White Paper
Using Item Integrity Rules in Oracle Fusion Product Hub

This white paper describes how to set up Item Rules to check Integrity of the Item data entered in Oracle Fusion Product Hub

Document Control

Change Record

Date	Author	Version	Change Reference
04-Jan-08	Ganesh Tirtur	1.0	First draft
07-Jan-08	Ganesh Tirtur	1.1	Incorporated the review comments
09-Mar-08	Ganesh Tirtur	1.2	Made changes to Business Entities (Only 3 entities are supported), updated Structure Attribute Group, Component Type rules and added updates for new features.
13-May-13	Magesh Kesavapillai	1.3	Revised the content to reflect the enhancements and changes done in and upto fusion release 1.0.6.
24-Jan-14	Magesh Kesavapillai	1.4	Added details about the following functions: - Exists function - IN, NOT_IN, BETWEEN, TO_NUMBER, DECODE - assignedtoOrg and assignedtoCatalog
06-May-16	Magesh Kesavapillai	1.5	- Added details for AUTO_SEQUENCE function - Added description about Impact analysis. - Added description for Web service invocation feature
19-Jan-17	Magesh Kesavapillai	1.6	Added details regarding the Impact Analysis enhancements - criteria based scope - scheduling of analysis
21-Apr-17	Magesh Kesavapillai	1.7	Fixed a typo with InvokeWebService( )
04-Jul-17	Magesh Kesavapillai	1.8	Added details regarding referencing of custom object data in rules

Reviews and Approvals

Reviewers

Name	Role	Status	Version Reviewed	Date Reviewed	Comments
Ganesh Tirtur			1.1	06-Jan-08	Posted on review.us.oracle.com
Gabriel Lomeli			1.0	04-Jan-08	Posted on review.us.oracle.com

Approvers

Name	Role	Status	Version Approved	Date Approved	Comments
	Development Owner				

Table of Contents

This white paper contains the following information.

Introduction

Oracle Product Data Hub allows users to define and manage Items and Structures in multiple organizations and organization hierarchies.

Item Rules are used to define integrity constraints on the attributes of those Items, Item Relationships, Item Supplier Associations and Structures. Integrity constraints can be defined on Operational as well as on User Defined Attributes (UDAs).

A typical rule might check that a certain attribute is less than another. For example:

```
[Item].[Physical_Attributes].[Net_Weight] <=
[Item].[Logistics].[Shipping_Weight]
```

In other words, the net weight of an item always has to be less than or equal to the shipping weight.

As well as checking constraints, the user can define assignment formulas. The formula for the "Daily Waste Percent" might be:

```
[Item].[Inventory].[Total_Waste_Percent] / [Item].[Inventory].[Shelf_Life]
```

So the daily waste percent is the total waste percent divided by the shelf life in days.

This Whitepaper discusses about the features and functions available in Fusion release 1.0.6. The functions and syntax for expressions are slightly different in earlier releases. It is suggested to refer to the respective documents for such cases.

Overview

Rules and Rule Sets

Rules encapsulate a single integrity constraint, like:

```
[Item].[Physical_Attributes].[Net_Weight] <=
[Item].[Logistics].[Shipping_Weight]
```

Rule Sets group multiple Rules together. The Rule Set is tied to an Attribute Group, Item Class or Change Type.

This allows the expressions the user types in to be validated by checking for allowable attributes. If the Rule Set is tied to an Attribute Group, then only the attributes in that group can be used in the expressions. If the Rule Set is tied to an Item Class, then only the Attribute Groups valid for that Item Class can be used and it applies to Change Rule Sets.

"Composite" Rule Sets can be used to aggregate Rule Sets that operate on different Attribute Groups and Item Classes. To activate a Rule Set it is assigned to the Master Rule set. Master Rule set is a seeded rule set that, in general, will contain all the rule sets along with sequence number; the sequence number denotes the order in which rule sets are to be executed in the run-time.

Validation and Assignment RulesThe Rules that make up the Integrity Validations and Assignments follow a simple syntax. Each attribute is referenced by the Entity name, its Attribute Group name followed by the Attribute name;
Entity name could be any one of these Item, Revision, CrossReference, RelatedItem, GTIN StyleItem, Structure or Supplier based on which attribute you want to refer to
For example,
[Item].[Physical_Attributes].[NetWeight]

Here, Item means that the attribute belongs to Item entity; also note that "Physical_Attributes" is the internal name of the Attribute Group, and "NetWeight" is the internal name of the Attribute.

Attributes can be combined into more complex expressions using the build-in operators and functions (see

Expression Reference for a complete description).

Comparison Operators	== != < <= > >=
Logical Operators	and or not (can also use && and ||)
String Functions	compare contains endsWith match startsWith + indexOf length lowercase substring trim uppercase
Math Functions	+ - * / sum abs amount min max round roundup rounddown
Date Functions	+ (Date + days) - (Date - days)
Comparison to current Production Value Functions	changed delta percent previous

The following expression checks if the Item name 3rd letter is "A" or "E" :

```
substring([Item].[Item_Basic].[NAME],2,3) == "A" or
substring([Item].[Item_Basic].[NAME],2,3) == "E"
```

Note that a more concise expression would have used the "in" operator, but the above illustrates the use of "or".

Example

The following integrity constraints are typical of what one might expect for an item, in this case a computer motherboard:

Validations

- Minimum CPU Speed (in MHz) has to be less than Maximum CPU Speed.

- If Purchasable Flag is set to Yes, then List Price has to be set.
- Unit Height cannot change by more than 3% without requiring approval.

Assignments

- Lead Percent is Total Lead Mass divided by Unit Weight.
- Sellable Date (date when item can be sold) is 10 days after the Availability Date.

To implement the above rules we have to create the appropriate Rule Sets and then add them into the Master Rule set:

- Create Assignment Rule Set
- Create Validation Rule Set
- Create Composite Rule Set
- Add Composite Rule Set into Master Rule set.

.

Assignment Rule Set

As a general rule, Rule Sets that do assignments should be executed before validations, since it lets the user write validations to ensure that the result of the assignments are valid.

Creating Rule Set

1. Go to the setup task “Manage Item Rulesets”
2. Click on “Create” button to create a Rule Set
3. Enter Display Name, Description and Internal Name.
   The Internal Name will not be changeable after the Rule Set is created.
4. Select Composite = “NO”.
   Rule Sets that contain rules have Composite set to “No”. Rule Sets that contain other Rule Sets have Composite set to “Yes”.
5. Select Type = “Assignments”.
6. Select “Item Class” in Association Type dropdown
7. Enter or select appropriate Item Class.
8. Hit “Continue”

   This creates an empty Rule Set with the appropriate context. All the Attribute Groups in the selected Item Class are available for authoring rules. We are now going to create the actual rules.

9. Click on the newly created rule set (in the above steps) to edit it
10. In the “Edit Rule Set” page click on “Create Rule” button in “Rules” tab
11. The assignment rule is:

```
Lead Percent is Total Lead Mass divided by Unit Weight.
```

Enter the following values:

Screen Field	**Value**
Sequence	10
Name	Lead_Percent
Description	Lead Percent is Total Lead Mass divided by Unit Weight.
Target Business Entity	Item
Set Value of Target Attribute Group and Target Attribute	Hazard.Lead_Percent
Primary If Expression	CAN BE NULL/EMPTY
Secondary If Expression	CAN BE NULL/EMPTY
Return Value	round(([Item].[Hazard].[Lead_Mass] / [Item].[Physical_Attributes].[Unit_Weight]) * 100, 2)
User Message	Percent is Mass ($[Item].[Hazard].[Lead_Mass]$) divided by Weight ($[Item].[Physical_Attributes].[Unit_Weight]$).

A few things to note:

- Primary/Secondary If expressions can be empty if they are not required.
- It is not necessary to check if attributes are null or, in the case of division, zero). In this case the assignment will be ignored.
- The optional user message is shown to the user to inform him/her about the assignment that took place.
- Users can use right click menu option (Insert Attribute) to populate the attribute group/attribute data into the fields such as Primary If expression, Secondary If expression, Return Value etc.

12. Click on Validate button to ensure that all the attribute names are typed in correctly. Note that it is currently necessary for the user to know the internal names of all Attribute Groups and Attributes. User can use right click menu to select the required attribute group and attribute; the internal names will be populated.
13. Once you are satisfied with the result, click “Save”.

This enters the first assignment rule into the Rule Set. We will now enter the second rule. Note that rules are executed in the sequence specified by the user. Therefore if an Attribute’s expression depends on a previously calculated value, the user has to ensure that the previous value appears ahead of the Attribute and is therefore computed first.

14. The second rule we will enter will show the if/else if capabilities of the assignment rule:

```
if ([Item].[Main].[Sellable_Flag] == "Yes")

      if ([Item].[Item_basic].[Item_Class] == "Perishables")
      then [Item].[Marketing].[Sellable_Date] =
[Item].[Planning].[Availability_Date] + 3

      else if ([Item].[Item_basic].[Item_Class]== "Consumables")
      then [Item].[Marketing].[Sellable_Date] =
      [Item].[Planning].[Availability_Date] + 6

      else [Item].[Marketing].[Sellable_Date] =
      [Item].[Planning].[Availability_Date] + 10
```

15. On the Edit Rule Set page, Rules tab,click "Create Rule".
    As mentioned before, the order of the rules matters if you want to use the result of previous assignments in the current calculation.
    In this case there is no dependency between the two rules, so the order is organizational.

16. Enter the rule as follows. Note that you will have to add an additional 2 rows of "Then Expressions" by selecting the last row in the table and clicking "Add Row":

Screen Field	**Value**
Name	Sellable_Date_Calculation
Description	Calculate Sellable date based on the Availability Date.
Set Value of Target Attribute Group and Target Attribute	Item.Marketing.Sellable_Date
Primary If Expression	Item.Main.Sellable_Flag == "Yes"
Secondary If Expression	Item.Item_basic.Item_Class == "Perishables"
Return Value	Item.Planning.Availability_Date + 3
Secondary If Expression	Item.Item_basic.Item_Class == "Consumables"
Return Value	Item.Planning.Availability_Date + 6
Secondary If Expression	CAN BE NULL/EMPTY
Return Value	Item.Planning.Availability_Date + 10
User Message	

- Leaving the last "Secondary If Expression" blank means it defaults to "True" and is always selected.
- No User message was entered, so nothing is shown to the user.

17. Click on Validate button and Save once all errors are removed.
18. That completes the Assignment Rule Set.

Organization assignment Rules

In our previous examples, we saw how users can assign values to attributes based on given criteria. In similar way, users can create rules to automatically assign items to organization(s), based on given criteria; criteria include attribute value, another organization assignment or a catalog assignment.

For example,
If an item is an “In-house” item, then it should be assigned to Organizations - Boston Manufacturing (Org code = M2) and Detroit Manufacturing (Org code = M3)
If an item that is a “bought” item, it should be assigned to the Organization – Texas Distribution Centre (Org code = T1)

Enter the rule as follows. Note that you will have to add additional row of “Then Expressions” by selecting the last row in the table and clicking “Add Row”:

Screen Field	**Value**
Name	Organization assignment
Description	Auto assignment of item to organization based on given attributes.
Return type	Organization
Primary If Expression	true
Secondary If Expression	Item.Logistics.Product_type == “In-house”
Return Value	M2, M3
Secondary If Expression	Item.Logistics.Product_type == “Bought-out”
Return Value	T1
User Message	

On the transaction side, when a user provides a specific value for a particular attribute of an item, the item is automatically assigned to the specified organization(s) as defined in the rule.

Catalog assignment Rules

Users can define assignment rules to automatically assign items to one or more Catalog/categories based on item/item revision level attributes or an organization assignment or another Catalog assignment.

Example:

Screen Field	**Value**
Name	Catalog assignment
Description	Auto assignment of item to catalog/category based on given attributes.
Return type	Organization
Primary If Expression	true
Secondary If Expression	Item.Properties.Size == “XS” AND Item.Properties.Material == “Wool”
Return Value	Catalog[KidsWear].Category[Winter]
User Message	XS, Wool item is assigned to Winter Category under KidsWear Catalog.

Validation Rule Set

Validation Rule Sets are created in much the same fashion as Assignment Rule Sets, except that the Type is set to “Validation”.

1. On Manage Rule Sets page, click “Create Rule Set” button
2. Enter Name, Description and Internal Name.
    a. The Internal Name will not be changeable after the Rule Set is created.
3. Select Composite = “NO”.
4. Select Type = “Validations”.
5. Select “Item Class” from Association Type.
6. Enter or select appropriate Item Class.
7. Click “Ok”
8. Go to “Edit Rule Set” page by clicking on the rule set and select “Rules” Tab (if not already selected)
9. Click “Create Rule” button.
10. The validation rule is:

```
Minimum CPU Speed (in MHz) has to be less than Maximum CPU Speed.
```

Enter the following values:

Screen Field	**Value**
Name	Max_Min_CPU_Check
Description	Check that Minimum CPU Clock Speed is less than Maximum CPU Clock Speed.
Severity	Reject
If Expression	
Validation Condition	Item.Motherboard_Spec.Min_CPU_Speed <= Item.Motherboard_Spec.Max_CPU_Speed
User Message	Minimum CPU Speed has to be less than or equal to Maximum CPU Speed.

Severity

The “Severity” of the Rule determines what action is taken if the validation fails. The Severity actions are as follows:

Severity	**Action**
Reject	The Business Entity cannot be saved until the validation passes.
Needs Approval	This data requires that a Change Order be created.
Warning	The Explanation Message is displayed to the user, but the entity can still be saved.

- "If" expressions can be empty if they are not required.
- It is not necessary to check if attributes are null or, in the case of division, zero). In this case the validation will be ignored.
  Null values in general cause the validation to be ignored, unless you explicitly want to test for a null value with the "isnull" function.
  This is to facilitate incremental processing of the item, where the complete item might not be completely filled in until the end of the process.

2. Click Validate and then Save buttons once rule is entered satisfactorily.
3. The next validation rule is:

   ```
   If Purchasable Flag is set to Yes, then List Price has to be set.
   ```

   This is an example of one field having to have a value based on the value of another field.

   Enter the following values:

Screen Field	**Value**
Name	Require_List_Price
Description	List Price has to be set if Purchasable Flag is set to "Yes".
Severity	Reject
If Expression	Item.Purchasing.Purchasable == "Yes"
Validation Expression	(!isnull(Item.Purchasing.List_Price))
User Message	If Purchasable is set to Yes then List Price cannot be null.

- The "!" in "!isnull" is shorthand for "not".

4. Click Validate and then Save buttons once rule is entered satisfactorily.
5. The next validation rule is:

   ```
   Unit Height cannot change by more than 3% without requiring approval.
   ```

   This is an example of checking the new value against the value that is currently in production. If this validation fails, the user will be required to create a Change Order for this change.

   Enter the following values:

Screen Field	**Value**
Name	Height_Change

Description	Unit Height cannot change by more than 3%.
Severity	Needs Approval
If Expression	
Validation Expression	abs(percent(Item.Physical_Attributes.Unit_Height)) > 3.0
User Message	A Unit Height change of 3% requires a Change Order.

- The “percent” function returns the percentage change of the new value to the value in Production. This change can be positive or negative, hence the “abs” function converts it to an absolute change.

6. Click Validate and then Save buttons once rule is entered satisfactorily.
7. We are now done with our validation Rule Set. Click on “Attributes” tab to review all the Attributes used in this Rule Set. Click “Save and Close” to return to the Manage Rule Sets page.

Organization Validation Rules:

Users can define validation rules to validate item-organization assignment based on attributes or another Organization assignment or a Catalog assignment.
For example, write rule to enforce the following validation
When the item class of the item is Exadata Servers, the assigned organization cannot be Seattle Distribution Centre (Org code = S2)
You can set the severity for the rule as Reject, Warning or Needs approval.

Screen Field	**Value**
Name	Org validation
Description	Validate Exadata Servers assignments
Severity	Reject
If Expression	Item.Item_basic.Item_Class == “Exadata_servers”
Validation Expression	!assignedtoOrg(“S2”)
User Message	Exadata servers cannot be assigned to Seattle Distribution Centre.

Note the function assignedtoOrg(“Org_code”), this validates whether an item is assigned to the specified organization and returns either TRUE or FALSE.

Catalog/category-assignment Validation Rules:

Users can define validation rules to validate item-catalog/category assignments based on attributes or another Catalog assignment or an Organization assignment.
For example, write rule to enforce the following validation
If an item is assigned to Catalog=LadiesWear and Category=Summer, it cannot be assigned to Catalog=Kids and Category=Summer
You can set the severity for the rule as Reject or Warning.

Screen Field	**Value**

Name	Catalog assignment validation
Description	Validate catalog assignments
Severity	Reject
If Expression	assignedtoCatalog(Catalog[LadiesWear].Category[Summer])
Validation Expression	!assignedtoCatalog(Catalog[Kids].Category[Summer])
User Message	Ladies wear products cannot be assigned to Kids wear catalog.

Note the assignedtoCatalog function, this validates whether an item is assigned to the specified Catalog/Category.

Composite Rule Set

We are now going to create a "Composite" Rule Set that includes both the Rule Sets we just created. This is the Rule Set that we will assign to the "Item" entity in the final step to activate the rules we have created.

1. On Manage Rule Sets page, click "Create Rule Set" button
2. Enter Name, Description and Internal Name.
    a. The Internal Name will not be changeable after the Rule Set is created.
3. Select Composite = "Yes".
4. Select Type = "Mixed".
    a. In more complex scenarios, you might have multiple layers of composite Rule Sets, some of which you only want to contain "Validations" or "Assignments". But eventually you will want to include both kinds of rules.
5. Click "Ok" and go to Edit Rule Set page by clicking on the Rue Set hyperlink.
6. Click on "Add Rule Set" and search for the Assignment Rule Set we just created. Check it and click on "Apply/Done" button.
7. In the Included Rule Sets list, check the just included Assignment Rule Set and click the "Add Rule Set" button. Now select the Validation Rule Set.
8. The Rule Sets will be executed in the order they are listed in the Included Rule Sets. So it is important to put Validations after Assignments if you want to validate some of the assignment results you computed.
9. Click "Save" to save the composite Rule Set.

Adding the Rule Set into the Master Rule set

The rules we created are not active until we add them to the Master Rule Set.
Also, note that the draft check box should be unchecked for a rule set to be executed in the run-time. Draft rule sets are not run during regular transactions.

You are now ready to test the validations by going to the Item Update screens and editing the appropriate Attribute Groups. Updated values should be validated against the rules, and you should see the error messages appear on the screen.

You can also check out some of the other tabs in the Rule Set screen. In the Validation and Assignment Rule Sets the “Rule Sets Where Included” tab should now show the Composite Rule Set. For the Composite Rule Set the Assigned To section in the “Usages” tab should show the Item assignment.

Change Policies

User can define change policies using validation rule sets.
User can set various severities based on the policies.

<u>**Severities and their meanings:**</u>
Reject: Changes not allowed.
Warning: Changes allowed.
Needs Approval: Forces to create Change Order

Structure Change Policies

User can define change policies on structures using validation rule sets.

For e.g.
Business Policy: Any changes made to “Manufacturing” structure of type Item requires approval and no changes allowed for “Engineering” structure during “Concept” lifecycle phase.

If the Structure looks like this,

```
MfgItem (Mfg Structure, Approval Required)
|
-----------MfgItem1
|
-----------MfgItem2
|
-----------EngItem1 (Eng Structure, Changes Not Allowed)
            |
            -----------EngItem12
           |
           ------------EngItem13
```

Rule 1:

Business Entity	Item
Rule Set Name	StructurePolicyRuleSet
Rule Set Type	Validation
Association Type	Item Class
Association Name	Root Item Class
Rule Name	SturctRule1
Valid Component Rules (Check box)	Unchecked
Severity	Needs Approval
If Condition	`[Structure].[StructureAttributes].[StructureName] == "MANUFACTURING"`
Validation Condition	`false`

Rule 2:

Rule Set Name	StructurePolicyRuleSet
Rule Set Type	Validation
Association Type	Item Class
Association Name	Root Item Class
Rule Name	SturctRule2
Valid Component Rules (Check box)	Unchecked
Severity	Reject
If Condition	`[Item].[Main].[CURRENT_PHASE_CODE]=="Design"&&[Structure].[StructureAttributes].[StructureName] == "Engineering"`
Validation Condition	`FALSE`

Note: *The component attributes based change policies are not supported.*

Item Supplier Site Organization Association Rules

Rules can be used to govern create and update of Item supplier site organization association and update of association attributes – Primary and EFFs.
Ex, In the following scenario, for any item under Electronics Item class, if you want to associate a supplier XYZ, you need approval.

Business Entity	Item
Rule Set Name	SupplierAssocRuleSet
Rule Set Type	Validation
Association Type	Item Class
Association Name	Electronics
Rule Name	SuppAssocRule1
Severity	Needs Approval
If Condition	`TRUE`
Validation Condition	`[Supplier].[ItemSupplierSiteOrg].[SUPPLIER_NAME] != "XYZ"`
User Message	`You need approval to add the supplier XYZ for Electronics items.`

After creating a ruleset as above, remember to add it to the Master Rule set.
Once the rule is set, when the user tries to add a supplier site organization association with a supplier name XYZ, to an item belonging to Electronics Item class, system will force a Change order.

Number and description generation rules

Rules can be used to generate the identification and description for items, change orders, and new item requests.

Generating numbers and descriptions with rules requires the following actions:

- Creating Rule Sets
- Creating Rules
- Completing Associations

Creating Rules to Generate Numbers for New Items

This scenario illustrates how to use rules to generate numbers for newly-created items.

1. Select the task Setup Rules.
2. From the Manage Rule Sets page, create and save a rule set with the settings shown in the following table.

Field (in Rule Set)	**Value**
Display Name	Example: MyItemNumGenRuleSet1
Type	Assignments
Association Type	Item Class
Association Name	Example: MyItemClass1

3. On the Edit Rule Set page, create and save a rule with the settings shown in the following table.

Field (in Rule)	Value
Name	Example: MyItemNumGenRule1
Return Type	Item Number
Primary If Expression	True
Secondary If Expression	True
Return Value	Example: [Item].[PhysicalAttributes].[UNIT_LENGTH]

4. Select the task Manage Item Classes.
5. From the Manage Item Classes page, edit the item class that you chose as the Association Name for your rule set.
6. On the Item Management tab of the Edit Item Class page, choose the settings shown in the following table, and save the item class.

Field (in Item Class)	Value
Name	Example: MyItemClass1
Item Number Generation Method (in Number Generation section)	Rule Generated
Associated Rule Set	Example: MyItemNumGenRuleSet1 **Note** Only rule sets with the corresponding Association Type (in this scenario, Item Class) are available. Also, your rule set will not appear in the Associated Rule Set list of values unless a valid rule has been created in the rule set.

7. When you create a new item from this item class, the item number field contains a message that the number will be rule-generated. When you save (or submit) the item, your rule generates a number for it.

Creating Rules to Generate Descriptions for Items

This scenario illustrates how to use rules to generate descriptions for items.

Field (in Rule)	**Value**
Name	Example: MyItemDescGenRule1
Return Type	Item Description
Primary If Expression	True
Secondary If Expression	True
Return Value	Example: `[StyleItem].[Main].[Item Description] + ", " + [Item].[Top Variants].[Color] + ", " + [Item].[Top Variants].[Top Size]`

Field (in Item Class)	**Value**
Item Description Generation Method (in Description Generation section)	Rule Generated
Associated Rule Set	Example:MyItemDescGenRule1

- When you create a new item from this item class, your rule generates a description for it.
- When any attributes used in your description generation rules are updated, the item descriptions are regenerated accordingly.

  Note The same rule set can contain rules for both item number generation and item description generation.

User can use in-built functions or operations of external Web service in number and description generation rules.

Business Entity	Item
Rule Set Name	NumberGenerationRuleSet
Rule Set Type	Assignment
Association Type	Item Class
Rule Type	Item Number

Primary If Expression	`True`
`Then Expression`	
Secondary If Expression	`True`
Return Value	`"DC" + substring([Item].[Camera Specs].[Camera Brand], 0, 1) + substring([Item].[Camera Specs].[Camera Color], 0, 1) + [Item].[Camera Specs].[Megapixel] + [Item].[Camera Specs].[Optical Zoom X] + substring([Item].[Camera Specs].[Wifi], 0, 1)`

Creating Rules to Generate Numbers for Change Orders

This scenario illustrates how to use rules to generate change order numbers for change orders. The scenario is the same as the one for generating numbers for items, with the differences shown as follows.

- On the Edit Rule Set page, the differences for your rule set are shown in the following table.

Field (in Rule Set)	**Value**
Display Name	Example: MyCONumGenRuleSet1
Association Type	Change type
Association Name	Example: MyCOType1

- In the rule details section of the Edit Rule Set page, the differences for your rule are shown in the following table.

Field (in Rule)	**Value**
Name	Example: MyCONumGenRule1
Return Type	Change order number
Return Value	Example: `Flexfield[CM_Global_Segment1]`

- On the Number Generation tab of the Edit Change Order Type page, the differences for your change order type are shown in the following table.

Field (in Change Order Type)	**Value**
Number Generation Method (in Number Generation section)	Rule Generated
Associated Rule Set	Example:MyCONumGenRuleSet1

- When you create a new change order, your rule generates a change order number for it after you submit it.

Creating Rules to Generate Numbers for New Item Requests

This scenario illustrates how to use rules to generate new item request numbers for new item requests. The scenario is the same as the one for generating numbers for items, with the differences shown as follows.

- On the Edit Rule Set page, the differences for your rule set are shown in the following table.

Field (in Rule Set)	Value
Display Name	Example: MyNIRNumGenRuleSet1
Association Type	New Item Request
Association Name	Example: MyNIRType1

- In the rule details section of the Edit Rule Set page, the differences for your rule are shown in the following table.

Field (in Rule)	Value
Name	Example: MyNIRNumGenRule1
Return Type	New item request number
Return Value	Example: `Flexfield[CM_Global_Segment1]`

- On the Number Generation tab of the Manage New Item Request Type Details page, the differences for your new item request type are shown in the following table.

  Note There are multiple item classes and change order types, but only a single type for new item requests. Consequently, all new item requests use the same rule set.

Field (in New Item Request Type)	Value
Number Generation Method (in Number Generation section)	Rule Generated
Associated Rule Set	Example:MyNIRNumGenRuleSet1

- When you create a new item request, your rule generates a new item request number for it.

Component type rules

Component type rules are used to check if the components associated with the structure are valid. COMPONENT_TYPE () function takes the list of valid user item types for components as argument.
The following example illustrates how a rule can be written to ensure that only supported component types can be used with a specific structure. In this scenario, only components of item type - TYPE 1 or TYPE 2 can be added to a Primary structure.

Business Entity	Item
Rule Set Name	ValidateComponentType
Rule Set Type	Validation
Association Type	Item Class
Association Name	Root Item Class
Rule Name	ValidateComponent
Valid Component Rules (Check box)	Checked
If Expression	[Structure].[StructureAttributes].[STRUCTURE_NAME]==”primary”
Validation Condition	`COMPONENT_TYPE(“TYPE1”,”TYPE2”)`
User Message	`Component type not supported.`

Item class change

User can change the item class on an item and set it to a different item class. Validation rules can be triggered to either generate the action of warning, reject or needs approval based on the functions, **TO_ITEM_CLASS(“Item Class Name”)** and **FROM_ITEM_CLASS (“Item Class Name”)**

Business Entity	Item
Rule Set Name	ValidateItemClass
Rule Set Type	Validation
Association Type	Item class
Association Name	
Rule Name	ValidateSourceTargetIC
Severity	Reject
If Expression	TO_ITEM_CLASS("TCCHLDVR") and FROM_ITEM_CLASS("TCParent")
Validation Condition	`false`
User Message	`An item class cannot be changed from TCParent to TCCHLDVR.`

Supported Business Entities

When you reference an attribute in your rule expressions, you will have to specify the entity first.

For ex, [Item].[Main].[ITEM_TYPE]

The business entities supported are Item,

Supported business entities are - Item, Revision, CrossReference, RelatedItem, GTIN, StyleItem, Structure and Supplier.

In the earlier releases it was not allowed to mix attributes from different entities in a single.

Now, you can reference attributes from certain other entities while creating rules for an attribute belonging to an entity.

For example,

You can create validation constraints for a supplier level attribute referencing an item level attribute.

Reject, if [Supplier].[Promotion_Attributes].[Discount] > = 20%, if [Item].[Price_Attributes].[MRP] < 10$

Another example in which a rule that references attributes from Item and Cross reference relationship

Warning:

IF Expression: [Item].[Item_Basic].[ITEM_CLASS]=="Electronics" AND [CrossReference].[CrossReferenceMain].[Type]=="Old_Part_Number"

Validation Condition: !isnull([CrossReference].[CrossReferenceMain].[Value])

Thus all the org level attributes are available at the item level for a user to writes the rule.

e.g. [Inventory].[RESERVABLE_TYPE] and [Main].[INVENTORY_ITEM_STATUS_CODE] are org level attributes are available and following rule can be written with the item business entity selected –

GTIN Validation rules:

The Rules supports GTIN attributes in rule expressions. This can be used to enforce your own business validations with regards to GTIN association.

- Based on the specified criteria involving attributes/organizations, GTIN rules can check if a GTIN is associated to an item or check if there any change in the associated GTIN.
- GTIN rules can also validate the digits of an item's GTIN.

Ex. substring([GTIN].[GTIN Main].[GTIN], 4, 4) == "5"

StyleItem entity:

This entity can be used to access the attributes of the corresponding style item of SKU items.

If a rule contains the "StyleItem" entity then it will only be run if:

- The user modified a style item.
- The user created/modified a SKU item.
- The user created/modified an entity (e.g revision, supplier association, etc.) on a SKU item and the rule also uses that entity.

Expression Reference

Constants

For the purpose of rule expressions, all values are of the following simple types:

Type	**Description**
String	All strings and characters
Number	All decimals, integers, etc.
Date	Date only.
DateTime	Combination of Date and Time.
Time	Time only.

Null or Empty values

Any attribute can have an empty or “null” value. The only way to test for empty values is to use the “isnull” function (see 0).

Strings

Strings are delimited by double or single quotes

The escape character is “\”.

Numbers

Depending on user’s EBS preferences, numbers can have a “.” or “,” as decimal point. If “,” is used as a decimal point, then function arguments will use “;” as the argument delimiter.

Date, DateTime, Time

Dates will be entered in the format specified by the user in his “Preference Option”. Date constants will automatically be reformatted if a user logs in with a difference preference.

So the following expression is valid:

```
Item.Logistics.StartDate > “2005-Feb-23”
```

Days can be added to the date using numbers. For example, to compute the sell date as 30 days after the availability date:

```
Item.Logistics.SellDate > Logistics.AvailableDate + 30
```

There is currently no support for calculations involving Custom Calendars.

Attribute Expressions

Attribute Expressions

The way to access attributes is by appending it to its entity and attribute group (separated by periods).

```
[Entity].[Attribute Group Name].[Attribute Name]
```

Users need to use the internal name of the Attribute Group and Attribute. The internal name is guaranteed not to have any spaces in it.

UOM

For attributes that have a Unit of Measure (UOM) defined, it can be accessed by adding ".UOM" to the end of the expression:

```
[Entity].[Attribute Group Name].[Attribute Name].[UOM]
```

For example:

    PhysicalAttributes.Weight.UOM

might return "kg".

Operational Attributes have their UOM explicitly stored in an attribute. For these, we will still allow access to the attribute. For example, the Operational Attribute "PhysicalAttributes.Weight" has its UOM stored in the attribute "PhysicalAttributes.Weight_UOM_Code". So for this attribute we allow expressions like:

    Item.PhysicalAttributes.Weight_UOM_Code == "KG"

Note that all comparisons between amounts are automatically adjusted to take into account different UOMs. So accessing the UOM should not be necessary for comparison purposes.

Item Class

To make a Rule Set specific to an Item Class, assign it to that Item Class during Rule Set creation. That Rule Set will then be active for each entity that has that Item Class as a parent or as an ancestor.

Currently there is no way to access any other attributes associated with an Item Class.

Item Class Change Function

TO_ITEM_CLASS("Item Class Name")
FROM_ITEM_CLASS ("Item Class Name")

e.g. The condition ,
TO_ITEM_CLASS("TCCHLDVR") and FROM_ITEM_CLASS("TCParent") checks that the new item class is 'TCCHLDVR' and the original item class is 'TCParent'.

Multi-Row Attribute Groups

In earlier releases, Multi-row Attribute Group expressions cannot include attributes from any other attribute group. They are restricted to only using attributes from that Multi-row Attribute Group. We only process the modified rows of a Multi-row attribute group.

This restriction is removed. You can write rules to reference a value in particular row of a multi-row attribute group.

loop_sum function

The loop_sum function will take one numeric sub-expression as an argument. It will run the sub-expression for each multi-row row and compute the sum of the results.
Example,
Reject,
If (loop_sum([Item].[Composition].[Percentage])) != 100.
In this example, the sum of the values in all the rows of Percentage attribute cannot be a value other than 100.

conditional_loop_sum

The conditional_loop_sum function will take two sub-expression arguments. The first argument must be a boolean expression and the second argument will be a numeric sub-expression. It will run the boolean sub-expression for each multi-row row and, if the boolean evaluates to true, will compute the numeric sub-expression for that row. The function will return the sum of the computed numeric expressions.
For example,

Mutl-row attribute group = Forecast:

Customer	**Location**	**Reqd_Qty**
ABC	Seattle	20
XYZ	Seattle	30
ABC	Boston	25

conditional_loop_sum([Item].[Forecast].[Location] == “Seattle”, [Item].[Forecast].[Reqd_Qty])

This will sum up the values of Reqd_Qty for which the corresponding Location is Seattle

Entity Flexfields

Entity flexfields do not belong to any attribute group, and are accessed using the FlexField keyword:

Flexfield[<Flexfield Internal Name>]

Example:

Flexfield[DiagonalLength]

Expressions

null Values are Ignored

In a typical business process, not all attribute values are entered at the same time. Multiple people will work on entering data for an item over some period of time. For example, the Production Engineer will enter the "Weight", while Marketing Manager will enter the "Description". To facilitate this form of processing, rules that reference attributes that have no value (also called a "null" value) are ignored. That means the expression is neither TRUE nor FALSE, but is treated like it did not exist. In other words:

Expressions that evaluate to NULL are ignored.

That means that you never need to code Validations like:

```
        if (not isnull(Item.PhysicalAttributes.Weight)) then Item.PhysicalAttributes.Weight <=
10
```

The "if" part is redundant, since if Weight was NULL the validation would get ignored anyway.

If you actually want to check that an attribute has a non null value, you can use the "isnull" function (see 0). It can be used to check that an attribute actually has a value entered.

Boolean Expressions

Boolean expressions return TRUE, FALSE or null.

These expressions can be used to be used in "If" expressions or in validations.

General Comparison Operators: == != < <= > >=

Syntax: *expression1 == expression2*
expression1 != expression2
expression1 < expression2
expression1 <= expression2
expression1 > expression2
expression1 >= expression2

If one or both expressions are null, then result is null. To check for null values you have to use "isnull" function.

String comparison is case INSENSITIVE. For case sensitive comparison use String "compare" function.

Null Comparison: isnull

Syntax: **isnull(** *expression*)

The function "isnull" returns TRUE if its argument is a NULL value, FALSE otherwise.

A	not(A)
null	T
not null	F

This function let's you explicitly test whether a value is null. Unlike other functions it will not be ignored if the value of the argument is null.

String Search: compare contains endsWith match startsWith

Syntax: **compare**(*string1*, *string2*)
contains*(look_for_string, look_in_string)*
endsWith*(look_for_string, look_in_string)*
match*(pattern, look_in_string)*
startsWith*(look_for_string, look_in_string)*

All string search functions are case SENSITIVE. If you want to do a case insensitive comparison use "==" comparison operator. **compare** returns 0 when string1 is exactly equal to string2, -1 if string1 is lexicographically less than string2, +1 if string1 is lexicographically greater than string2.

"**match**" does a regular expression search on a string.

See Section 0 on how to construct regular expressions for the match operator.

Logical Operators: and or not

Syntax: *expression1* and *expression2*
expression1 or *expression2*
not expression1

The logical "and" function implements the following truth table:

A	B	A and B
F	F	F
F	T	F
T	F	F
T	T	T
F	null	F
Null	*	null

The processor stops after it finds the first FALSE. Hence there is an asymmetry between "F and null" and "null and F".

The logical "or" function implements the following truth table:

A	B	A or B
F	F	F
F	T	T
T	F	T
T	T	T
T	null	T
Null	*	null

The processor stops after it finds the first TRUE. Hence there is an asymmetry between "T and null" and "null and T".

The logical "not" function implements the following truth table:

A	not(A)
F	T
T	F
Null	null

Comparison to Current Production: changed delta percent previous

For tolerance rules we often want to compare the new value of an attribute to the "current production" value.

Syntax:	**changed**(*attribute)*
	changed(*attributeGroup)*
	delta*(attribute)*

percent(*attribute*)
previous(*attribute)*

changed will return TRUE if the current value of attribute differs from the current production value, FALSE otherwise. This function will work with null values. If only the Attribute Group is specified, then the function will return TRUE if any attribute in that Attribute Group has changed.

delta will return the difference between new and old. String comparisons will be case-insensitive. For Booleans, TRUE is considered greater than FALSE.

Operator	**Number**	**Date, DateTime**
new < curr. prod.	new - curr. prod.	new - curr. prod.
new > curr. prod	new - curr. prod.	new - curr. prod.
new == curr. prod	0	0
curr. prod value does not exist	null	null
both new and curr. prod are null	0	0
either new or curr. prod are null (but not both)	null	null

percent is only valid for numbers and will return:

```
(delta(attribute) / "current production value" ) * 100
```

To access the actual previous value, use **previous.**

assignedTo Functions

There are two types of assignedTo functions;
assignedtoOrg("OrgCode"): this validates if an item is assigned to the specified organization and returns either TRUE or FALSE.
assignedtoCatalog(Catalog[Catalog Name].Category[Category Name]): this validates if an item is assigned to the specified Catalog/Category and returns either TRUE or FALSE.

Exists Function

Syntax: exists(<boolean expression>)
This method will loop through the rows of the entities used in the boolean expression and return true if the expression is satisfied for any of the rows.

For example, the user could write the following:
exists(Structure.[StructureAttributes].[STRUCTURE_NAME] == "ManufacturingBOM")
This would loop through all the structures on an item and return true if the any structure had name "ManufacturingBOM".

This function can be used to validate whether a particular relationship exists for an item:
Example: exists(relateditem.relateditemMain.Type == ”Up-sell”

Also, this can be used to verify if there is a row existing in a multi-row attribute group;
Example: exists(isNull(item.ingredients.ingredient_name) == false)

Number Operators

Math: + - * / sum

Syntax: *expression1* + *expression2*
expression1 - *expression2*
expression1 * *expression2*
expression1 / *expression2*
sum(expression1, expression2, …)

Perform regular mathematical expressions. It will return null if any argument is null.

Division by 0 will return null.

The number of decimal digits returned by division is the maximum number of digits from expression1 and expression2.

sum can be used to add up a series of values.

abs

Syntax: abs (*expression)*

Returns the absolute value of an expression.

Example:

abs(percent([Item].[PhysicalAttributes].[Weight])) <= 10

i.e. the percentage weight change has to be less than 10.

amount

Syntax: amount (*expression, target UOM)*

Returns an amount in the given UOM. This ensures that comparisons or calculations happen in the desired UOM. Example:

[Item].[PhysicalAttributes].[Weight] <= amount(10, 'kg')

i.e. the weight has to be less than or equal to 10 kg.

min, max

Syntax: min(*expression1, expression2, ...)*

Returns the minimum or maximum value of a series of values. Can also be used for arrays or in query expressions.

Example:
```
MAX([Item].[Physical Attributes].[Length],10)
```

round, roundup, rounddown

Syntax: round(*expression, decimals)*
roundup(*expression, decimals)*
rounddown(*expression, decimals)*

Rounds a number to the specified decimal places. "**round**" will round to the nearest value. "**roundup**" will round away from zero, "**rounddown**" will round towards zero.

round(1.5758, 2)	1.58
roundup(1.5758, 2)	1.58
rounddown(1.5758, 2)	1.57

String Functions

See also 0 for String comparison functions.

+

Syntax: *expression1 + expression2*

Concatenate two expressions and return the resulting string. Note that this will also return a valid string if the expressions are of other data types.

indexOf

Syntax: **indexOf***(look_for_string, look_in_string)*

Return position of “look_for_string” in “look_in_string”. String position starts at 0. If not found then return -1.

Search is case sensitive.

If either expression is null the resultant value is null.

length

Syntax: **length***(expression)*

Return the length of the given string.

If expression is null the resultant value is null.

lowercase

Syntax: **lowercase***(expression)*

Return the lowercase equivalent of the given expression.

If expression is null the resultant value is null.

substring

Syntax: **substring***(string, start)*
substring*(string, start, end)*

Return substring of “string” starting at “start” and ending before “end”. If “end” is omitted then return to end of string. String position starts at 0.

If “start” is less than 0 then start at the beginning of the string.

If “end” is greater than length of string then return up to the end of the string.

If any expression is null the resultant value is null.

trim

Syntax: **trim***(expression)*

Remove all leading and trailing (but not middle) whitespace characters from a string.

If expression is null the resultant value is null.

uppercase

Syntax: **uppercase***(expression)*

Return the uppercase equivalent of the given expression.

If expression is null the resultant value is null.

Date Functions

See also 0 for String comparison functions.

+ -

Syntax: *expression1 + expression2*
expression1 - expression2

Add or subtract certain number of days from a date. A single number will be interpreted as days. Else use ISO 8601 Format. Examples:

Item.Logistics.LeadTime + 3	: 3 days after the Item Lead Time.

IN Function

This function works both for String and Number; Returns either TRUE or FALSE
Syntax: IN (Expression, String1/Number1, String2/Number2,…)
Example: IN("RED",[Item].[BODY_ATTR].[COLOR], [Item].[COVER_ATTR].[COLOR])

NOT_IN Function

This function works both for String and Number; returns either TRUE or FALSE.
Syntax: NOT_IN (Expression, String1/Number1, String2/Number2,…)

Note: If you pass in a number as the first argument of IN or NOT_IN, all the other arguments will be treated as numbers. Likewise, passing in a String as the first expression will cause remaining arguments be treated as strings.

BETWEEN function

Returns a logical value indicating whether the specified value falls within a range.
Syntax: BETWEEN(value, min, max)

TO_NUMBER Function

Converts the string argument to a return value of the NUMBER datatype.
Syntax: TO_NUMBER(String str)
Example: TO_NUMBER(String [Item].[PHYSICAL_PROP].[COUNT])

DECODE Function

Search for an expression's values and then evaluate them in terms of a specified result.
Syntax: DECODE(expression, search1, result1, [search2, result2, ...], [default])
Example:
DECODE ([Item].[BODY_ATTR].[COLOR], "RED", "RED_COLOR","BLUE", "BLUE COLOR", NONE)==”NONE”

Note:

- The type of the "expression" argument will determine the expected types of the "search" arguments.
- The type of the "result" and "default" arguments will be based on the type of the "result1" argument.
- The return type is based on the type of the "result1" argument. For example, if you pass in a numeric value for the result1 argument then the function will return a Number.

AUTO_SEQUENCE Function

Syntax: AUTO_SEQUENCE("Sequence_Name", Starting #, Increment by)
This function looks for the “sequence_name” in the tables, if the sequence exists, then the function returns the next value from the sequence; if the sequence does not exist, it creates a sequence and returns the starting number. This function can be used in assignment as well as validation rules.
Example: AUTO_SEQUENCE("EDC Number", 1000, 1)

Custom Functions

If you want to use your custom functions in the Rule expressions, you can do so. Refer to the following examples that show how you can use your SQL and Java functions in Rules.

Custom SQL Function

Assume that you want to use a function "AGGREGATE_VARS" in the rule expression. For simplicity reason, assume that this function performs an addition of 2 input variables and returns the sum of the variables.

STEP 1 – CREATE SQL FUNCTION:

```
create or replace package PIM_SQL_RULE_FUNCS as

function AGGREGATE_VARS(var1      NUMBER,     var2      NUMBER)
return NUMBER;

end PIM_SQL_RULE_FUNCS;


create or replace package body PIM_SQL_RULE_FUNCS as
function AGGREGATE_VARS(var1      NUMBER,    var2      NUMBER)
return NUMBER
IS
   returnVar NUMBER;
begin
   returnVar := var1 + var2;
   return returnVar;
end AGGREGATE_VARS;
end PIM_SQL_RULE_FUNCS;
```

STEP 2 – GRANT PRIVILEGE ON FUSION_RUNTIME USER:

```
grant execute on PIM_SQL_RULE_FUNCS to fusion_runtime;
```

STEP 3 – USE THE FUNCTION IN PIM RULE:

Now you use this function in your validation rule to validate if the return value is greater than 20. The input variables are the item attributes Attr1 and Attr2 belonging to Item attribute group AttrGrp.

```
PIM_SQL_RULE_FUNCS.AGGREGATE_VARS([Item].[AttrGrp].[Attr1], [Item].
[AttrGrp].[Attr2]) > 20
```

(You can also use in assignment rules as well)

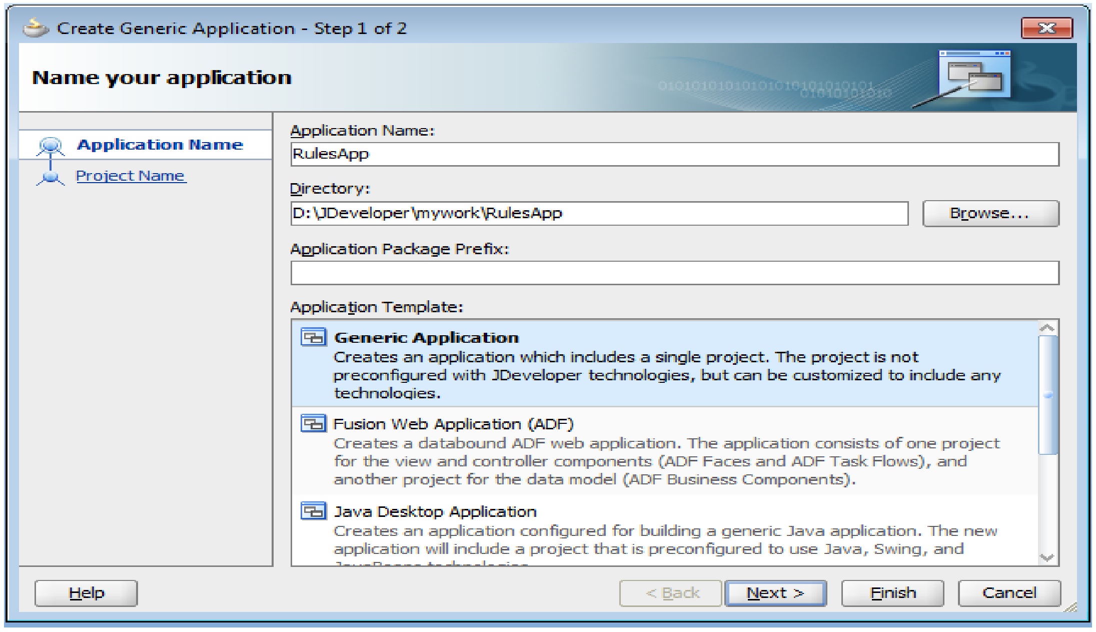
Custom JAVA Function

If you want to use custom Java functions in Rule expressions, you should follow the below steps:

STEP 1 – SETUP THE PROJECT TO DEVELOP THE CUSTOM CLASS

PIM Rules assumes that the custom java classes follow the same packaging standard as the Fusion Applications. The steps mentioned here assume that end user uses Jdeveloper 11g to build the custom java classes. The following simple steps are to be followed to create a work space and a project.

1. Open Jdeveloper 11g. Create a new generic application. Click on next.



Application Name: RulesApp

Directory: D:\JDeveloper\mywork\RulesApp

Application Package Prefix:

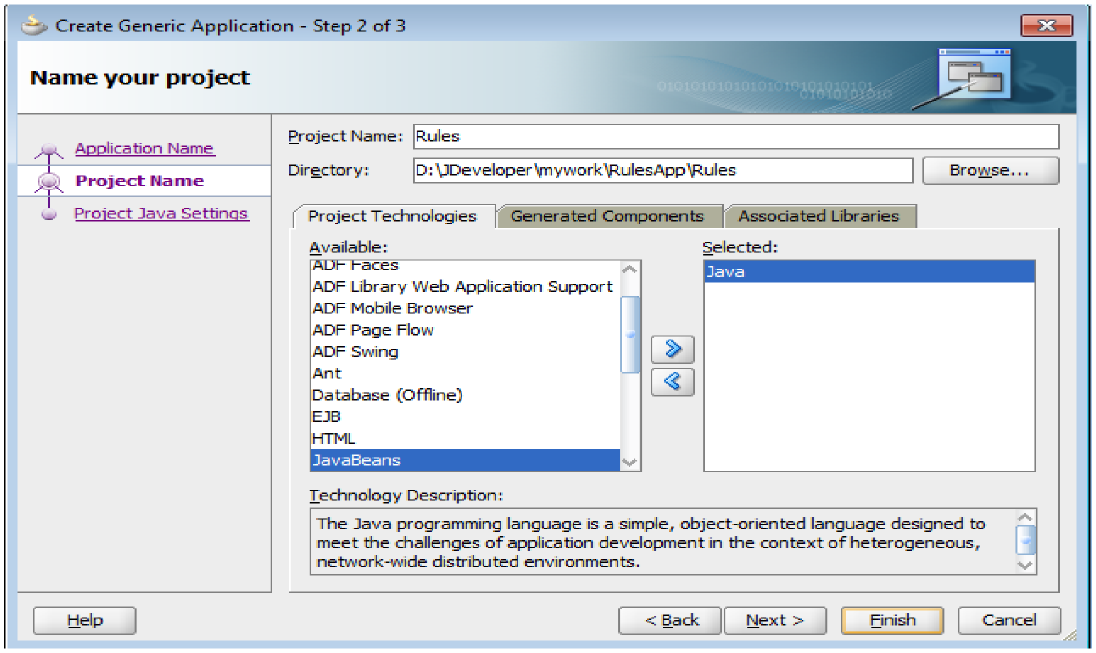
Application Template:

Generic Application
Creates an application which includes a single project. The project is not preconfigured with JDeveloper technologies, but can be customized to include any technologies.

Fusion Web Application (ADF)
Creates a databound ADF web application. The application consists of one project for the view and controller components (ADF Faces and ADF Task Flows), and another project for the data model (ADF Business Components).

Java Desktop Application
Creates an application configured for building a generic Java application. The new application will include a project that is preconfigured to use Java, Swing, and

2. Choose “Java” from project technologies shuttle. Give a project name. Click on Finish.


Create Generic Application - Step 2 of 3

Name your project

Application Name
Project Name
Project Java Settings

Project Name: Rules

Directory: D:\JDeveloper\mywork\RulesApp\Rules

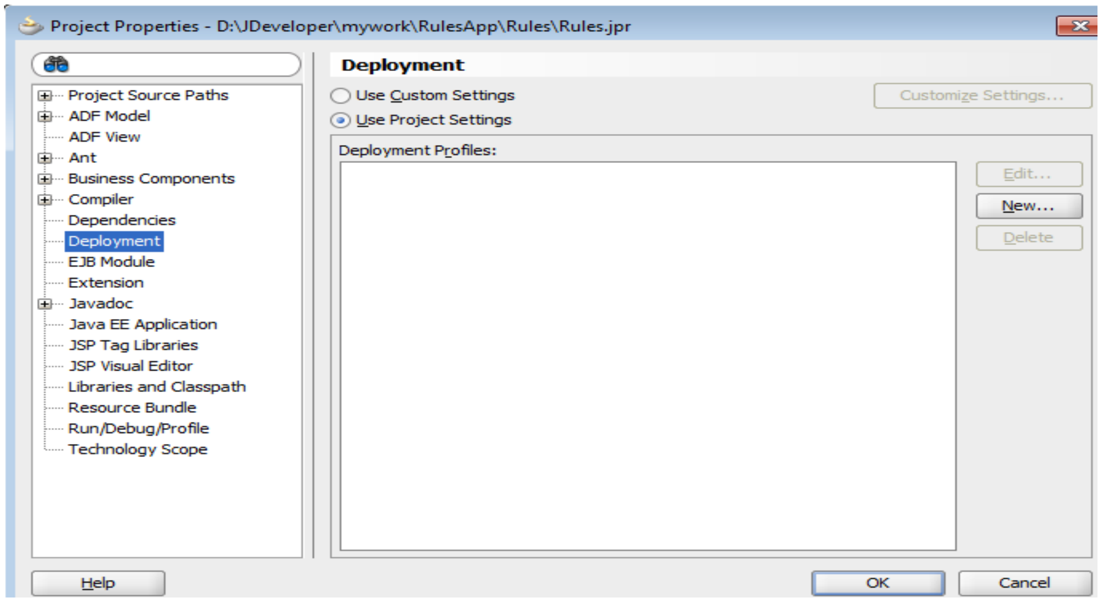
Project Technologies | Generated Components | Associated Libraries

Available:
ADF Faces
ADF Library Web Application Support
ADF Mobile Browser
ADF Page Flow
ADF Swing
Ant
Database (Offline)
EJB
HTML
JavaBeans

Selected:
Java

Technology Description:
The Java programming language is a simple, object-oriented language designed to meet the challenges of application development in the context of heterogeneous, network-wide distributed environments.

3. The Application and project are created. Right click on project and choose "project properties". A window comes up showing the properties. Select the Deployment option.


Project Properties - D:\JDeveloper\mywork\RulesApp\Rules\Rules.jpr

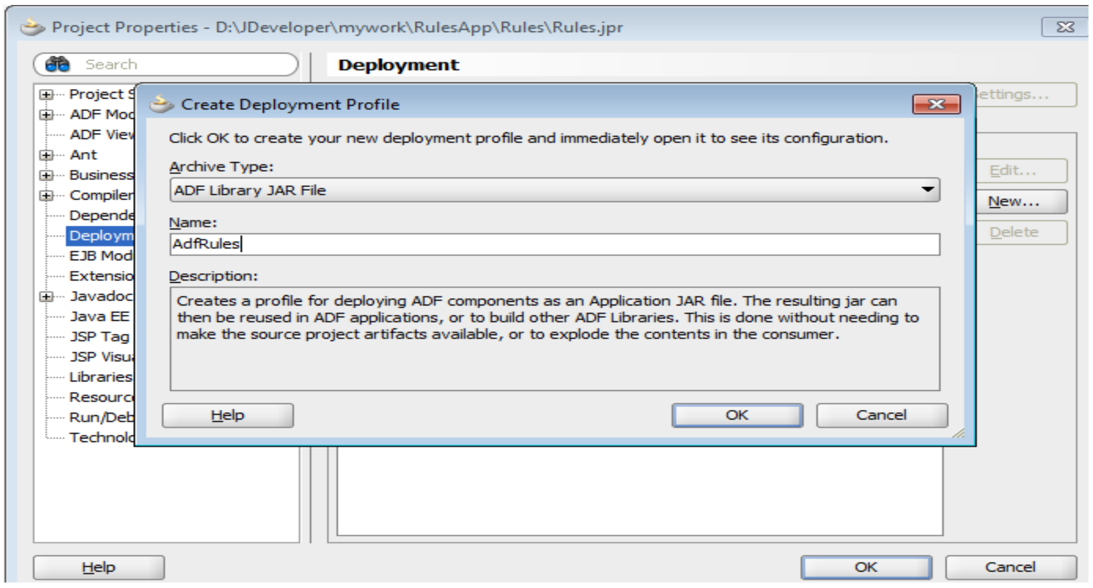
Project Source Paths
ADF Model
ADF View
Ant
Business Components
Compiler
Dependencies
Deployment
EJB Module
Extension
Javadoc
Java EE Application
JSP Tag Libraries
JSP Visual Editor
Libraries and Classpath
Resource Bundle
Run/Debug/Profile
Technology Scope

Deployment

Use Custom Settings [✓ Use Project Settings]

Deployment Profiles:

4. Create new deployment profile. Select the Archive Type as "ADF Library JAR File". Give a jar name and ok.


Project Properties - D:\JDeveloper\mywork\RulesApp\Rules\Rules.jpr

Create Deployment Profile

Click OK to create your new deployment profile and immediately open it to see its configuration.

Archive Type: ADF Library JAR File

Name: AdfRules

Description:

Creates a profile for deploying ADF components as an Application JAR file. The resulting jar can then be reused in ADF applications, or to build other ADF Libraries. This is done without needing to make the source project artifacts available, or to explode the contents in the consumer.

5. You can optionally set a default package for the project also. Save all the work

Now the project is setup to write java class which can hold custom java rule.

Step 2 – Writing a Java class and custom function

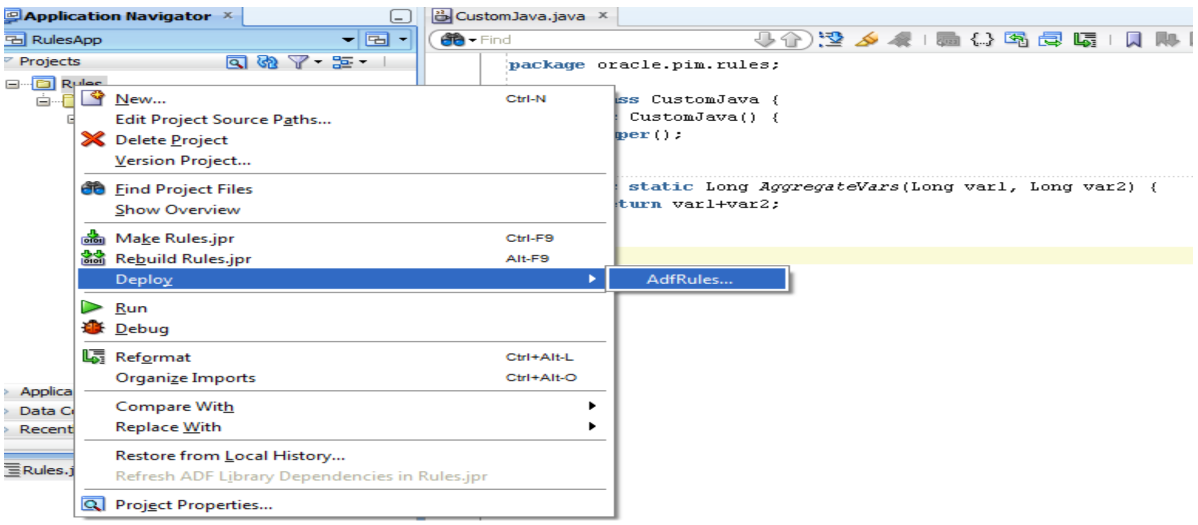
In the project created in step 1, user can create a new java class which holds the custom function. The custom function should be a static function and should return a value. A sample code for a custom class is given below.

```
package oracle.pim.rules;


public class CustomJava {
   public CustomJava() {
      super();
   }


   public static Long AggregateVars(Long var1, Long var2) {
      return var1+var2;
   }
}
```

Once the class is ready, build and deploy the jar file. Use the deployment profile created in Step-1.



```
package oracle.pim.rules;

ss CustomJava {
: CustomJava() {
per();

: static Long AggregateVars(Long var1, Long var2) {
turn var1+var2;
```

You can find the location of the deployed jar from the deployment profile in project properties. Pick up the jar file for next step.

STEP 3 – PARK THE JAR FILE IN FUSION APPS DEPLOYMENT CLASS PATHS

Open a session on the box where the Fusion Applications are deployed. Locate the PIM and PIM Common applications. They can be found under location:

$APPL_TOP/fusionapps/applications/scm/deploy/EarScmPim.ear

$APPL_TOP/fusionapps/applications/scm/deploy/EarScmPimCommon.ear

Go into APP-INF/lib location of each of these EARS. Park the jar file containing the custom java class in these locations. Bounce the Product Management and Product Management Common servers. Now we are ready to use the custom java functions in rules.

STEP 4 – REFERRING THE CUSTOM JAVA FUNCTION IN PIM RULES

The custom java function can be referred as the same way we refer a static function in java. Here is a sample rule expression:

oracle.pim.rules.CustomJava.AggregateVars([Item].[AttrGrp].[Attr1], [Item].[AttrGrp].[Attr2])>10

Calling Web service from within rules

There is an alternate way to use custom functions in Rule expressions.
You could write a program/function and create a public web service by including the function.
You can reference this service/operation in rule expression. In the run-time, when the rule is executed, the web service is called and the value returned by the service is obtained.

First you have to register the web service along with the credentials used to invoke it.
Use the function InvokeWebService ( ) in rule expression to reference the registered web service.

Syntax:
InvokeWebService("Name_of_WS_Connection", "Method_name", input_argument1, input_argument2,…….)

When the rule is executed, the above function invokes the web service, passes input payload, and receives a value returned by the service. The returned value can be used for assignment or validation.

Referencing custom object data in rules

You may model certain non-standard information as custom objects and store the relevant data. For example, you may maintain a table that contains a matrix of selling restrictions by target market. And, you want to refer the data to determine whether the item is sellable or not based on the target market.
Custom Object: Selling_Restrictions

Target Market	**Restriction**
EU	No
North America	Yes
APAC	Yes
Middle East	No

You can create a rule to fetch values from custom object and perform assignment or validation.
<u>The syntax:</u>
getCustomObjectValue(Custom Object Name, Custom Object Return Attribute Name, Custom Object Query Attribute Name1, Value1,..... Custom Object Query Attribute NameN, ValueN)

where,
Custom Object Name = The label of Custom Object from which data is to be fetched
Custom Object Return Attribute Name = The label of the Custom Object column whose value is to be fetched
Custom Object Query Attribute Name1 = The label of the Custom Object column whose value is to be matched (==) with Value1
Value 1 = It can be a hard coded value or can refer to an attribute, ex, [ITEM].[AG1].[A1]

So, in the above example, you can use create rules to conditionally fetch value from 'Restriction' and use it for assignment or validation

Analyze the impact of Rules

Any new rule that you create will be executed only on any new items or modified items; these are not really applied to the already existing items in the system. Analyze item rule set impact is a way to study the impact of the new rules on existing set of items. Before you productionize a newly created rule you can perform a "what-if" analysis on existing items using analyze item rule set impact.
When you create a new rule, mark it as draft and include it in Master rule set; Draft rule sets will not be evaluated during regular production transactions.
When you perform Impact analysis, draft rules along with other rules, are applied on the given set of items and results are presented. You can review the results; and once you are satisfied with

the results, you can productionize the changes by removing draft flag and performing mass update actions supported post-impact analysis.

While defining the scope of items on which the analysis has to be performed, you can specify the scope in the form of fixed list of items or in the form of saved searches. For example, you have specified a saved search that contains criteria that Item class is equal to Fasteners AND Item status is Active. During execution of the analysis, query will run and get all the items that match the specified criteria at that point of time. Analysis will be performed on this set of items.

Automating Impact Analysis and Update

You can schedule the impact analysis followed by automatic update. You can indicate whether an analysis should automatically trigger import and specify a schedule for the analysis run. For example, you have enabled auto-update for an analysis and specified a schedule of daily run at 10 am from D1 to D7. On the scheduled date and time, Impact analysis process will run and all the applicable non-draft rules will be applied on scope items; it will also trigger import job automatically which will apply the applicable rules on production data.

Supported Attributes

All User Defined Attributes (EFFs) can be accessed using the <Entity>.<Attribute Group Name>.<Attribute> syntax. In addition, rules will support access to certain pre-defined attributes; we use pre-defined Attributes Groups to access the attributes.

Validation of Item Revisions is not supported at this time.

Rule Set Context

This is a special set of attributes that gives some context to the rule execution.

Context Attributes	**Description**
RuleSetVersion	Currently always set to “1.0”
ExecutionDate	Date for which rules are invoked.
ExecutionDateTime	Date and Time for which rules are invoked.
BatchID	Set to Batch ID if we are running Import.
BatchName	Set to Batch Name if we are running Import.

Example:

Context.ExectionDate >= “7/6/2007”

Item Primary Attributes

Attribute Group Name : Main

Item Primary Attributes
DESCRIPTION
LONG_DESCRIPTION
INVENTORY_ITEM_STATUS_CODE
PRIMARY_UOM_CODE
ITEM_TYPE
CURRENT_PHASE_CODE
DUAL_UOM_CONTROL
ALLOWED_UNITS_LOOKUP_CODE
SECONDARY_UOM_CODE
DUAL_UOM_DEVIATION_HIGH
DUAL_UOM_DEVIATION_LOW
TRACKING_QUANTITY_IND
TRADE_ITEM_DESCRIPTOR
HTML_LONG_DESCRIPTION
ONT_PRICING_QTY_SOURCE
SECONDARY_DEFAULT_IND

Item Basic Attributes

Attribute Group Name : Item_Basic

Item Primary Attributes
Name
ITEM_CLASS
ORGANIZATION_CODE
STYLE_ITEM_FLAG
APRPOVAL_STATUS

Remember to specify internal names/values in your rule expression.
For example, For example, if you want to assign a value for an attribute based on item's Approval_Status as Approved, you should write an IF expression as below: [Item].[Item_Basic].[APPROVAL_STATUS]) == "A". The other approval statuses and their respective internal values are as below:

Value	**Meaning**
A	Approved
D	Draft
N	Not submitted for approval
R	Rejected

S	Submitted for approval
SCH	Scheduled

Structure Attributes

Structure attributes will be accessed using “StructureAttributes” attribute group name only in the context of Item entity.

Attribute Group Name: StructureAttributes

StructureAttributes
StructureName
OraganizationCode
CommonItemName
CommonOrganizationCode
CommonStructureName

Item Supplier Site Organization Intersection Attributes

Attribute Group: Intersection_Primary:

Item Supplier Site Organization Intersection Attributes
Primary_Flag
Status

Example: Intersection_Primary.Primary_Flag

Supplier Attributes

Supplier Attributes will be accessed using the “Supplier” group. Example:

Supplier.Supplier_Name == “Acme”

Supplier Attribute
Supplier_Name
Supplier_Number
DUNS
Taxpayer_ID
Tax_Registration_Number

Supplier Site

Supplier Site Attributes will be accessed using the “SupplierSite” group. Example:

SupplierSite.State == “CA”

Supplier Site Attributes
Supplier_Site_Name
City
State
Country

Item Relationship Attribute Groups

<u>Cross Reference</u>
Attribute group: CrossReferenceMain

Attributes
Type
Description
Value
ApplicableOrganization

<u>Related Item</u>
Attribute group: RelatedItemMain

Attributes
Type
Description
StartDate
EndDate
Reciprocal
PlanningEnabled

Execution Behavior

When Rules are Run

Rules are only run for attributes that have changed.

Validation with “Reject” Severity

If a validation that has severity equal to “Reject” fails, then the whole Entity is rejected.

Change Management

Change Order Required

For an attribute to require a Change Order, You can create a validation (rule) for that attribute with severity as Needs Approval.

Related Attributes

We also make sure that all *related* attributes are also requiring Change Order. “*Related”* means All updated attributes used in any validation that uses an attribute that requires Change Order.

In other words, if any attribute requires a Change Order, then all the updated attributes in that Validation rule (from “If” and “Then” Expressions) will also be required to be part of the same Change Order.

Change Order-required dependencies

If the attributes computed in assignment rules are used in subsequent rules, then they can form a chain of dependencies. In order to ensure that the data remains consistent we want to propagate the Change Order requirement along the dependency chain.

Note that Propagation of the Change Order requirement only propagates along *updated* attributes. If an attribute is not updated, then it should not affect other attributes.

Glossary

Alternate Catalog	Alternate Taxonomies that do not affect attribution (i.e. they do not specify Attribute Groups for each category). See also “Item Class”.
Item Class	The taxonomy that specifies which Attribute Groups is valid. See also “Alternate Catalog”.

Regular Expression Syntax

Characters

x	The character x
\\	The backslash character
\0n	The character with octal value 0n (0 <= n <= 7)
\0nn	The character with octal value 0nn (0 <= n <= 7)
\0mnn	The character with octal value 0mnn (0 <= m <= 3, 0 <= n <= 7)
\xhh	The character with hexadecimal value 0xhh
\uhhhh	The character with hexadecimal value 0xhhhh
\t	The tab character ('\u0009')
\n	The newline (line feed) character ('\u000A')
\r	The carriage-return character ('\u000D')
\f	The form-feed character ('\u000C')
\a	The alert (bell) character ('\u0007')
\e	The escape character ('\u001B')
\cx	The control character corresponding to x

Character classes

[abc]	a, b, or c (simple class)
[^abc]	Any character except a, b, or c (negation)
[a-zA-Z]	a through z or A through Z, inclusive (range)
[a-d[m-p]]	a through d, or m through p: [a-dm-p] (union)
[a-z&&[def]]	d, e, or f (intersection)
[a-z&&[^bc]]	a through z, except for b and c: [ad-z] (subtraction)
[a-z&&[^m-p]]	a through z, and not m through p: [a-lq-z](subtraction)

Predefined character classes

.	Any character (may or may not match line terminators)
\d	A digit: [0-9]
\D	A non-digit: [^0-9]
\s	A whitespace character: [\t\n\x0B\f\r]
\S	A non-whitespace character: [^\s]
\w	A word character: [a-zA-Z_0-9]
\W	A non-word character: [^\w]

Boundary matchers

^	The beginning of a line
$	The end of a line
\b	A word boundary
\B	A non-word boundary

\A	The beginning of the input
\G	The end of the previous match
\Z	The end of the input but for the final terminator, if any
\z	The end of the input

Greedy quantifiers

X?	X, once or not at all
X*	X, zero or more times
X+	X, one or more times
X{n}	X, exactly n times
X{n,}	X, at least n times
X{n,m}	X, at least n but not more than m times

Reluctant quantifiers

X??	X, once or not at all
X*?	X, zero or more times
X+?	X, one or more times
X{n}?	X, exactly n times
X{n,}?	X, at least n times
X{n,m}?	X, at least n but not more than m times

Possessive quantifiers

X?+	X, once or not at all
X*+	X, zero or more times
X++	X, one or more times
X{n}+	X, exactly n times
X{n,}+	X, at least n times
X{n,m}+	X, at least n but not more than m times

Logical operators

XY	X followed by Y
X|Y	Either X or Y
(X)	X, as a capturing group

Precedence

The precedence of character-class operators is as follows, from highest to lowest:

1 Literal escape \x
2 Grouping [...]
3 Range a-z
4 Union [a-e][i-u]
5 Intersection [a-z&&[aeiou]]

Note that a different set of metacharacters are in effect inside a character class than outside a character class. For instance, the regular expression . loses its special meaning inside a character class, while the expression - becomes a range forming metacharacter.

For a more precise description of the behavior of regular expression constructs, please see Mastering Regular Expressions, 2nd Edition, Jeffrey E. F. Friedl, O'Reilly and Associates, 2002.

Change Record

Date	**Description of Change**
11-Jun-2007	Created Document.
23-May-2013	Revised the contents to reflect the changes and enhancements made in and upto Fusion 1.0.6
06-May-2016	Revised the contents to reflects the changes and enhancements made in and upto Fusion 11.1.11.0.0

Oracle Corporation

Author and Date

Magesh Kesavapillai, 06-May-2016

Contributors

Name	**Contribution**
Raghunath Gadiyaram	Steps for using custom functions

Copyright Information

Disclaimer

Trademark Information