow/` directory and edit the `wfin26partialimportfcdatav1.xml` file.
    b) Succeeding the activity `SpawnSubWorkFlow_PROJECTTAG`, add another activity for the new repository as shown in the following example:

```
<Activity Name="SpawnSubWorkFlow_NEWREPO">
<Action>InitiateSubFlow</Action>
<Description>Spawn the subworkflow for NEWREPO repository</Description>
<Execution>SYNCHR</Execution>
<Parameter direction="in" type="string" eval="constant"
name="eventState">SPAWNWORKFLOW</Parameter>
<Parameter direction="in" type="string" eval="constant"
name="ProcessID">standard/workflow/wfin26partialimportfcdatasubflowv1</
Parameter>
<Parameter direction="in" type="document" eval="variable"
name="InDocument">inDoc</Parameter>
<Parameter name="RepositoryName" direction="in" eval="constant"
type="string">NEWREPO</Parameter>
<Parameter direction="in" eval="variable" type="string"
name="ProjectTagNames">ProjectTagName</Parameter>
<Parameter direction="in" eval="variable" type="string"
name="FolderPath">TempFilePath</Parameter>
<Parameter name="OutDocument" direction="out" eval="variable"
type="document">inDoc</Parameter>
</Activity>
```

c) Comment out the existing activity transition for `SpawnSubWorkFlow_PROJECTTAG` by adding "<!--" at the beginning of the transition and adding "-->" at the end of the transition. Example for the transition to be commented out is as follows:

```
<Transition FromActivity="SpawnSubWorkFlow_PROJECTTAG"
ToActivity="PrepareForImport"/>
```

d) Add the activity transitions as shown in the following example:

```
<Transition FromActivity="SpawnSubWorkFlow_PROJECTTAG"
ToActivity="SpawnSubWorkFlow_NEWREPO "/>
<Transition FromActivity="SpawnSubWorkFlow_NEWREPO"
ToActivity="PrepareForImport "/>
```

Customizing Workflow for a Newly Added Relationship

To customize the workflow for a newly added relationship, perform the following steps:

- NEWRELATONSHIP (newly created relationship)
- Its corresponding data source and input maps should be created in the Fulfillment Catalog Metadata Studio Project and deployed successfully into the Fulfillment Catalog enterprise.
- Follow the naming convention of Repository name, its data source name, and input maps names as per `RelationshipDataMap.csv` file present in `$AC_HOME/samples/` directory. Ensure corresponding `CSV` file as per the created data source for import.

Procedure

1. Open the `$AC_HOME/samples/` directory and edit the `RelationshipDataMap.csv` file as shown in the following example:
2. After the line `REQUIRES_PRODUCT, GROUPREQUIRESPRODUCTS_PARENT_IMAP, REL_GROUPREQUIRESPRODUCT_DS, RelationshipAttribute_GroupRequiresProducts.csv` add the new line for every forward relationship of the new repo `NEWREPO, <RELATIONSHIPNAME>PARENT_IMAP, REL<RELATIONSHIPNAME>DS, RelationshipAttribute<RELATIONSHIPNAME>.csv`.

   - The first line of `RelationshipDataMap.csv` indicates that the reponame should be written first followed by `ForwardRelationshipNameInputMap` name, datasource name, and CSV file name, in the same order it should be imported.
   - Adhere to the `<RELATIONSHIPNAME>_PARENT_IMAP` format for the input map when creating it in MDM Studio.
   - Adhere to the `REL_<RELATIONSHIPNAME>_DS` format for the data source when creating it in MDM Studio.
   - Adhere to the `RelationshipAttribute_<RELATIONSHIPNAME>.csv` file to be imported.

Customizing Workflow for a Newly Added Attribute to Existing Repository

There is no change required for importing newly added attribute to existing repository.

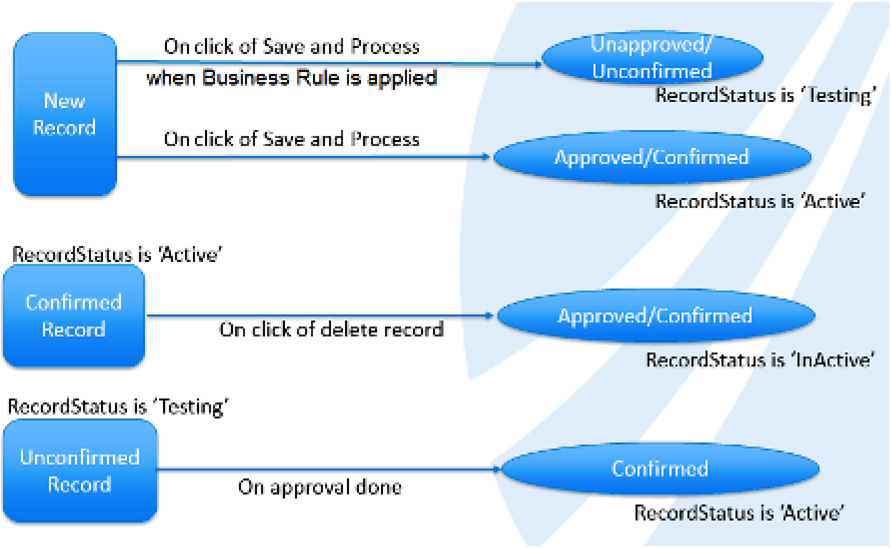
New attribute should be created for the repository in the Fulfillment Catalog Metadata Studio Project and the data source, and the input map of the repository should be modified and deployed into the Fulfillment Catalog enterprise successfully. The data source and input map names that need to be modified should be present in the `$AC_HOME/samples/CatalogDataMap.csv`.

Customizing Workflow for Newly Added Relationship Attribute to Existing Relationship

There is no change required for importing newly added attribute to existing relationship.

> New relationship attribute should be created for the relationship and its corresponding data source. Input maps in the Fulfillment Catalog Metadata Studio should be modified and deployed to the Fulfillment Catalog enterprise successfully. The data source and input map names that should be modified should be present in the `$AC_HOME/samples/RelatonshipDataMap.csv`.

Enforcement of RecordStatus



Frequently Asked Questions

The following are a list of Frequently Asked Questions (FAQs)

1. Question 1
2. Question 2
3. Question 3

Question 1

I am facing the error "`JAV-8001: Unexpected error. Class: 'ActiveCatalogUtility' and method name: 'initiateWorkflow'. Additional information: Unknown`" on a newly created enterprise. What do I do?
If you are facing the error "`JAV-8001: Unexpected error. Class: 'ActiveCatalogUtility' and method name: 'initiateWorkflow'. Additional information: Unknown`" on a newly created enterprise, then the following sections will provide you information about the initial checks that are to be performed, and a workaround that will act as a solution.

Verify if the problem occurs only on a newly created enterprise

First, you need to verify if the problem is actually happening on a newly created enterprise. To verify the same, perform the following steps:

Procedure

1. Click **Master Data** > **Datasources**. Ensure that the data source row count is zero for all data sources.
2. Check the message displayed in the `$MQ_LOG/sqlldr.log` . If the message is "`fix PATH`" then one of the following conditions are true:
   a) The environment variable `PATH` is correctly set, that is, `$MQ_HOME/bin` will appear before `$ORACLE_HOME/bin`.
   b) The `LD_LIBRARY_PATH` is correctly set to `$ORACLE_HOME/lib:$LD_LIBRARY_PATH`.
   c) If step (a) is not true then the JVM arguments has the same `PATH` in the JBOSS startup script.

Workaround

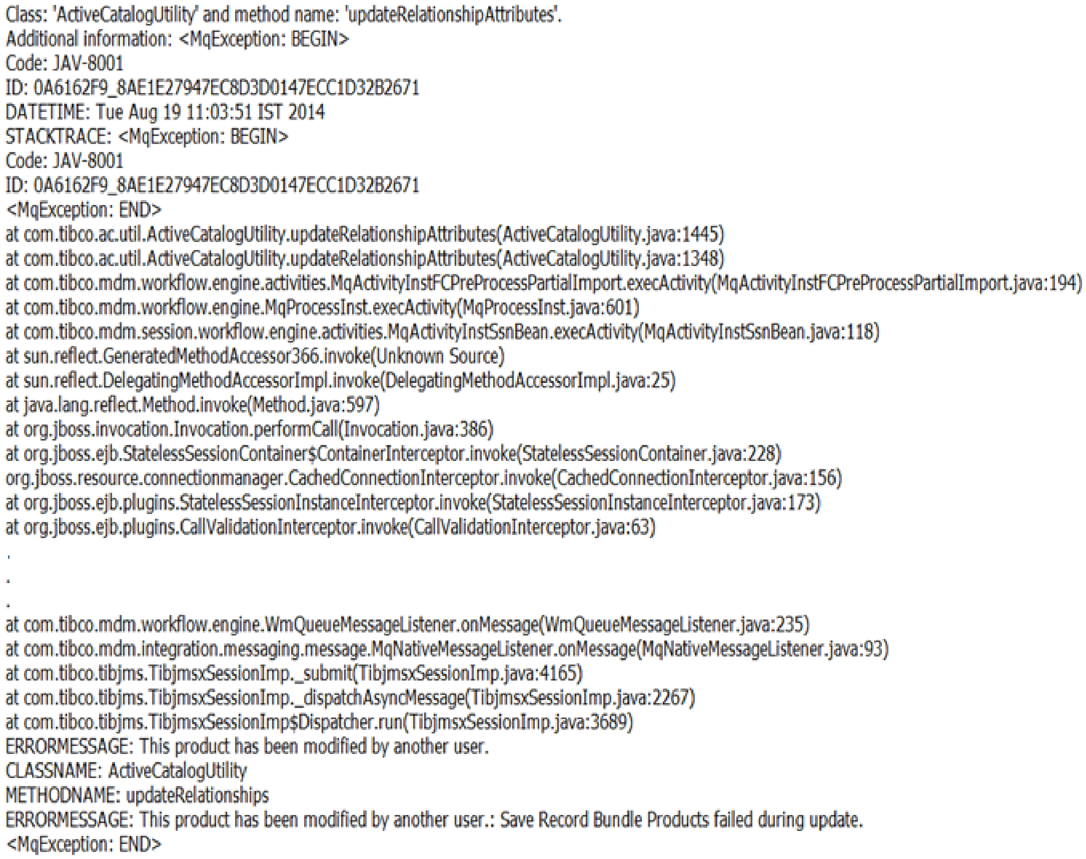
The workaround provided is the solution to the mentioned scenario:

Fix the `$PATH` environment variable correctly and restart the application.

Question 2

When performing the Partial Import feature an error message `This product has been modified by another user.` is displayed during the Partial Import and exits the feature.

Import Event Error during Partial Import "This Product has been Modified by Another User"



```
Class: 'ActiveCatalogUtility' and method name: 'updateRelationshipAttributes'.
Additional information: <MqException: BEGIN>
Code: JAV-8001
ID: 0A6162F9_8AE1E27947EC8D3D0147ECC1D32B2671
DATETIME: Tue Aug 19 11:03:51 IST 2014
STACKTRACE: <MqException: BEGIN>
Code: JAV-8001
ID: 0A6162F9_8AE1E27947EC8D3D0147ECC1D32B2671
<MqException: END>
at com.tibco.ac.util.ActiveCatalogUtility.updateRelationshipAttributes(ActiveCatalogUtility.java:1445)
at com.tibco.ac.util.ActiveCatalogUtility.updateRelationshipAttributes(ActiveCatalogUtility.java:1348)
at com.tibco.mdm.workflow.engine.activities.MqActivityInstFCPreProcessPartialImport.execActivity(MqActivityInstFCPreProcessPartialImport.java:194)
at com.tibco.mdm.workflow.engine.MqProcessInst.execActivity(MqProcessInst.java:601)
at com.tibco.mdm.session.workflow.engine.activities.MqActivityInstSsnBean.execActivity(MqActivityInstSsnBean.java:118)
at sun.reflect.GeneratedMethodAccessor366.invoke(Unknown Source)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
at java.lang.reflect.Method.invoke(Method.java:597)
at org.jboss.invocation.Invocation.performCall(Invocation.java:386)
at org.jboss.ejb.StatelessSessionContainer$ContainerInterceptor.invoke(StatelessSessionContainer.java:228)
org.jboss.resource.connectionmanager.CachedConnectionInterceptor.invoke(CachedConnectionInterceptor.java:156)
at org.jboss.ejb.plugins.StatelessSessionInstanceInterceptor.invoke(StatelessSessionInstanceInterceptor.java:173)
at org.jboss.ejb.plugins.CallValidationInterceptor.invoke(CallValidationInterceptor.java:63)
.
.
.
at com.tibco.mdm.workflow.engine.WmQueueMessageListener.onMessage(WmQueueMessageListener.java:235)
at com.tibco.mdm.integration.messaging.message.MqNativeMessageListener.onMessage(MqNativeMessageListener.java:93)
at com.tibco.tibjms.TibjmsxSessionImp._submit(TibjmsxSessionImp.java:4165)
at com.tibco.tibjms.TibjmsxSessionImp._dispatchAsyncMessage(TibjmsxSessionImp.java:2267)
at com.tibco.tibjms.TibjmsxSessionImp$Dispatcher.run(TibjmsxSessionImp.java:3689)
ERRORMESSAGE: This product has been modified by another user.
CLASSNAME: ActiveCatalogUtility
METHODNAME: updateRelationships
ERRORMESSAGE: This product has been modified by another user.: Save Record Bundle Products failed during update.
<MqException: END>
```

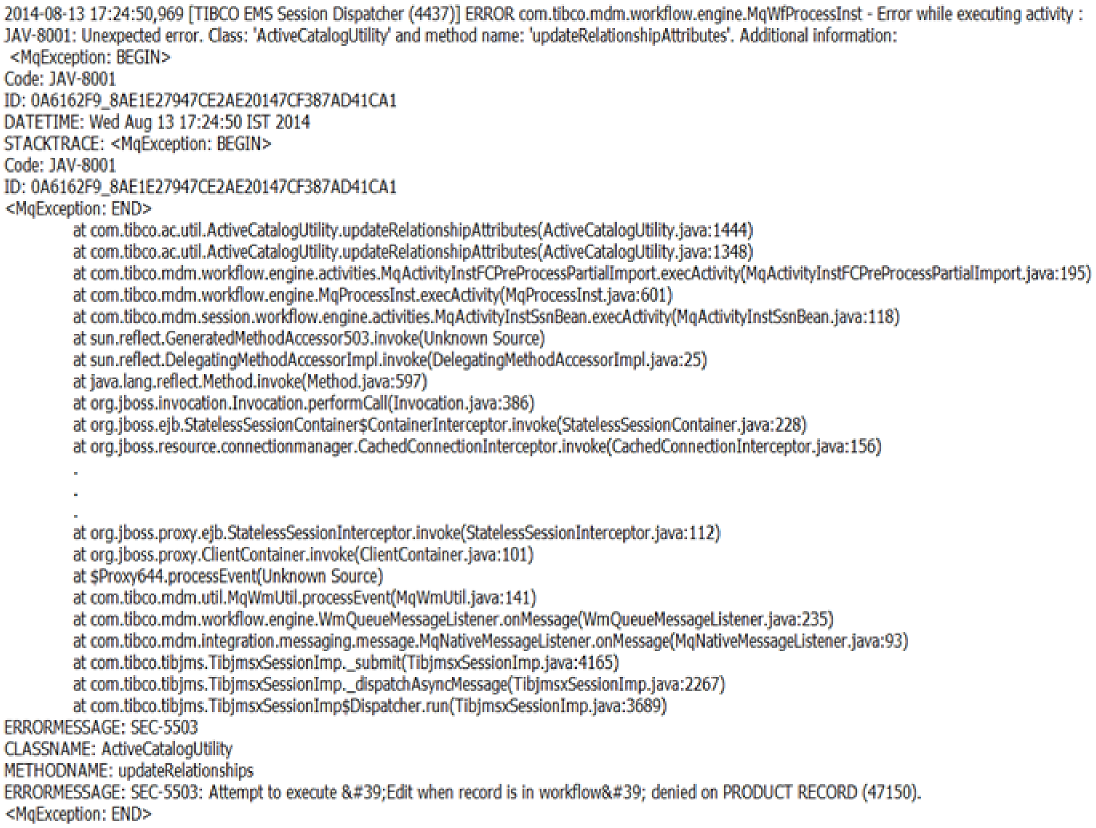
Workaround

Redo the partial import into the same enterprise. If the error reappears perform the import procedure again. Repeat the task till the Partial Import event shows the success status in the event log screen.

Question 3

When performing the Partial Import feature an error message `SEC-5503: Attempt to execute &#39;Edit when record is in workflow&#39; denied on <Repository> RECORD.` is displayed during the Partial Import and exits the feature.

Import Event Error during Partial Import "SEC-5503: Attempt to execute &#39;Edit when record is in workflow&#39; denied on <Repository> RECORD"



```
2014-08-13 17:24:50,969 [TIBCO EMS Session Dispatcher (4437)] ERROR com.tibco.mdm.workflow.engine.MqWfProcessInst - Error while executing activity :
JAV-8001: Unexpected error. Class: 'ActiveCatalogUtility' and method name: 'updateRelationshipAttributes'. Additional information:
 <MqException: BEGIN>
Code: JAV-8001
ID: 0A6162F9_8AE1E27947CE2AE20147CF387AD41CA1
DATETIME: Wed Aug 13 17:24:50 IST 2014
STACKTRACE: <MqException: BEGIN>
Code: JAV-8001
ID: 0A6162F9_8AE1E27947CE2AE20147CF387AD41CA1
<MqException: END>
        at com.tibco.ac.util.ActiveCatalogUtility.updateRelationshipAttributes(ActiveCatalogUtility.java:1444)
        at com.tibco.ac.util.ActiveCatalogUtility.updateRelationshipAttributes(ActiveCatalogUtility.java:1348)
        at com.tibco.mdm.workflow.engine.activities.MqActivityInstFCPreProcessPartialImport.execActivity(MqActivityInstFCPreProcessPartialImport.java:195)
        at com.tibco.mdm.workflow.engine.MqProcessInst.execActivity(MqProcessInst.java:601)
        at com.tibco.mdm.session.workflow.engine.activities.MqActivityInstSsnBean.execActivity(MqActivityInstSsnBean.java:118)
        at sun.reflect.GeneratedMethodAccessor503.invoke(Unknown Source)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:597)
        at org.jboss.invocation.Invocation.performCall(Invocation.java:386)
        at org.jboss.ejb.StatelessSessionContainer$ContainerInterceptor.invoke(StatelessSessionContainer.java:228)
        at org.jboss.resource.connectionmanager.CachedConnectionInterceptor.invoke(CachedConnectionInterceptor.java:156)
        .
        .
        .
        at org.jboss.proxy.ejb.StatelessSessionInterceptor.invoke(StatelessSessionInterceptor.java:112)
        at org.jboss.proxy.ClientContainer.invoke(ClientContainer.java:101)
        at $Proxy644.processEvent(Unknown Source)
        at com.tibco.mdm.util.MqWmUtil.processEvent(MqWmUtil.java:141)
        at com.tibco.mdm.workflow.engine.WmQueueMessageListener.onMessage(WmQueueMessageListener.java:235)
        at com.tibco.mdm.integration.messaging.message.MqNativeMessageListener.onMessage(MqNativeMessageListener.java:93)
        at com.tibco.tibjms.TibjmsxSessionImp._submit(TibjmsxSessionImp.java:4165)
        at com.tibco.tibjms.TibjmsxSessionImp._dispatchAsyncMessage(TibjmsxSessionImp.java:2267)
        at com.tibco.tibjms.TibjmsxSessionImp$Dispatcher.run(TibjmsxSessionImp.java:3689)
ERRORMESSAGE: SEC-5503
CLASSNAME: ActiveCatalogUtility
METHODNAME: updateRelationships
ERRORMESSAGE: SEC-5503: Attempt to execute &#39;Edit when record is in workflow&#39; denied on PRODUCT RECORD (47150).
 <MqException: END>
```

Workaround

Procedure

1. Click **Administration** > **Resource Security**. The **Manage Resource Security** page is displayed.
2. Select the value **Repository** for the field **Resource Type**.
3. Select a value for the field **Resources**.

   Do not select the value **All**.

4. Click the **Show Permissions** button. The **Manage Resource Security for Repositories** page is displayed.
5. Select the value **Record** for the field **Resource Type** and click **Add Grantee**. The **Select Grantee** page is displayed.
6. Select **admin** and click the **OK** button.
7. Select the option **Allow** for the **Full Control** permission and click the **Save** button. A message saying that the permissions are saved is displayed.

   Repeat the entire procedure for all the repositories by selecting the repository name as the value for the **Resources** field in Step 3.

Samples

The following list contains samples that are available for your reference:

- Conditional Affinity Sample
- Sample Order XML
- Sample Plan Item XML
- Sample XPATHs

Conditional Affinity Sample

The conditional affinity sample is as follows:

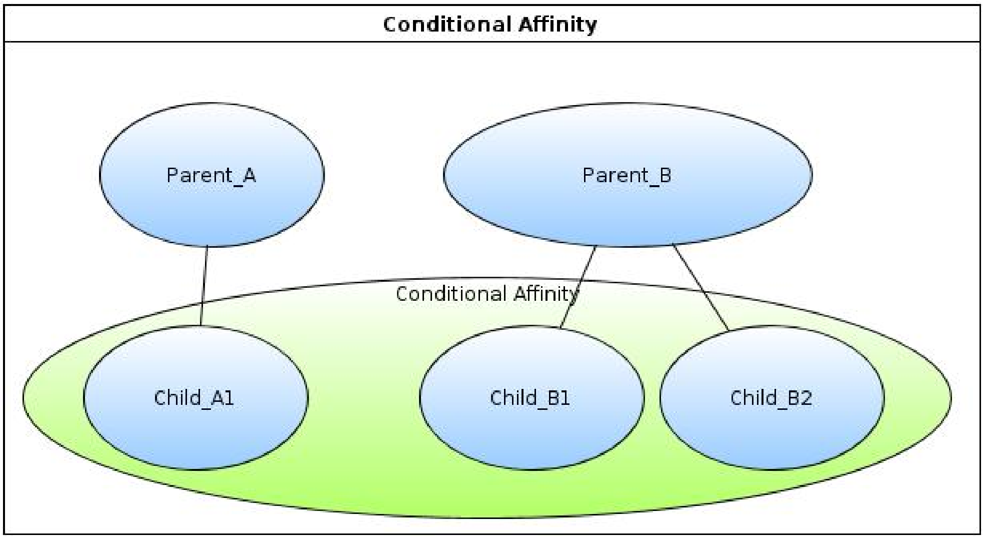
A product model has two parents:

- Parent_A
- Parent _B

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

- Child_A1
- Child_B1
- Child_B2

Conditional Affinity



The XPATH defined in the product model is evaluated against the submitted order schema.

The following table describes the attribute-based conditional affinity scenarios:

Attribute	Sample XPATH Expressions	Order Payload

AffinityCondition	`exists($var/Order/ OrderHeaderUDF[name=UDFNAME and value="UDFVALUE"])`	`<ord1:udf>`
		`<ord1:name>UDFNAME</ ord1:name>`
		`<ord1:value>UDFVALUE</ ord1:value>`
		`</ord1:udf>`

Attribute	Sample XPATH Expressions	Order Payload

AffintyCorrelation	`$var/Order/orderlines[productID='Child_A1']/`
	`OrderlinesUDF[name='UDFNAME']/value/text()`	

AffintyParentGroup	Child_B1 and Child_B2 have immediate parent. The two will be affinity grouped when: `AffintyParentGroup=true`	

AffinityActionValue
The `affinityAction` Group must be true
	`$var/Order/orderlines[productID='Child_A1']/`
	`OrderlinesUDF[name='UDFNAME']/value/text()`	

AffintyCorrelation

```
<ord1:line>
  <ord1:lineNumber>1</
ord1:lineNumber>
  <ord1:productID>Child_A1</
ord1:productID>
  <ord1:quantity>1</
ord1:quantity>
  <ord1:uom>1</ord1:uom>
  <ord1:action>PROVIDE</
ord1:action>
  <ord1:actionMode>New</
ord1:actionMode>
  <ord1:udf>
   <ord1:name>UDFNAME</
ord1:name>
   <ord1:value>UDFVALUE</
ord1:value>
  </ord1:udf>
</ord1:line>
```

AffinityActionValue

```
<ord1:line>
 <ord1:lineNumber>1</
ord1:lineNumber>
 <ord1:productID>Child_A1</
ord1:productID>
 <ord1:quantity>1</
ord1:quantity>
 <ord1:uom>1</ord1:uom>
 <ord1:action>PROVIDE</
ord1:action>
 <ord1:actionMode>New</
ord1:actionMode>
 <ord1:udf>
 <ord1:name>UDFNAME</
ord1:name>
  <ord1:value>UDFVALUE</
ord1:value>
  </ord1:udf>
</ord1:line>
```

Sample Order XML

The sample order XML is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<Order Id="544">
    <orderID>8ae1f9ac3898656f0138986ae70c0001</orderID>
    <sessionID>CORRELATION-3baee8b0-b483-47aa-89b2-bf7b03d0c41f</sessionID>
    <orderlines Id="545">
        <lineID>1</lineID>
        <productID>CFS TV</productID>
        <action>PROVIDE</action>
        <quantity>1.0</quantity>
        <requiredByDate>2011-04-30T23:50:00+05:30</requiredByDate>
        <LineUsed>false</LineUsed>
        <OrderlinesUDF Id="546">
            <name>OrderRef</name>
            <value>OrderRefID</value>
            <flavor>input</flavor>
        </OrderlinesUDF>
    </orderlines>
    <orderlines Id="547">
        <lineID>2</lineID>
        <productID>CFS Live Box</productID>
        <action>PROVIDE</action>
        <quantity>1.0</quantity>
        <requiredByDate>2011-04-30T23:50:00+05:30</requiredByDate>
        <LineUsed>false</LineUsed>
        <OrderlinesUDF Id="548">
            <name>OrderRef</name>
            <value>OrderRefID</value>
            <flavor>input</flavor>
        </OrderlinesUDF>
    </orderlines>
    <orderlines Id="549">
        <lineID>3</lineID>
        <productID>CFS VOIP</productID>
        <action>PROVIDE</action>
        <quantity>1.0</quantity>
        <requiredByDate>2011-04-30T23:50:00+05:30</requiredByDate>
        <LineUsed>false</LineUsed>
        <OrderlinesUDF Id="550">
            <name>OrderRef</name>
            <value>OrderRefID</value>
            <flavor>input</flavor>
        </OrderlinesUDF>
    </orderlines>
    <status>NewOrder</status>
    <currentTime>2012-07-18T10:19:03+05:30</currentTime>
    <TineDelay>0</TineDelay>
    <customerref>Apple</customerref>
    <OrderHeaderUDF Id="551">
        <name>Company</name>
        <value>Orange</value>
        <flavor>input</flavor>
    </OrderHeaderUDF>
    <Originator>Orchestrator</Originator>
    <OrderRef>OrderRefID</OrderRef>
    <businessTransactionID>a7eb1e1de1fa45c993f65589dba70648</businessTransactionID>
</Order>
```

Sample Plan Item XML

The sample plan item is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<PlanItem Id="2169">
    <planID>PF1</planID>
    <parentID>CORRELATION-1b1260e6-9cdd-4903-a184-d473aa11b622</parentID>
    <lineID>2</lineID>
    <dependentOn>PF10.7747556</dependentOn>
    <planDesc>  PROVIDE</planDesc>
    <planItemID>PF10.8786092</planItemID>
    <EOL>N</EOL>
    <TimeDelay>0</TimeDelay>
    <status>addDependency</status>
    <singleUse>false</singleUse>
    <sequence>0</sequence>
    <sequenceName>leaf</sequenceName>
    <action>PROVIDE</action>
    <productID>GSMDataService</productID>
    <itemMark4Del>false</itemMark4Del>
    <mustComplete>true</mustComplete>
    <affinityType>ConditionalAffinity</affinityType>
    <affintyPlanID>PF1</affintyPlanID>
    <affintyPlanDesc> AFFINITY PROVIDE</affintyPlanDesc>
    <udfs Id="2170">
        <name>TASKID</name>
        <value>PF10.8786092</value>
        <flavor>config</flavor>
    </udfs>
    <udfs Id="2172">
        <name>PRODUCTID</name>
        <value>GSMDataService</value>
        <flavor>config</flavor>
    </udfs>
    <udfs Id="2173">
        <name>RECORD_TYPE</name>
        <value>SERVICE</value>
        <flavor>config</flavor>
    </udfs>
    <udfs Id="2174">
        <name>MSISDN</name>
        <value>123</value>
        <flavor>input</flavor>
    </udfs>
    <Ancestors>PF10.7747556</Ancestors>
    <cancelUsed>false</cancelUsed>
    <M_EP_UDFS Id="2171">
        <name>M_EP_UDFS</name>
        <value>PF10.8786092</value>
    </M_EP_UDFS>
    <pI_Used>false</pI_Used>
    <isLeaf>false</isLeaf>
    <counter>0</counter>
    <LinkID>1</LinkID>
    <affinityCorrelation>$var/PlanItem[productID='GSMLine']/udfs[name='MSISDN']/
value/text()</affinityCorrelation>
    <affinityParentGroup>false</affinityParentGroup>
    <affinityActionGroup>false</affinityActionGroup>
    <isDynamic>false</isDynamic>
</PlanItem>
```

Sample XPATHs

The sample is as follows:

- `<ns0:affinityCondition>exists($var/Order/OrderHeaderUDF[name="SubscriberProduct"and value="Product BB Network Access"])</ns0:affinityCondition>`
- `<ns0:affinityCorrelation>exists($var/Order/OrderHeaderUDF[name="SubscriberProduct"and value="Product BB Network Access"])</ns0:affinityCorrelation>`
- `<ns0:affinityActionValue>$var/Order/orderlines[productID='CFS STB']/action/text()</ns0:affinityActionValue>`
- `<affinityCorrelation>$var/PlanItem[productID='GSMLine']/udfs[name='MSISDN']/value/text()</affinityCorrelation>`