TIBCO® MDM Studio

Rulebase Designer User Guide

Version 6.1.1 | June 2024

Document Updated | October 2024

Contents

Overview

A repository consists of a list of records, each with its own set of attributes. Each attribute is defined as being of a particular type and having a certain length.

A repository rule allows you to specify more complex constraints on attributes. For example, a repository defines Price as a Number. Using a repository rule, you can specify that this number must be between 0 and 100. You can also specify that, if Price is not zero, another attribute (Currency) must have a specified value.

A repository rule is an encapsulated piece of business logic that specifies validations, transformations, and access controls for a record. Some examples are:

- The storage temperature must be between –20F and 80F.
- The product effective date must be before the product first ship date.
- The subclass code must be Chicken, Beef, or Vegetable when the product class code is Soup.
- Volume is calculated by multiplying height x depth x length.
- Only Supervisors can access records with Product Code equal to ‘HS’.

Repository rules are specified in Rulebase files. Two files can be defined for a repository:

- New record file — called to initialize a new record
- Validation file — called for existing records
- Search control rules file — called from record search.

For more information on the Rulebase files, see Establish a Rulebase File.

The Rulebase file of a repository rule consists of a header, variable declarations, and a constraint. The header gives the name and description of the rule. The constraint contains a *condition* and an *action*. The condition describes when the rule must be applied. The action describes what the rule actually does and controls which attributes the rule is applicable to. For more information, see Global Property Settings for Rules.

Rulebase Designer Overview

The TIBCO MDM Rulebase Designer provides an intuitive graphical user interface to help business users understand and design rule and rule flows for MDM.

The Rulebase Designer adds a visual element to designing rulebases and makes the process quicker and more intuitive.

The Rulebase Designer is based on TIBCO Business Studio and acts as an 'add on' component to TIBCO Business Studio. Rulebase models are stored in a **.rul** format, contained in a special folder called **Rulebase Models**.

What you can do with the Rulebase Designer

The Rulebase Designer provides an interface to:

- Graphically declare **Variables** and **Constraints**.
- Graphically define **If-Then-Else** conditions (flowchart representation).
- Graphically define **Actions** corresponding to **Then** or **Else** conditions.
- Define **Expressions** through an expression editor (for Conditions/Actions described by expressions and where clauses).
- Define **Expressions** with ease using the context sensitive help in the expression editor.
- **Drag and drop** semantics in the expression editor.
- **Check syntax** of rules at development time.
- **Import** existing rulebases.
- **Direct deployment** to TIBCO MDM using network deployment on the TIBCO MDM Server.

Establish a Rulebase File

Following rulebase files can be defined for a repository:

- New record file — called to initialize a new record

- Validation file — called for existing records
- Search control rules file — called from record search.

The new record file is called when adding a new record. Subsequently, the validation file is called when modifying the record. The names of these files are defined using the Configurator. The default names are as follows:

- **InitialConfig > Rule Base > New Record Data Population Rulebase File Name** = `newrecord.xml`
- **InitialConfig > Rule Base > Record Save Validation File Name** = `catalogvalidation.xml`
- **InitialConfig > Rule Base > Record Search Rules File Name** = `searchcontrolrules.xml`

> **Note:** Do not rename the default rulebase file, `catalogvalidation.xml`, generated by the system.

When a repository is created, these files are created in folder `$MQ_COMMON_DIR/<enterprise-internal-name>/catalog/master/<repository id>`.

The <repository id> can be obtained from the Repository List page.

These validation files are also supported for a relationship catalog. This catalog is created when any relationship attribute is defined for the relationship. The initialization or validation for relationship attributes can be defined in these files. The ID of relationship catalog can be obtained from the database by executing the following SQL:

```
select relationshipcatalogid from relationshipdefinition where
ownerid=<repository id> and name=<relationship name> and active='Y
```

For a list of supported actions for a relationship catalog, refer to the Relationship and Multi-value Attributes Vs Regular Attributes section of the *TIBCO MDM User Guide*.

The system looks for these files in the following order:

```
$MQ_COMMON_DIR/<enterprise-internal-name>/catalog/master/<repository id>
```

`$MQ_COMMON_DIR/standard/rulebase`

Note: The second location is checked only when no file is specified in the first location.

Refer to Chapter 1 of the *TIBCO Installation and Configuration Guide* for details on MQ_HOME and MQ_COMMON_DIR, under the environment variables section.

New Record File

A new record file is used to assign default values to the attributes of a new record. Assigned values must come from constants or function calls.

This file is also used for propagation of attribute values to child records.

Some rules from the Validation File also come into play for new records.

Note: Drop downs can be filled up only through a constraint in the `catalogvalidation.xml` file.

Validation File

The validation file usually contains the bulk of the rules. All kinds of rules are applicable:

- assignments
- validations
- propagations
- access controls

Search Control Rules File

The search control rules file is used to configure drop-down for attributes on the Record Search screen.

Global Property Settings for Rules

The TIBCO MDM stores some system wide configuration parameters for rules in the `ConfigValues.xml` file.

These properties are set using the Configurator.

LogFlag

This parameter produces rulebase execution logs in the `$MQ_COMMON_DIR/Temp` directory. It should be used for development or debugging only and should not be enabled in a production environment.

This parameter can be specified in **System Debugging >** RuleBase Debug Mode of the Configurator. The Rulebase debugging requires large amount of disk space. Below is an example of how to set it to true so a detailed stack trace can be obtained for rule base execution.

Log files start with "rb" and end with "`.xml`". Several files can be produced in one go; browse to locate the right file.

Log Files

Prefix	Description
`rbb1*.xml`	First pass of rulebase. Computes propagations.
`rbb2*.xml`	Propagations.
`rbb3*.xml`	Second pass of rulebase. This is the file you need for debugging validations and assignments.

Attribute Names Checking

Setting the **Configurator** > **Miscellaneous** > **Check Attribute Names** parameter to true halts the processing (throw an exception) if an attribute in the rulebase is not present in the repository.

If this parameter is set to false, an error is logged in error.log and a null value is returned for that attribute and normal processing continues.

This parameter is provided mainly for backward compatibility with previous versions. In past versions, attributes not found in the repository were ignored. Now, more stringent checking is present, which can cut down on "typo" type errors, but it also means a rulebase that references a deleted attribute will no longer work.

The recommended approach is to set this flag to true in the system where you need to use the rulebase so that the system will alert you immediately if a rulebase contains an invalid attribute reference.

Date Formats allowed in Rulebase

The following date formats are supported in rulebase:

- mm/dd/yyyy (default date format)
- dd-mon-yyyy
- mm/dd/yy
- ddmmyyyy
- yyyy-mm-dd
- yyyy/mm/dd
- dd-mm-yyyy
- dd/mm/yyyy
- dd-mm-yy

Getting Started

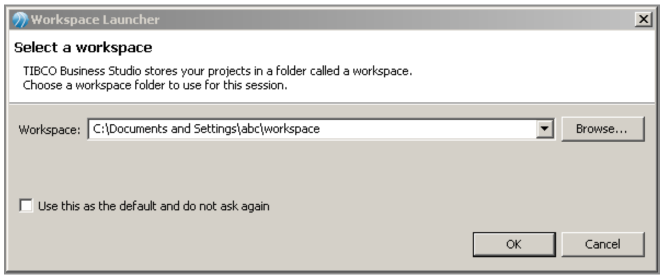
This chapter explains how to start the Rulebase Designer, what you will see at startup, information on accessing samples, tutorials, and help, and elements in the perspective.

Starting TIBCO MDM Rulebase Designer

Procedure

1. After the installation completes, start the TIBCO MDM Rulebase Designer by selecting **Start > Program Files > TIBCO > <environment name> >TIBCO Business Studio <ver> > TIBCO Business Studio.**
2. Provide a workspace location (folder where projects will be saved).

Result



TIBCO Business Studio stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Documents and Settings\abc\workspace

☐ Use this as the default and do not ask again

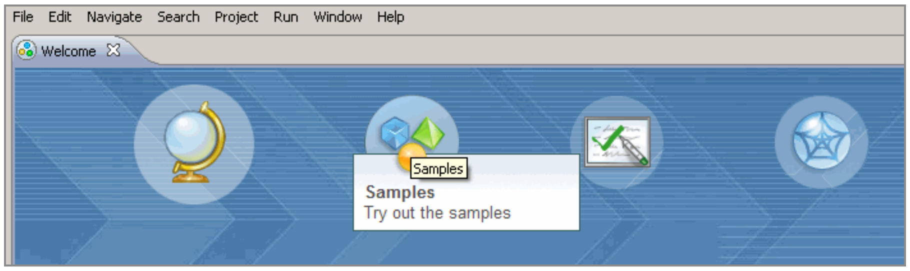
TIBCO Business Studio opens up and you are ready to start using it.

Welcome Screen

After you select the workspace for the first time, Eclipse opens up with the Welcome screen.

This contains icons to samples and tutorials among other things.

Note: This Welcome screen shows up only the first time and will not be displayed for subsequent openings of Eclipse. If you want to go to this screen again, you can access it from **Help > Welcome**.



Accessing Samples

TIBCO MDM Studio Samples are a collection of the TIBCO MDM standard processes, process modeling tutorials, repository data models, rulebase model, custom import project, TIBCO MDM Model templates, and Process java transitions.

The sample models are provided to illustrate the modeling capabilities of TIBCO MDM Studio. Each of these models needs further elaborations for their intended purpose. Install the sample projects to view the TIBCO MDM processes, data models, and their associated rules. All the samples are available in the TIBCOHome directory.

Follow these steps to install the **Sample**s.

Procedure

1. On the **File** menu, click **Import**.The import wizard is displayed.
2. From the **General** folder, select **Existing Studio Projects into Workspace**.
3. Click **Next**. The import wizard for selecting the directory path is displayed.
4. Click **Select archive file** option. Click **Browse** and select the sample project zip archives from \<TIBCOHome>\studio-mdm\<version>\samples folder.
5. Click **Finish**. The select project opens in the workspace.

Modelling Perspective

The Modelling Perspective consists of the following views:

- Project Explorer
- Diagram Editor
- Properties Tab
- Rulebase Data View
- Problems Tab
- Palette

Project Explorer

A hierarchical view of resources that lists all existing projects and files under projects.

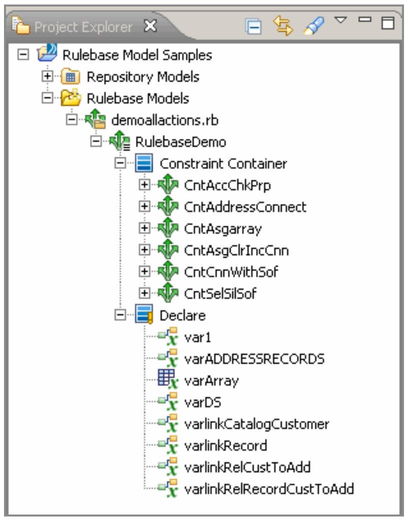


Diagram Editor

Enables graphical creation of rulebases. Consists of two sections:

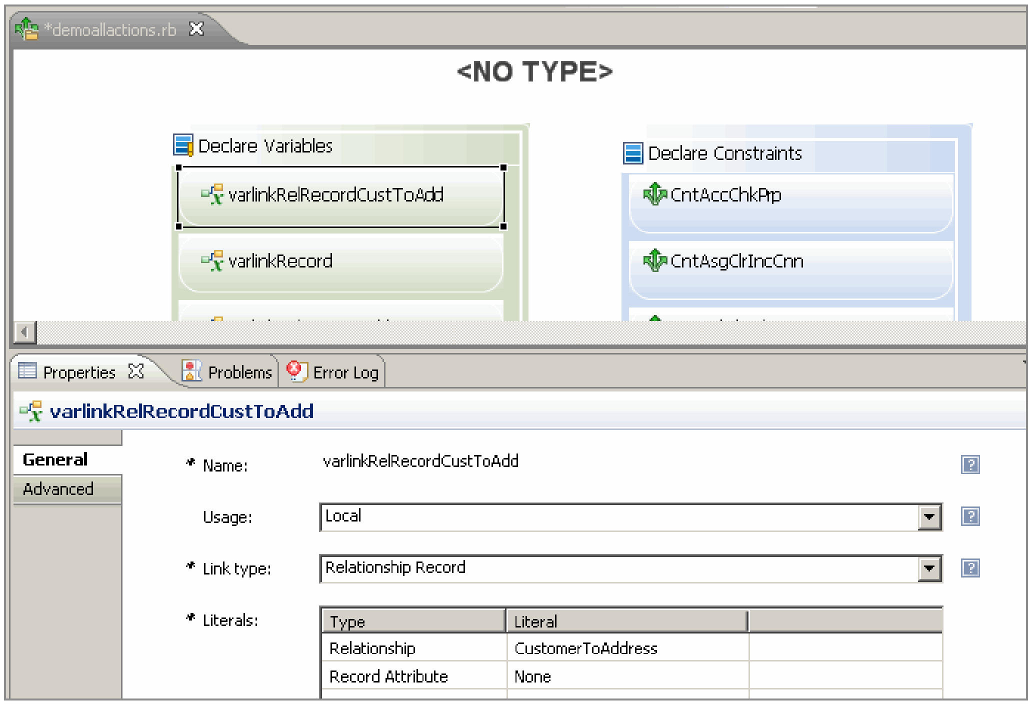
- The diagram editing canvas (to design and edit a rulebase diagram).

- The Palette (for selection of artifacts such as variables and constraints).

Properties Tab

This view allows for editing of properties of rulebase components in the diagram.

Editable and non-editable fields for the selected component in the canvas is displayed in the **Properties** Tab.



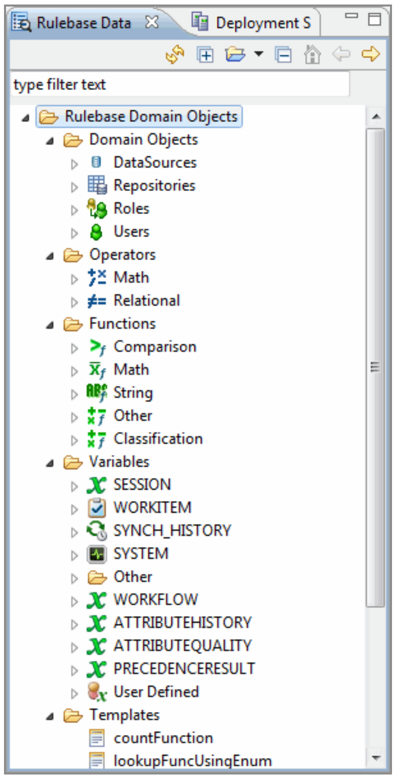
* Name:	varlinkRelRecordCustToAdd
Usage:	Local
* Link type:	Relationship Record

* Literals:

Type	Literal
Relationship	CustomerToAddress
Record Attribute	None

Rulebase Data View

This view lists TIBCO MDM artifacts imported through the Import Metadata artifacts.



Problems Tab

Displays errors (encountered in the current instance) to help diagnose problems with the Rulebase plug-in.

Palette

The Palette (to the right of the screen) contains different artifacts to help build a rulebase.

Select and drop into the main drawing pane to define or modify a rulebase model.

Main Palette

This is the main palette; it enables you to declare variables and constraints for your basic rulebase.

Declare Variables	
Relationship Data-type	Create a new data type variable.
Link-type	Create a new link type variable.
Declare Constraint	
Constraint	Create a new constraint.
Declare Decision Table	
DecisionTable	Create a new Decision Table.

Expression Editor Palette

This palette corresponds to constraints and provides icons to define conditions and actions
.

Actions	
Access	Create a new Access action.
ApplyPrecedence	Create a new ApplyPrecedence action.
Assign	Create a new Assign action.
Assign Identity	Create a new AssignIdentity action.
Categorize	Create a new Categorize action.
Check	Create a new Check action.

Clear	Create a new Clear action.
Connect	Create a new Connect action.
Disconnect	Create a new Disconnect action.
Include	Create a new Include action.
Propagate-Inline	Create a new Propagate Inline action.
Propagate-Rulebase	Create a new Propagate Rulebase action.
Select	Create a new Select action.
Slice	Create a new Slice action.
Softlink	Create a new Softlink action.
UnCategorize	Create a new UnCatgorize action.
Condition Expression	
Expression	Create new Expression.
Group Expression	Create new Group Expression.
AND	Create new AND Operator.
OR	Create new OR Operator.
NotExpression	Create new Not Expression. The Not Expression returns negation of an expression. If an expression value returns "true" then that expression written inside the "NotExpression" returns "false" as a result

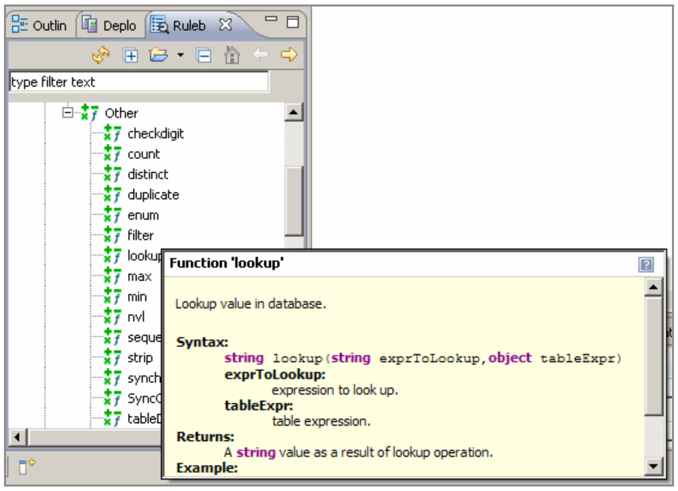
	value. These kind of expressions are used when you need negation of computational expressions.
NotGroupExpre...	Create new Not Group Expression. The Not Group Expression returns negation of an expression. If an expression value returns "true" then that expression written inside the "NotGroupExpression" returns "false" as a result value. These kind of expressions are used when you need negation of computational expressions.

HTML Tooltips

For ease of use, help is displayed when hovering over elements in the UI.

For example, in the **Rulebase Data View**, hover over functions to get a detailed description of that function including syntax and an example.

Or hover over a variable to get the description for that variable.



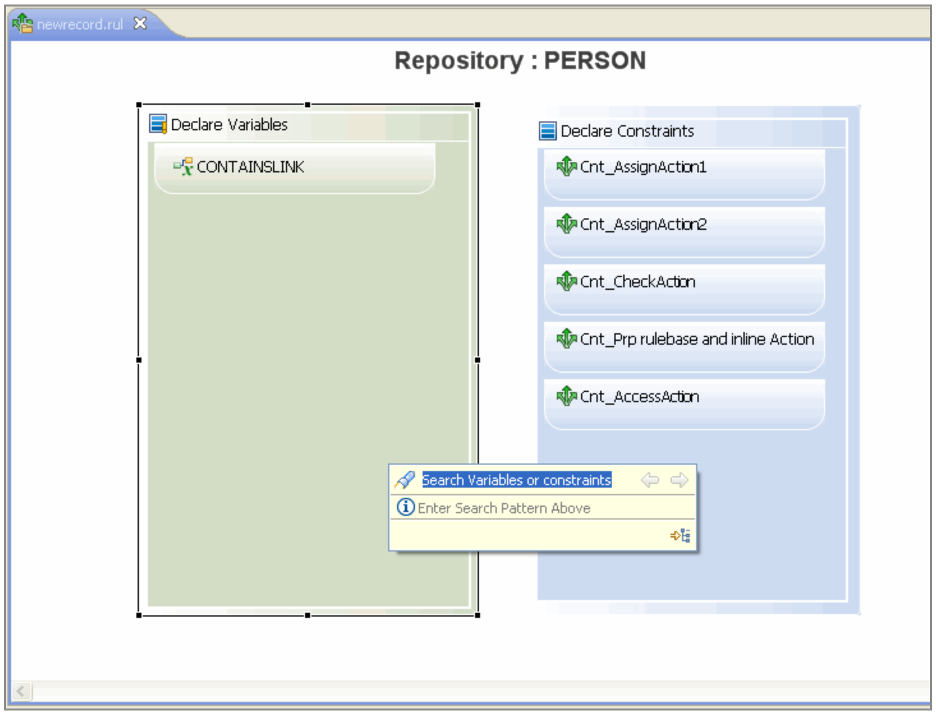
Function 'lookup'

Lookup value in database.

Syntax:
string `lookup(`**string** `exprToLookup,`**object** `tableExpr)`
exprToLookup:
expression to look up.
tableExpr:
table expression.
Returns:
A **string** value as a result of lookup operation.
Example:

Performing Quick Search

Using the type ahead search dialog, you can search for the variables declared in the Declare section and the constraints declared in the Constraints section.



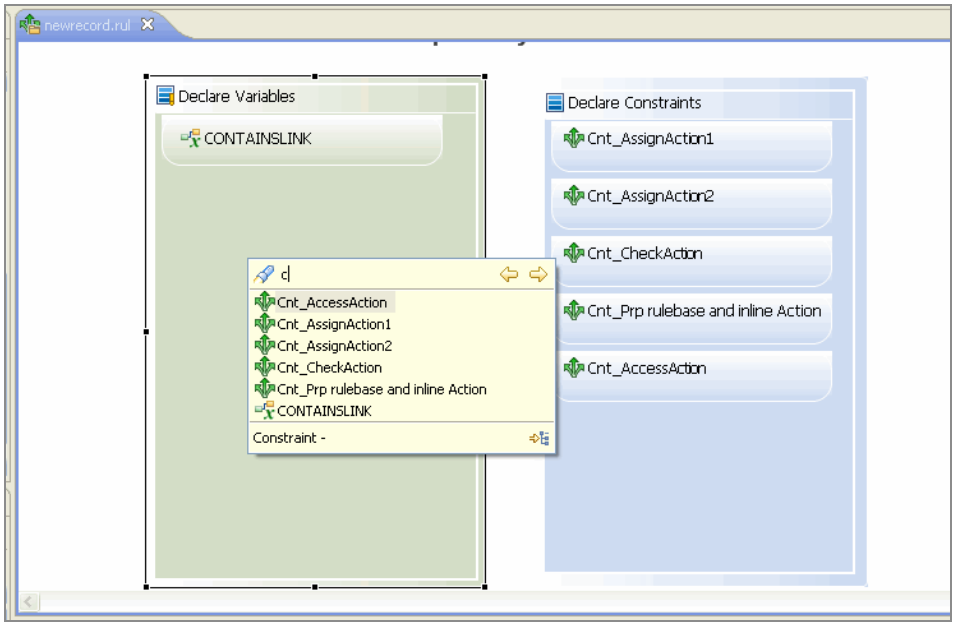
Procedure

1. In the toolbar, press Ctrl+F or click .

   The type ahead search dialog is displayed.

2. Enter the search criteria in the Search Variable or constraints field.

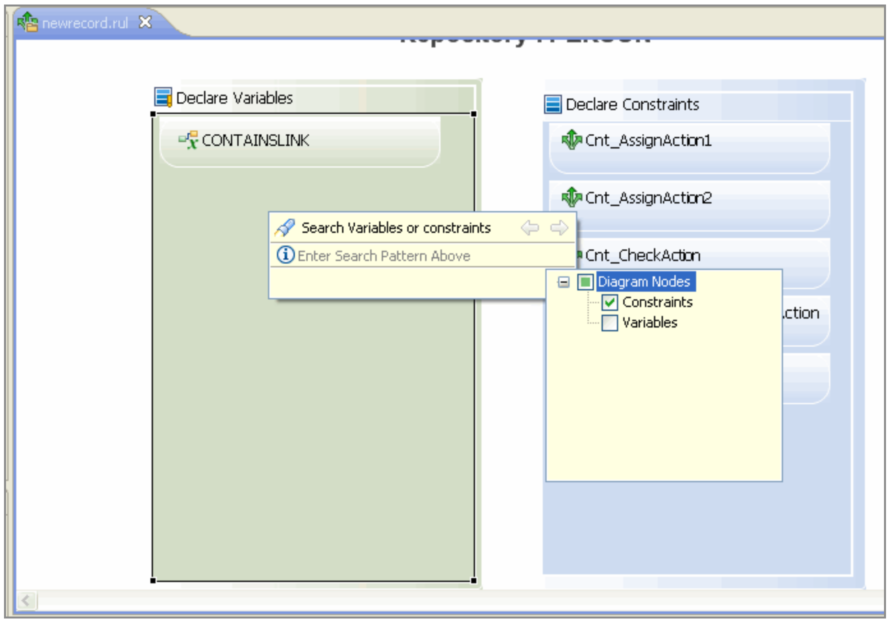
   The matching search result for the search criteria entered is shown.



By default the search criteria shows both variables and constraints. You can restrict the search criteria to show only variables or only constraint or both.

3. Click **Select Search Categories**.

   A list of search categories are displayed.



- select the **Diagram Node** check box to include both variables and constraints

in the search result.
- select the **Constraints** check box to include only constraints in search result.
- select the **Variables** check box to include only variables in the search result.

Rulebase Navigation

The Rulebase model tree navigation is enhanced. It is categorized into the following five sections.

- Common
- Gdsn
- MassUpdate
- Other
- Search

The rulebases are grouped based on a specific category. For example, Rulebase of type Gdsn is grouped into Gdsn folder, rulebase of type massupdate is grouped in MassUpdate folder. Similarly the Common folder contains general type rulebases. The rulebase of type Other is grouped in Other folder and rulebase of type search is grouped in Search folder.

Create a New Rulebase

Before creating a rulebase you must create a project to hold the rulebase. The Create a New Rulebase chapter explains in details how to create a new Rulebase.

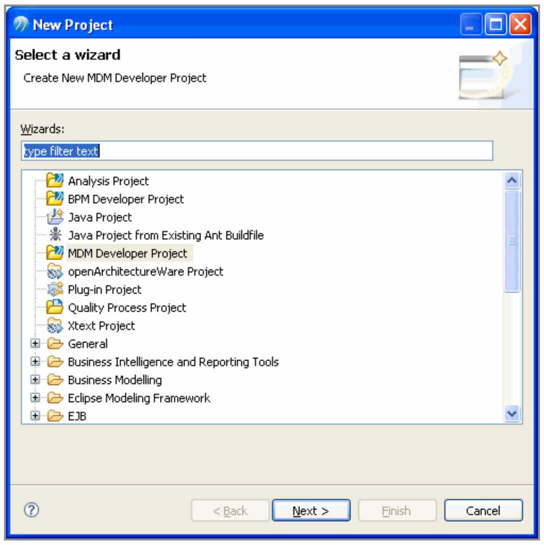
Sr. No.	Step	Additional Information
1.	Creating a new Project to hold your Rulebase Model	First create a project with containers (appropriate folders) for **rulebase models** and **repository models**. Repository models are required because rules run on repositories.
2.	Defining or Importing Repository Data	Next, either create a repository model or import one (into the repository models folder) for association with the rulebase. **Note: Note:** To create a Rulebase of type other than Initialization, Validation, or Search, you must not associate a repository.
3.	Creating a Rulebase Model	Finally, create a rulebase model.
4.	Importing Users and Roles	Optionally, import users and roles.

Creating a New Project to Hold Your Rulebase Model

To create a rulebase model, you first need to create a project to hold your model.

Procedure

1. Go to **File > New > Project**. The Create New TIBCO MDM Developer Project wizard is displayed.



New Project

Select a wizard

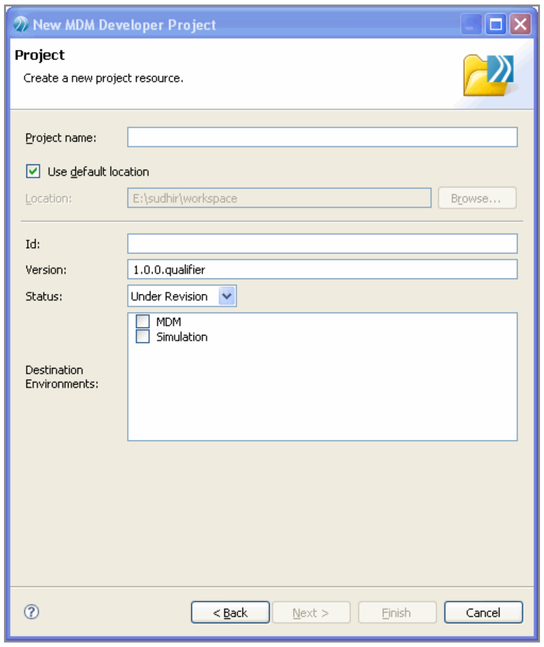
Create New MDM Developer Project

Wizards:

type filter text

- Analysis Project
- BPM Developer Project
- Java Project
- Java Project from Existing Ant Buildfile
- MDM Developer Project
- openArchitectureWare Project
- Plug-in Project
- Quality Process Project
- Xtext Project
- General
- Business Intelligence and Reporting Tools
- Business Modelling
- Eclipse Modeling Framework
- EJB

2. Select **MDM Developer Project** and click **Next**.
3. Provide a name for the Project. Clear the **Use default location** check box if you want to provide a different location for the project (by default, the current workspace). Select Destination Environment as **MDM**. Click **Next**



New MDM Developer Project

Project

Create a new project resource.

Project name:

☑ Use default location

Location: E:\sudhir\workspace

Id:

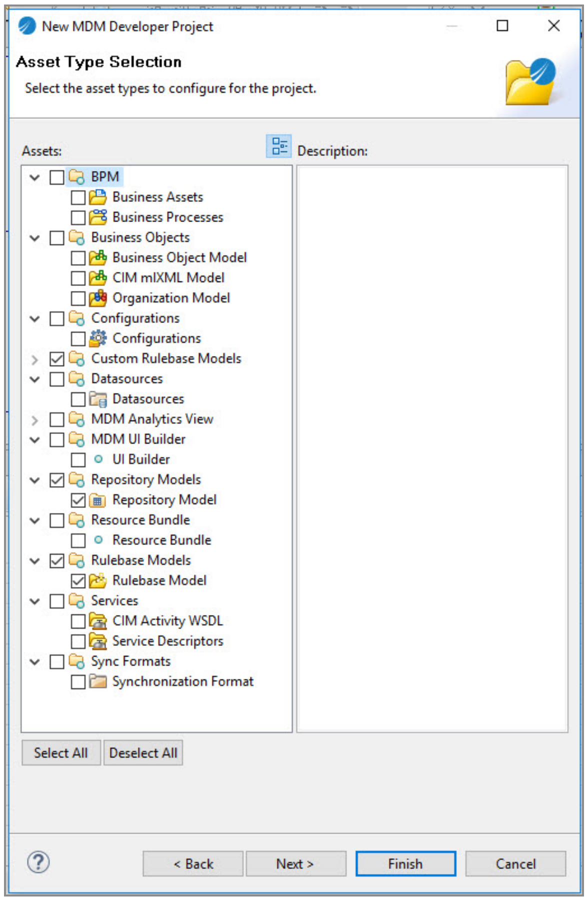
Version: 1.0.0.qualifier

Status: Under Revision

Destination Environments: ☐ MDM ☐ Simulation

4. The Asset Type Selection dialog box is displayed - select the **Repository Models** and

click **Next**.


New MDM Developer Project

Asset Type Selection

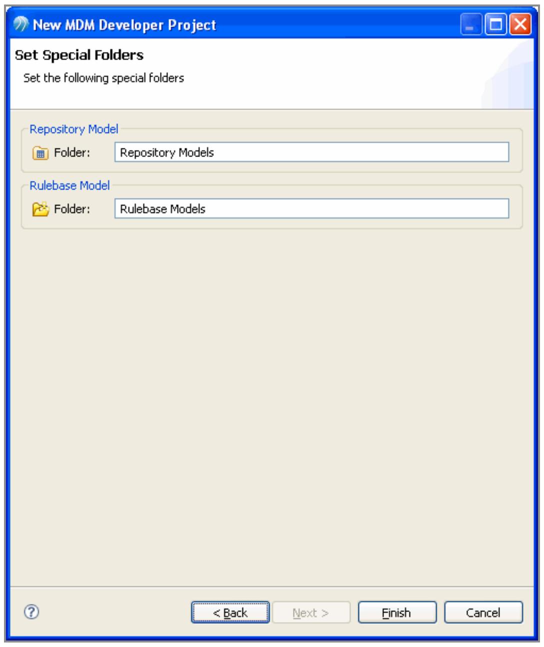
Select the asset types to configure for the project.

Assets:

Description:

- [ ] BPM
  - [ ] Business Assets
  - [ ] Business Processes
- [ ] Business Objects
  - [ ] Business Object Model
  - [ ] CIM mIXML Model
  - [ ] Organization Model
- [ ] Configurations
  - [ ] Configurations
- [x] Custom Rulebase Models
- [ ] Datasources
  - [ ] Datasources
- [ ] MDM Analytics View
- [ ] MDM UI Builder
  - [ ] UI Builder
- [x] Repository Models
  - [x] Repository Model
- [ ] Resource Bundle
  - [ ] Resource Bundle
- [x] Rulebase Models
  - [x] Rulebase Model
- [ ] Services
  - [ ] CIM Activity WSDL
  - [ ] Service Descriptors
- [ ] Sync Formats
  - [ ] Synchronization Format

5. The folder for the **Repository Models** is displayed. Click **Finish**. See Rulebase Properties for more details.



Set Special Folders

Set the following special folders

Repository Model

Folder: Repository Models

Rulebase Model

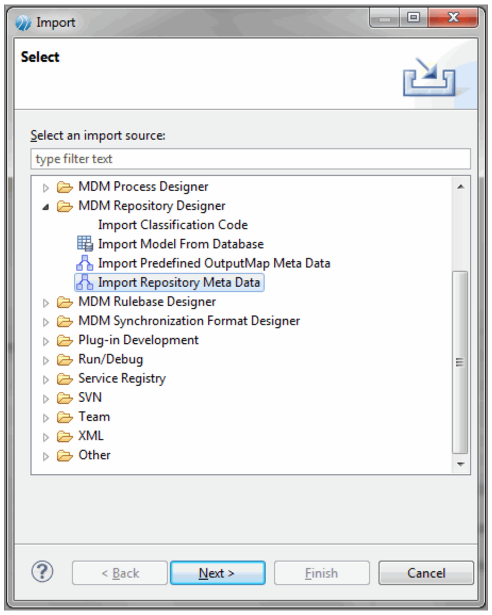
Folder: Rulebase Models

Defining or Importing Repository Data

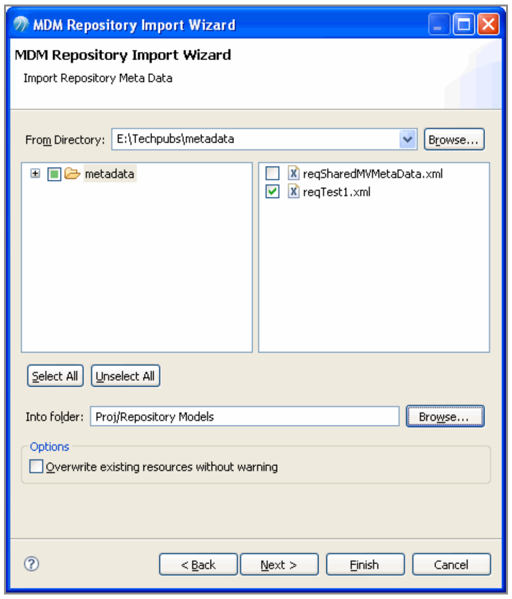
Note: Rules run on repository data. Before defining rules, the metadata you want the rules to run on must be created, defined, or imported.

Procedure

1. Right-click the **Repository Models** folder in the Project Explorer and select **Import**.
2. Select **Import Repository Meta Data** under **MDM Repository Designer**. Click **Next**.



3. Browse and select the repository metadata (.xml format). Click **Finish**.


MDM Repository Import Wizard
MDM Repository Import Wizard
Import Repository Meta Data
Select All
Unselect All
Browse...
Options
< Back
Next >
Finish
Cancel

From Directory: E:\Techpubs\metadata

metadata

- [ ] reqSharedMVMetaData.xml
- [✓] reqTest1.xml

Into folder: Proj/Repository Models

- [ ] Overwrite existing resources without warning

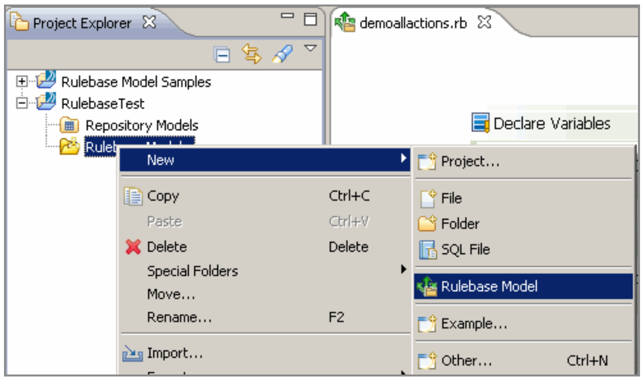
A repository (.rep) import file is created under the target folder.

Creating a Rulebase Model

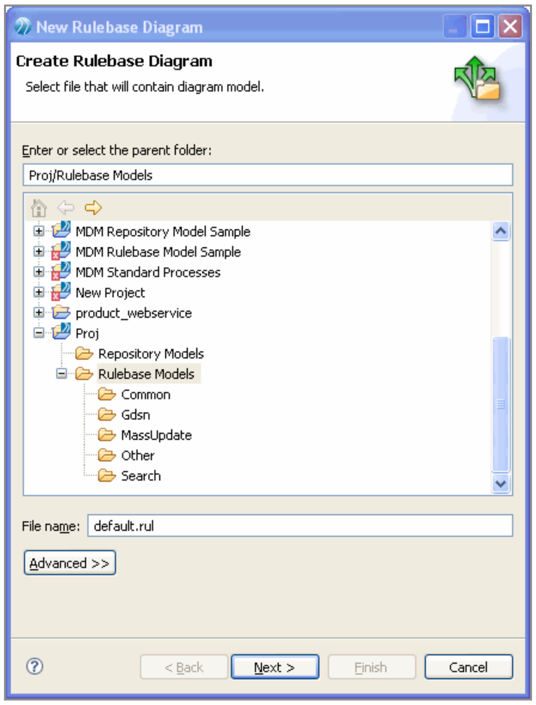
Note: Ensure that you have defined or imported repository metadata before attempting to create a rulebase model.

Procedure

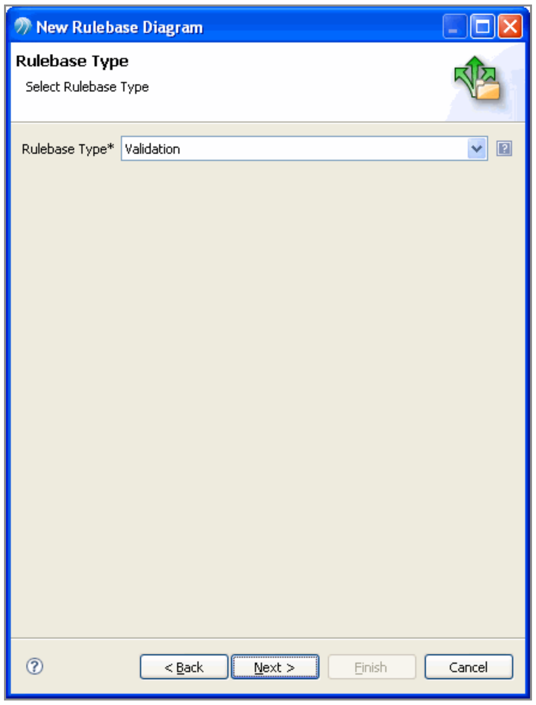
1. Right click the Rulebase Models folder in the Project Explorer and select New Rulebase Model.



2. Accept the default name for the rulebase model (default.rul) and location or enter a new location and name. Click **Next**.



3. Select the type of Rulebase: Initialization, Validation, Search, Other, Gdsn, MassUpdate.



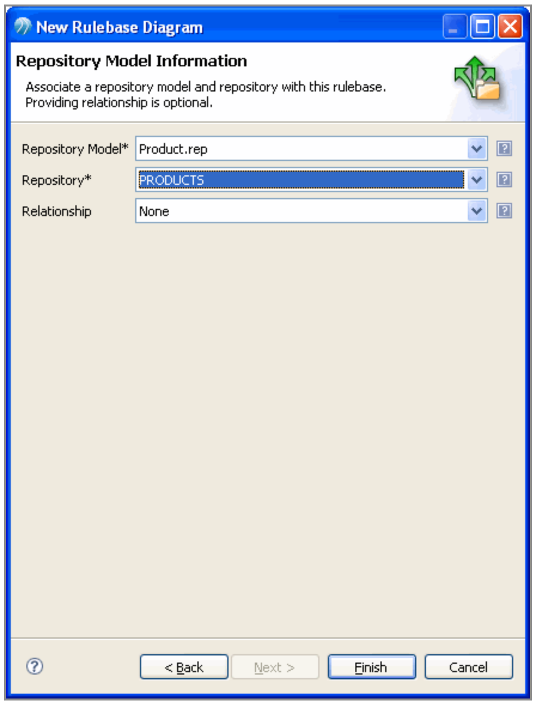
4. Click **Next**.

5. Associate repository data with your rulebase.
   - Select the Repository Model .rep file. This is mandatory field.
   - Select the Repository to associate. This is mandatory field.
   - Select the Relationship to associate.

   > **Note:** To associate a relationship, select the relationship’s source (and not target) repository from the **Repository** drop-down; the **Relationship** drop-down will then get populated with available relationships that you can select to associate.

6. Click **Finish**.

Result

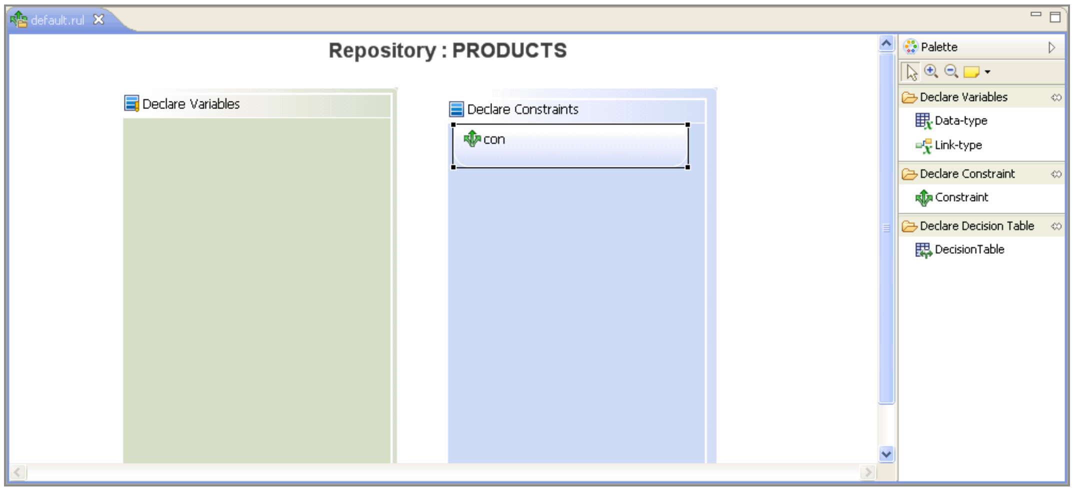

New Rulebase Diagram

Repository Model Information

Associate a repository model and repository with this rulebase.
Providing relationship is optional.

Repository Model*	Product.rep
Repository*	PRODUCTS
Relationship	None

The rulebase diagram is then displayed in the Editor. Use the palette to start building your rulebase by declaring variables and adding constraints. For more details, see Types of Variables.

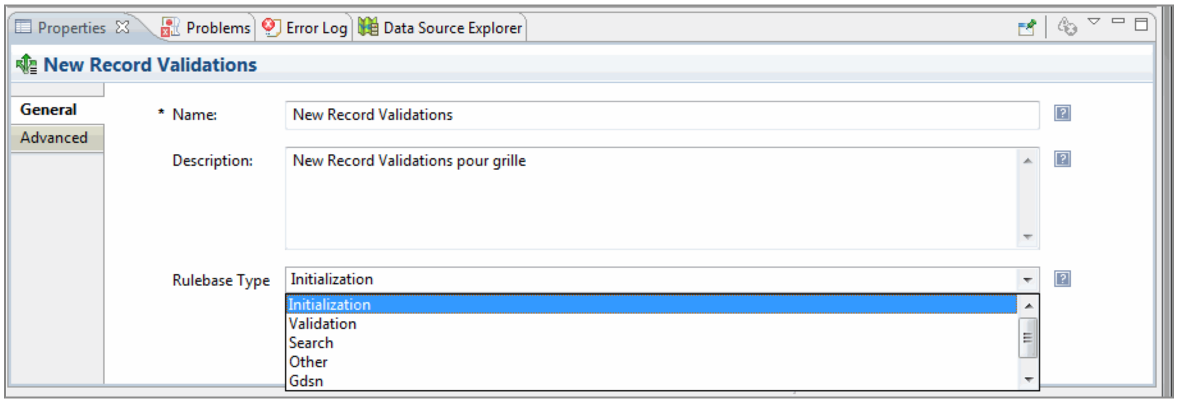


Rulebase Properties

Double click the rulebase (**.rul**) file under the **Rulebase Models** folder in the Project Explorer to view or change the properties.

General Properties

This displays the Rulebase Name, Description, and Type.



New Record Validations

* Name: New Record Validations

Description: New Record Validations pour grille

Rulebase Type: Initialization

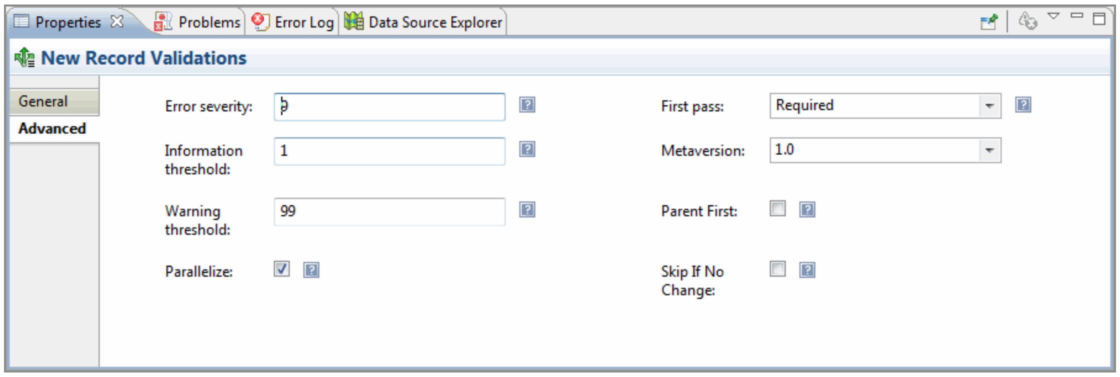
Advanced Properties

The following properties are displayed here:

- **Error Severity**: Defines the default severity for the rulebase and is set to 9 by default.

All validations with severity less than the set value are considered errors. All validations with severity greater than the set value are considered warnings. You can override this rulebase level setting by going to an individual constraint and changing its Severity.

- **Information Threshold**: Information threshold, set to 1 by default.
- **Warning Threshold**: Warning threshold, set to 99 by default.
- **Parallelize**: Indicates that this rulebase does not depend on order of execution of other rulebase within record hierarchy and can be executed in parallel.
- **First Pass**: When saving records, a rulebase is executed twice. The first time the rulebase handles propagations and the second time, assignments or validations are done. Set this value to **Required** if the parent record needs to propagate values to a child. **Skip** is the default option (no propagations defined in the rulebase).
- **Metaversion**: The version of the file.
- **Parent First**: If there are one or more parents in the hierarchy, evaluate the parent rulebase first. This is usually used to indicate, the child record has a dependency on the parent or higher level parent and parent record rulebase has to be processed prior to child.
- **Skip If No Change**: Indicates that for a record in the hierarchy, if records are not modified, no validation be done. This directives applies only for operations which modify record (merge, modify, delete).



Error severity:	9	First pass:	Required
Information threshold:	1	Metaversion:	1.0
Warning threshold:	99	Parent First:	☐
Parallelize:	✓	Skip If No Change:	☐

Actions Allowed for Different Types of Rulebases

The following actions are supported for different types of rulebases.

Rulebase Type	Action	Usage	Filename
Initialization	Assign Propagate	Add	newrecord.xml
Relationship	Assign Check Softlink Select Access	While creating, need to select parent repository and relationship.	newrecord.xml searchcontrolrules.xml catalogvalidation.xml
Search	Select	Browse and Search	searchcontrolrules.xml
Validation	All actions	Add/Modify/View	catalogvalidation.xml
Other	Assign Access Select Include Clear	In other rulebase or in workflow activities.	<filename>.xml
GDSN	commonly used actions like Assign, Access, Select	to support gdsn	<filenname_gdsn>.xml
MassUpdate	Assign Access	Advanced Mass Update	<filename>.xml

Rulebase Type	Action	Usage	Filename
	Select Include Clear		

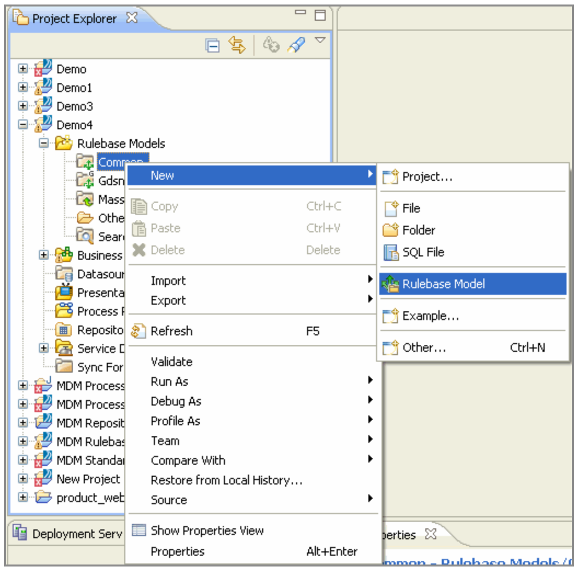
- The Initialization, Validation, Search, and Other type Rulebases are saved in \common_dir\<Enterprise name>\catalog\master\<Catalog ID>\
- The GDSN and Mass update type Rulebase are saved in \common_dir\<Enterprise name>\rulebase\
- The Relationship type Rulebase is saved in \common_dir\<Enterprise name>\catalog\master\<Relationship table ID>\

Adding Rulebase to a Folder

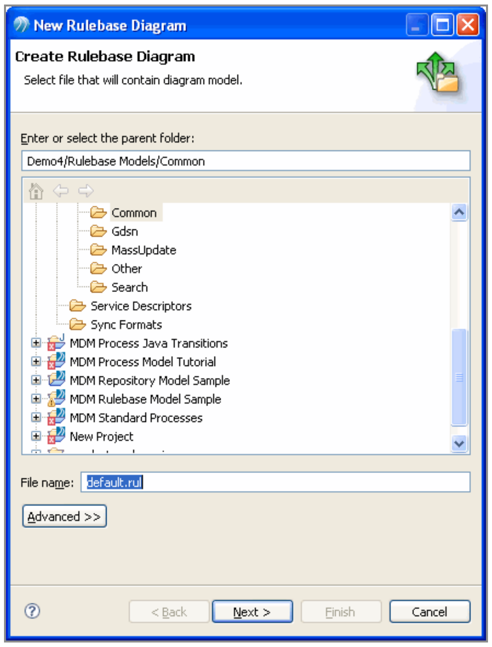
To add a new rulebase to a particular folder.

Procedure

1. Select the folder in which you want to add the rulebase. Right-click on the folder, for example, Common, and select **New** and click **Rulebase Model**.

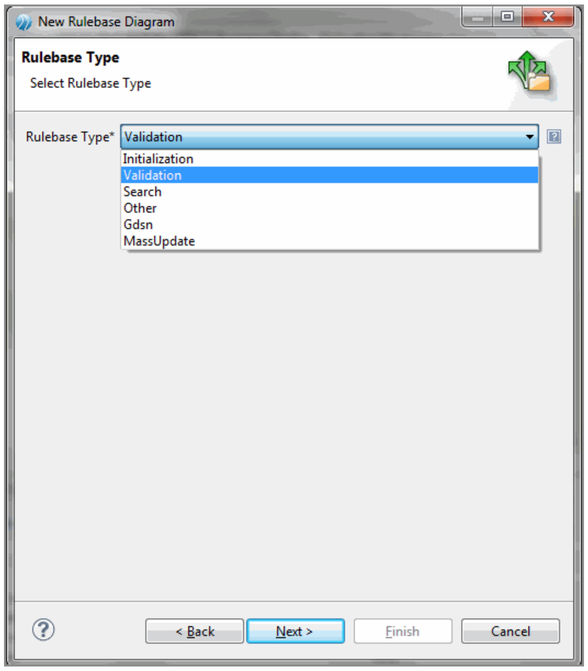


2. The Create Rulebase Diagram dialog is displayed.

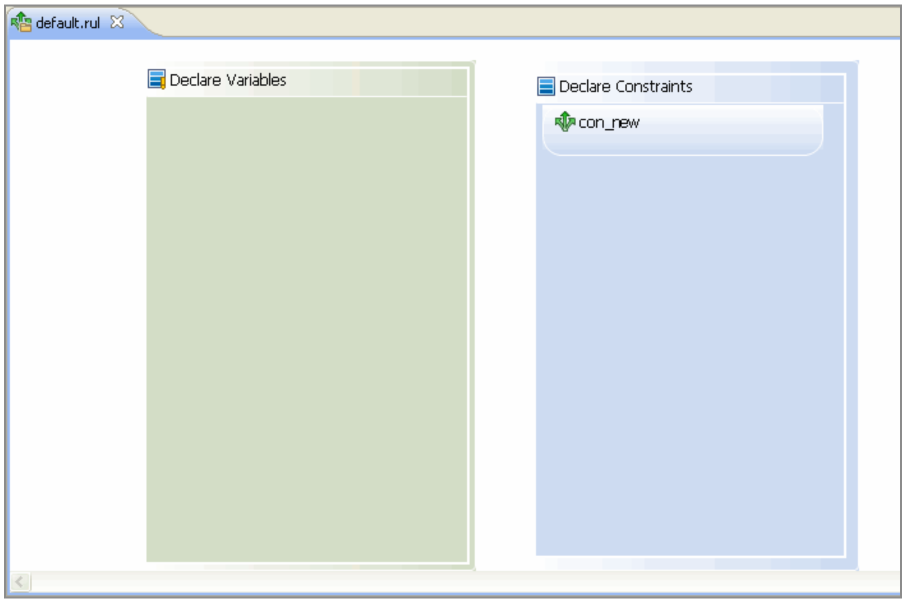

New Rulebase Diagram
Create Rulebase Diagram
Select file that will contain diagram model.
Enter or select the parent folder:
Demo4/Rulebase Models/Common
Common
Gdsn
MassUpdate
Other
Search
Service Descriptors
Sync Formats
MDM Process Java Transitions
MDM Process Model Tutorial
MDM Repository Model Sample
MDM Rulebase Model Sample
MDM Standard Processes
New Project
File name: default.rul
Advanced >>
< Back
Next >
Finish
Cancel

3. Enter appropriate rulebase base name in **File name** field.

4. Click **Next**. The **Rulebase Type** selection dialog is displayed.



5. Select the **Rulebase Type** from the drop-down list. Click **Next**.
6. The Rulebase diagram for the newly created rulebase is displayed.

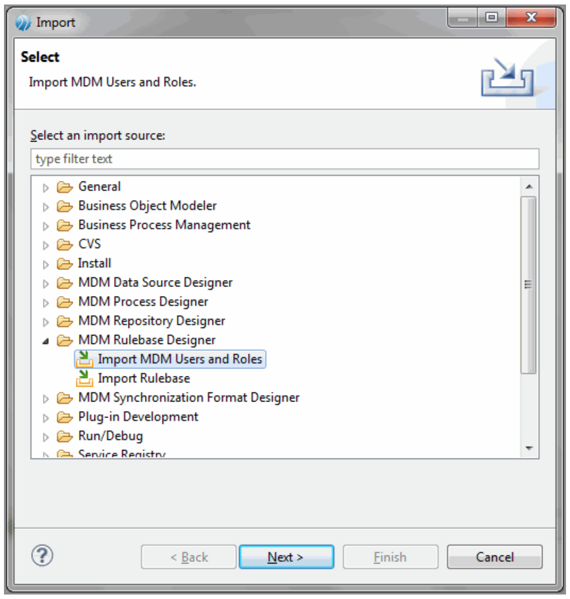


Importing Users and Roles

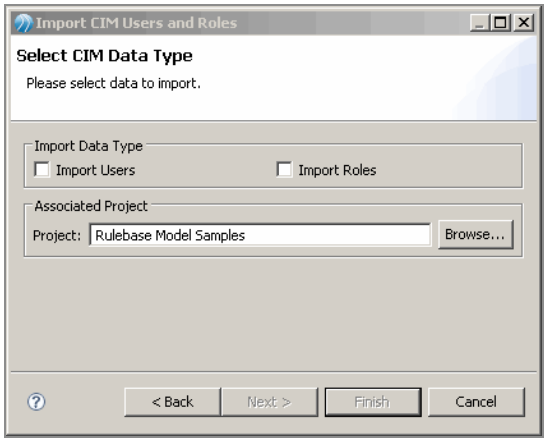
Importing of Users and Roles is an online activity.

Procedure

1. Right-click the **Rulebase Models** folder in the Project Explorer and click **Import**. Select **Import CIM Users and Roles** under **MDM Rulebase Designer.** Click **Next**.



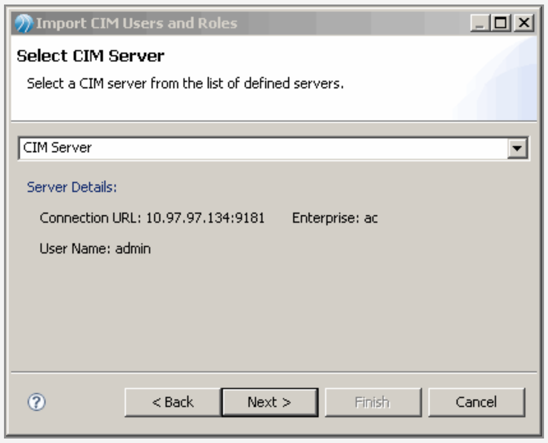
2. Import users, roles or both by selecting the related checkboxes. Browse to select the associated Project. Click **Next**.



Project: Rulebase Model Samples

3. Select the TIBCO MDM Server from the drop-down list of defined servers. Click **Next**.

Note: If you opted to import only roles (and not users), skip directly to Step 7.



Select a CIM server from the list of defined servers.

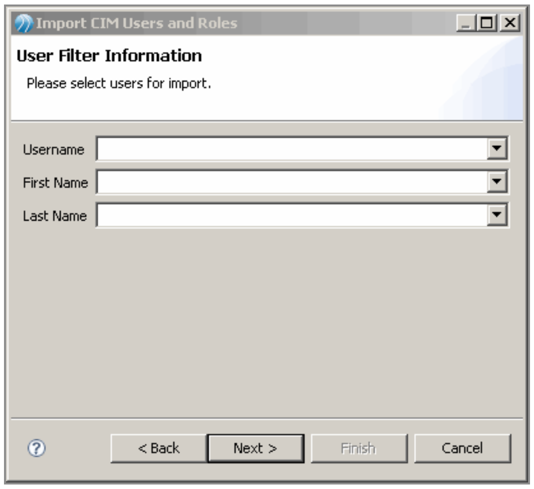
CIM Server

Server Details:

Connection URL: 10.97.97.134:9181   Enterprise: ac

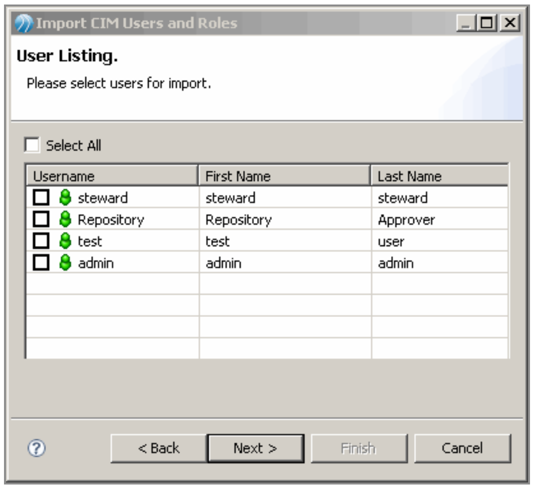
User Name: admin

4. This dialog can be used to filter user information for retrieval, for example, if you specify First Name as **A**, all users with A in the first name will be displayed in the next screen. Optionally, leave this screen blank (to display all users) and click **Next**.



Please select users for import.

5. If you left the previous screen blank, all applicable users will be displayed here. If you provided some filter criteria in the previous screen, users that match that criteria will be displayed. Select the checkboxes of users to import and click **Next**.



User Listing.

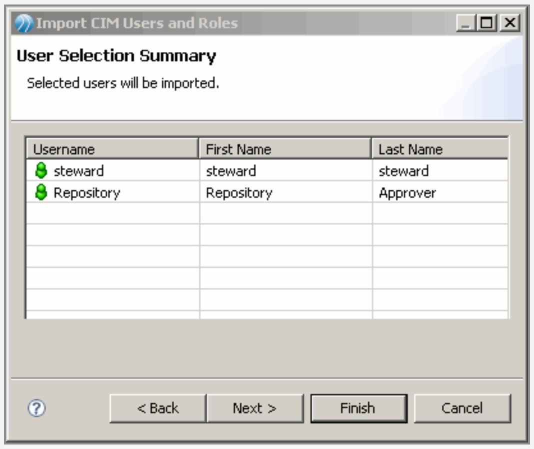
Please select users for import.

☐ Select All

Username	First Name	Last Name
☐ steward	steward	steward
☐ Repository	Repository	Approver
☐ test	test	user
☐ admin	admin	admin

6. Selected users are displayed for confirmation. Click **Next**.

Note: If you selected to import both roles and users but do not want to import roles at this point, click Finish instead of **Next**.

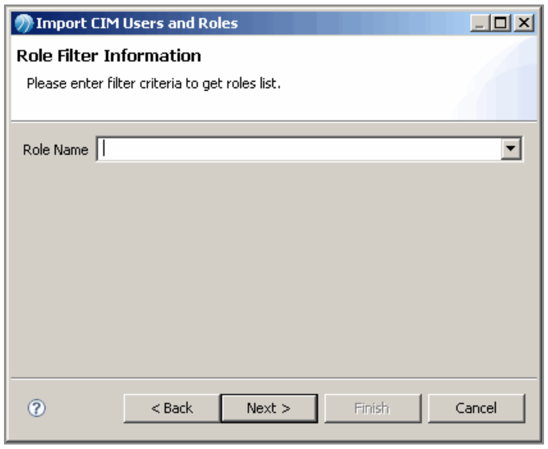


User Selection Summary

Selected users will be imported.

Username	First Name	Last Name
steward	steward	steward
Repository	Repository	Approver

7. Next, enter information for the role to import or leave this screen blank and click **Next** to see all available roles.

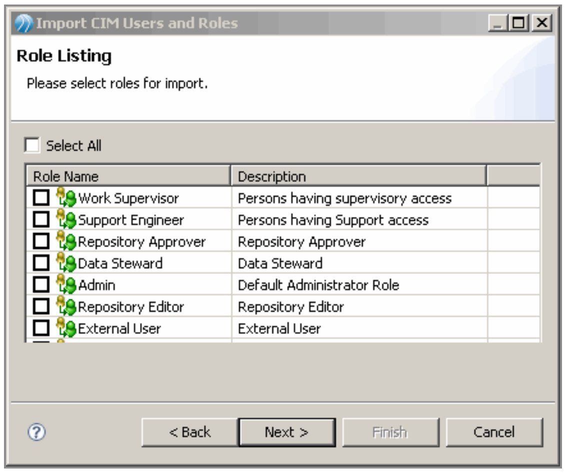


Role Filter Information

Please enter filter criteria to get roles list.

Role Name

8. A list of applicable roles are displayed for selection; select the appropriate checkboxes and click **Next**.



Role Listing

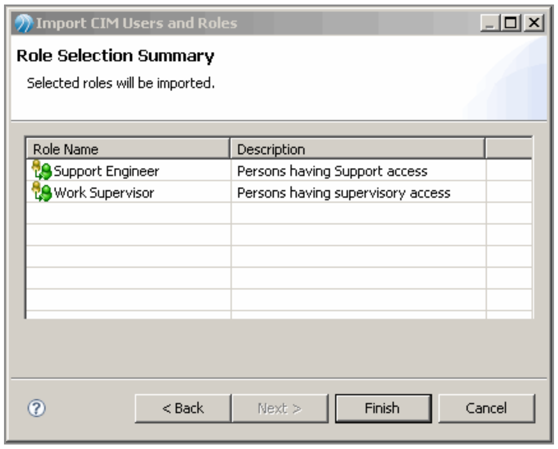
Please select roles for import.

☐ Select All

Role Name	Description
☐ Work Supervisor	Persons having supervisory access
☐ Support Engineer	Persons having Support access
☐ Repository Approver	Repository Approver
☐ Data Steward	Data Steward
☐ Admin	Default Administrator Role
☐ Repository Editor	Repository Editor
☐ External User	External User

9. Selected roles are displayed for confirmation. Click **Finish**.

Result



Role Selection Summary

Selected roles will be imported.

Role Name	Description
Support Engineer	Persons having Support access
Work Supervisor	Persons having supervisory access

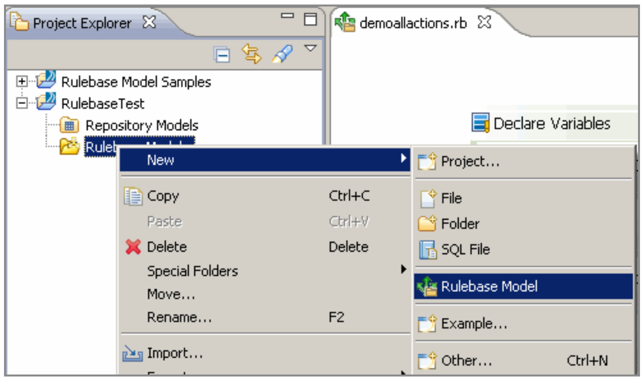
You will get a message confirming import of roles and/or users.

Creating a Rulebase Model

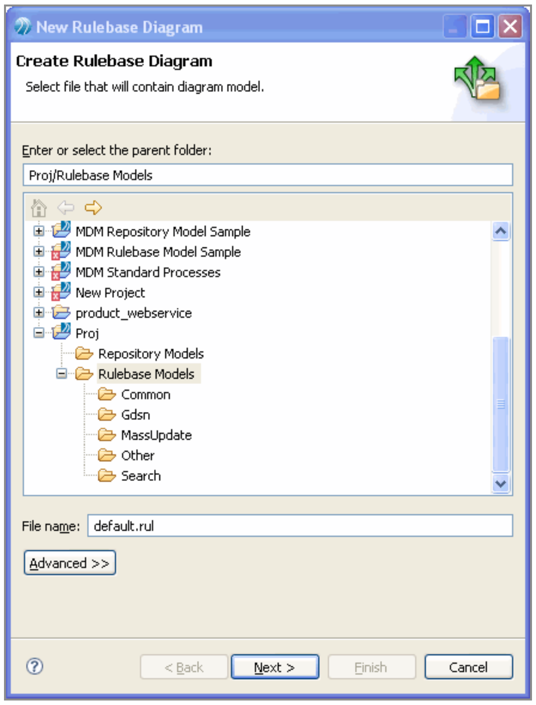
Note: Ensure that you have defined or imported repository metadata before attempting to create a rulebase model.

Procedure

1. Right click the Rulebase Models folder in the Project Explorer and select New Rulebase Model.



2. Accept the default name for the rulebase model (default.rul) and location or enter a new location and name. Click **Next**.


New Rulebase Diagram

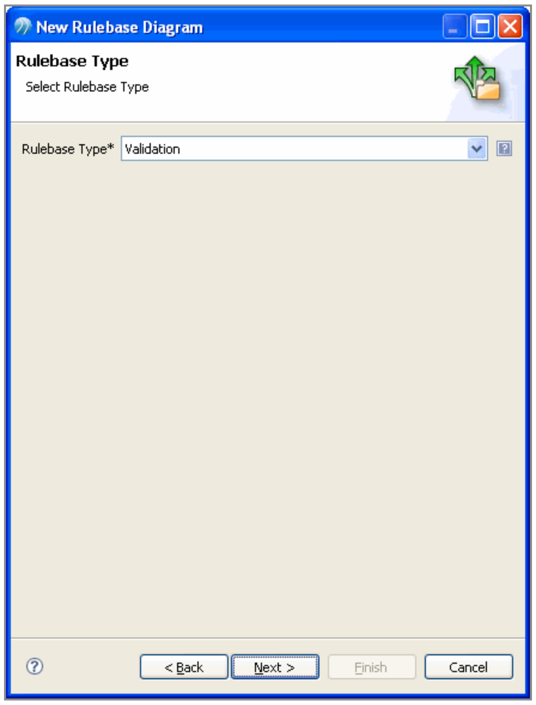
Create Rulebase Diagram

Select file that will contain diagram model.

Enter or select the parent folder: Proj/Rulebase Models

File name: default.rul

3. Select the type of Rulebase: Initialization, Validation, Search, Other, Gdsn, MassUpdate.


New Rulebase Diagram

Rulebase Type

Select Rulebase Type

Rulebase Type* Validation

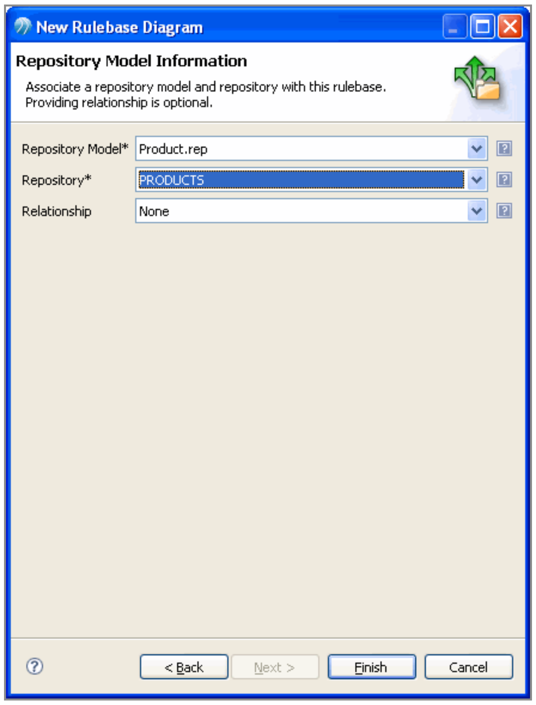
4. Click **Next**.

5. Associate repository data with your rulebase.
    - Select the Repository Model .rep file. This is mandatory field.
    - Select the Repository to associate. This is mandatory field.
    - Select the Relationship to associate.

    Note: To associate a relationship, select the relationship's source (and not target) repository from the **Repository** drop-down; the **Relationship** drop-down will then get populated with available relationships that you can select to associate.

6. Click **Finish**.

Result

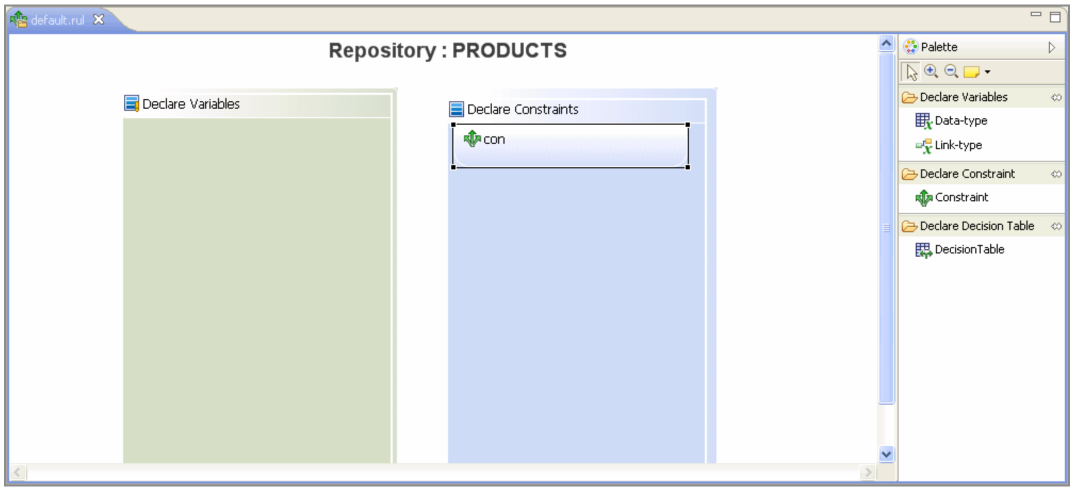

New Rulebase Diagram

Repository Model Information

Associate a repository model and repository with this rulebase.
Providing relationship is optional.

Repository Model* Product.rep
Repository* PRODUCTS
Relationship None

The rulebase diagram is then displayed in the Editor. Use the palette to start building your rulebase by declaring variables and adding constraints. For more details, see Types of Variables.


default.rul
Repository : PRODUCTS
Declare Variables
Declare Constraints
con
Palette
Declare Variables
Data-type
Link-type
Declare Constraint
Constraint
Declare Decision Table
DecisionTable

Custom Rulebase

In addition to the predefined functions, you can use custom functions to create your own custom rulebase. The custom functions are written in the `.java` file. After the `.java` files are compiled, custom functions are populated in the **Rulebase Data View**. The custom rulebase is used for creating the custom rulebase model.

Creating a New Project for Custom Rulebase Model

Procedure

You can create the custom project by following the steps given in the Creating a new Project to hold your Rulebase Model topic and select the Custom Rulebase Model folder in the Asset Type Selection window.

Alternatively, complete the following steps to create custom project containing only the Custom Rulebase Model folder:

1. From the menu, select **File > New > Project**.

   The Select a wizard page is displayed.

2. On the Select a wizard page, select **Custom Rulebase Project** and click **Next**.

3. On the Project page, provide the following information and click **Next**:

   - In the **Project name** field, provide a name for the project.
   - To change the default location, clear **Use default location** and provide the custom path to the project. (By default, the current workspace is selected).
   - Select **Destination Environment** as **MDM**.

4. On the Set Special Folders page, ensure that the **Custom Rulebase Models** folder is selected and click **Finish**.

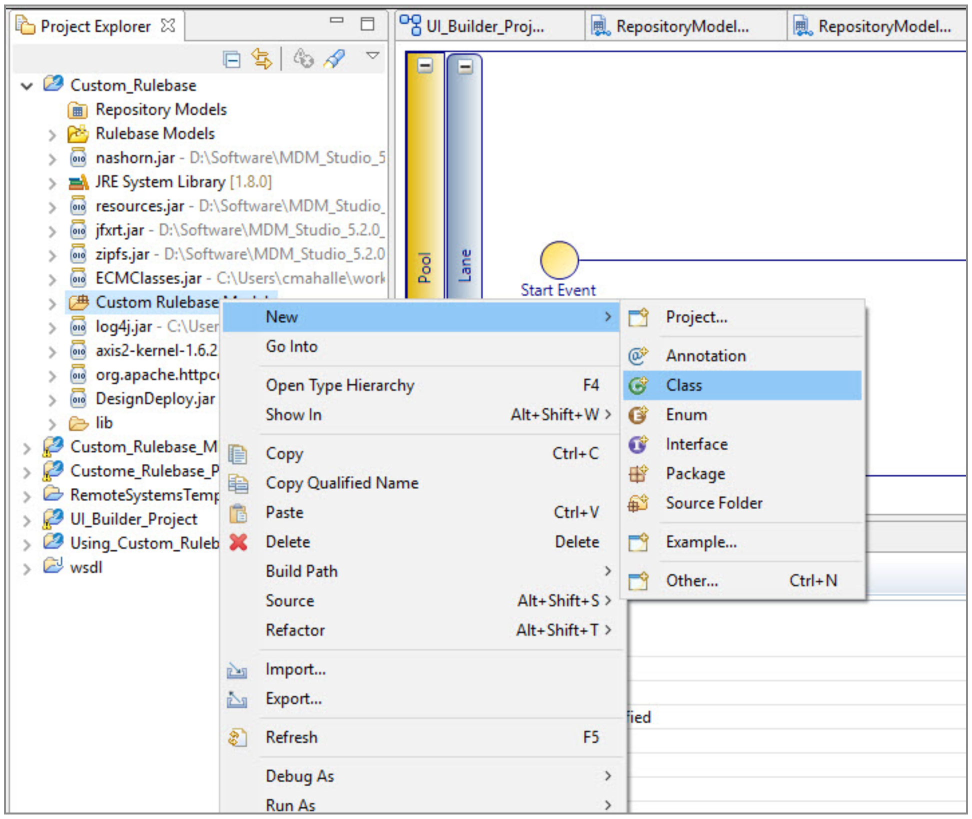
   A custom rulebase project is created which contains the **Custom Rulebase Model** folder.

Creating Custom Rulebase Model

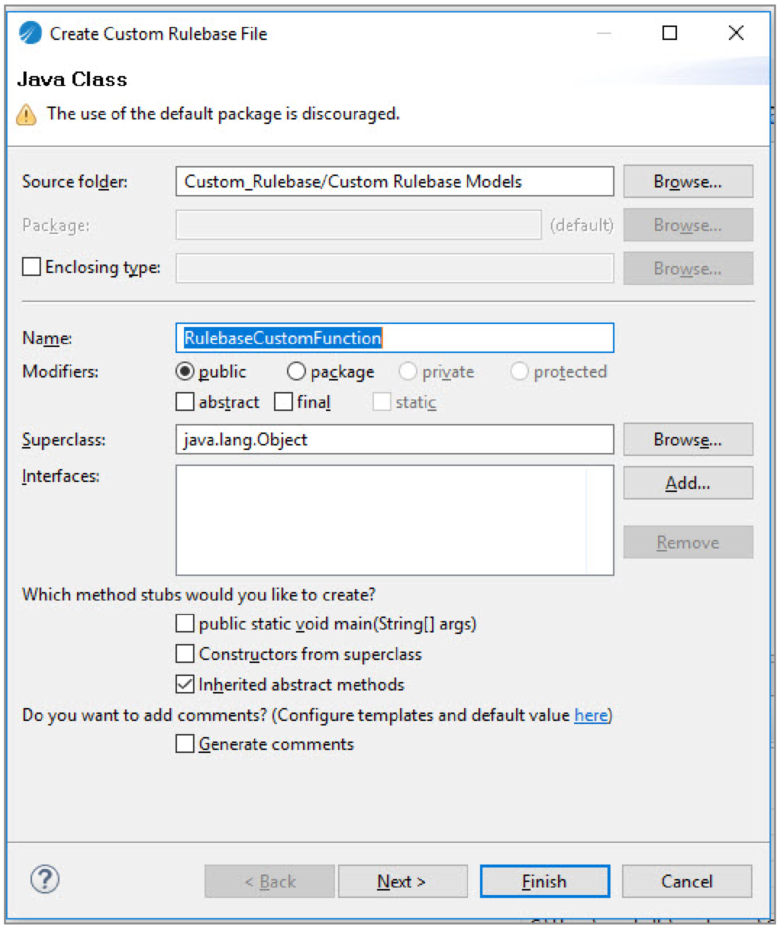
You can create a custom rulebase model by using following instructions:

Procedure

1. In the Project Explorer, right-click the **Custom Rulebase Model** folder and select **New** > **Class**.



2. On the Java Class page, enter the name of the class. (The default name is RulebaseCustomFunction.)


Create Custom Rulebase File

Java Class

The use of the default package is discouraged.

Source folder: Custom_Rulebase/Custom Rulebase Models

Package: (default)

Enclosing type:

Name: RulebaseCustomFunction

Modifiers: (● public) ○ package ○ private ○ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

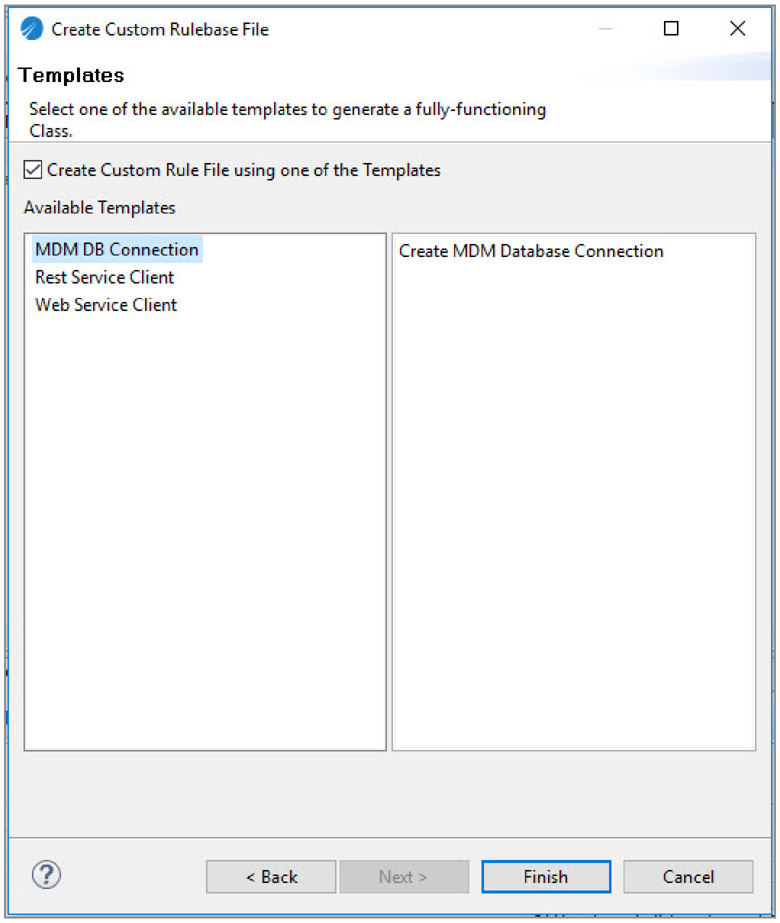
☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

3. Click **Finish** to create a class without predefined sample, or click **Next** to create a class by using predefined template.
4. In the Templates window, select **Create Custom Rule File using one of the Templates**, this enables the selection of **Available Templates**. And select one of the available templates based on your requirement.


Create Custom Rulebase File
Templates
Select one of the available templates to generate a fully-functioning Class.
☑ Create Custom Rule File using one of the Templates
Available Templates
MDM DB Connection
Rest Service Client
Web Service Client
Create MDM Database Connection
< Back
Next >
Finish
Cancel

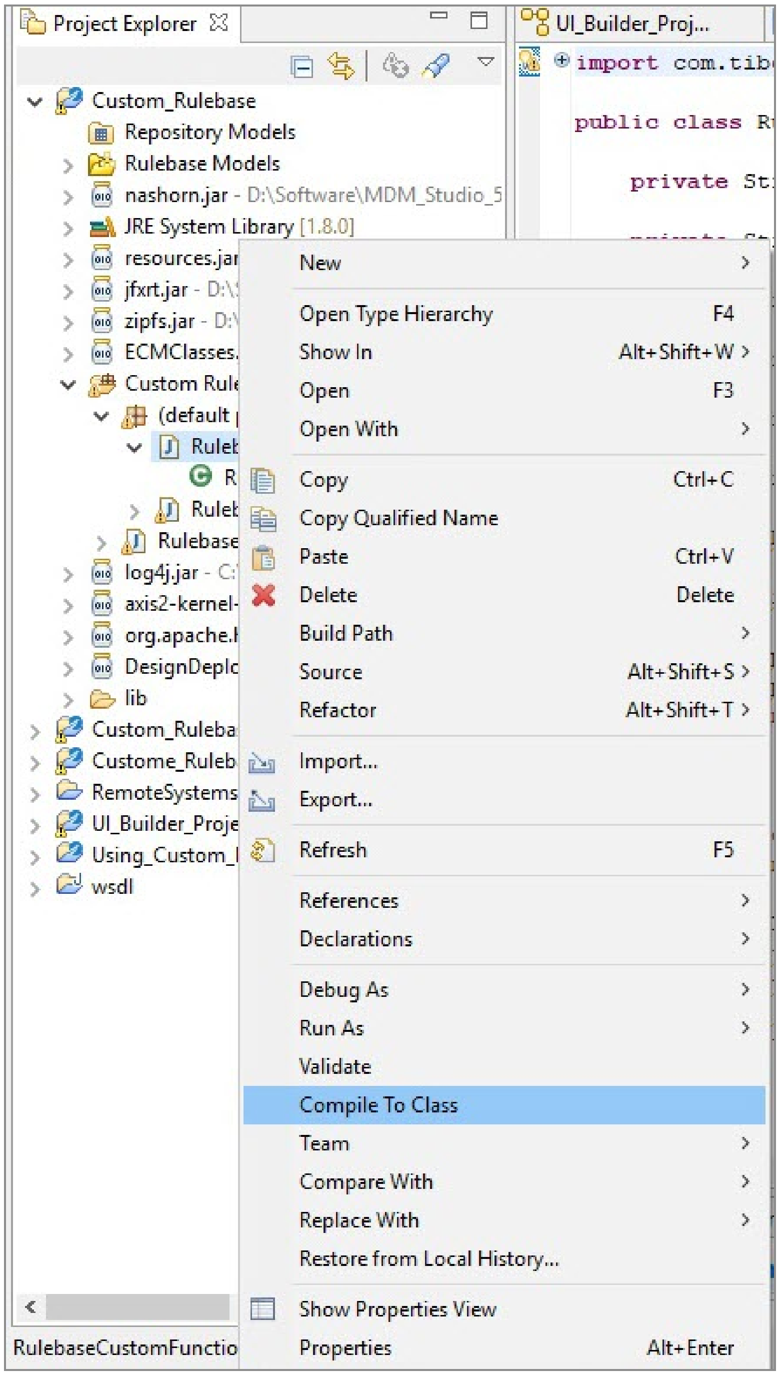
5. Click **Finish**.

Compiling Custom Rulebase File

After the Custom Rulebase file is created, it is compiled.

Procedure

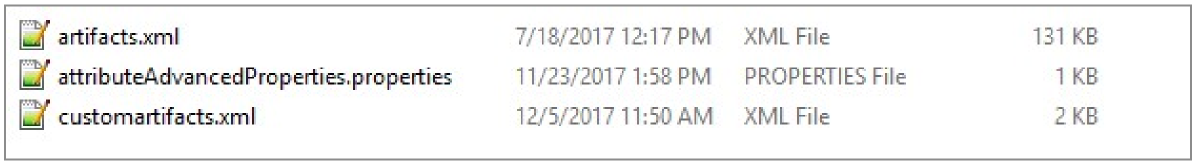
1. Right-click the `RulebaseCustomeFunction.java` file and select **Compile to Class**.


Project Explorer
UI_Builder_Proj...
Custom_Rulebase
Repository Models
Rulebase Models
nashorn.jar - D:\Software\MDM_Studio_5
JRE System Library [1.8.0]
resources.jar
jfxrt.jar - D:\
zipfs.jar - D:\
ECMClasses.
Custom Rule
(default
Ruleb
R
Ruleb
Rulebase
log4j.jar - C:
axis2-kernel-
org.apache.
DesignDeplo
lib
Custom_Ruleba
Custome_Ruleb
RemoteSystems
UI_Builder_Proje
Using_Custom_
wsdl
RulebaseCustomFunctio
import com.tib
public class R
private St
New
Open Type Hierarchy F4
Show In Alt+Shift+W
Open F3
Open With
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
References
Declarations
Debug As
Run As
Validate
Compile To Class
Team
Compare With
Replace With
Restore from Local History...
Show Properties View
Properties Alt+Enter

Custom Rulebase File and Rulebase Model Integration

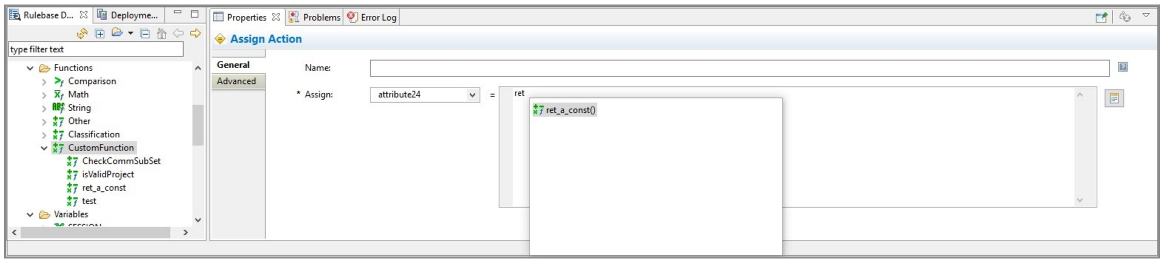
After the successful compilation of Custom Rulebase file, the `customartifacts.xml` file is created in the `.metadata` folder of that particular project.

In the `customartifacts.xml` file,the functions are added from Java files such as Name of Method and Function and Input and Output Parameters, except `execCustomFunction` method name.



artifacts.xml	7/18/2017 12:17 PM	XML File	131 KB
attributeAdvancedProperties.properties	11/23/2017 1:58 PM	PROPERTIES File	1 KB
customartifacts.xml	12/5/2017 11:50 AM	XML File	2 KB

The dynamically added functions are added in **Functions** > **Rulebase Data View** > **CustomFunction**. You can use these functions in the Expression Editor as shown in the following image:



Name:

* Assign: attribute24 = ret

Deploying Custom Rulebase Model

You can deploy Custom Rulebase model in the enterprise or standard folder.

Before you begin

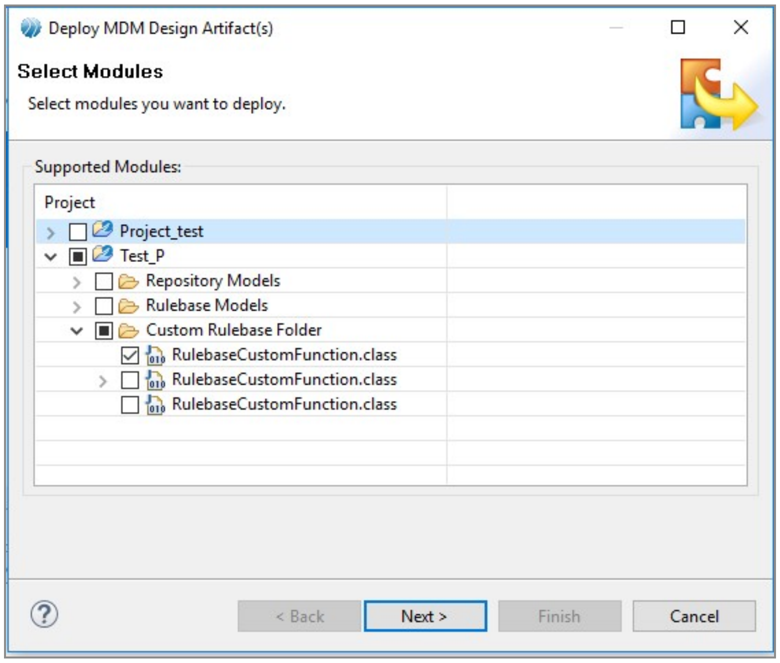
Make a note of the following considerations when deploying a custom rulebase model:

- Each project has only one **Custom Rulebase Models** folder that contains only `.java` files.
- If a project does not have **Custom Rulebase Models** folder, you can create it by using the **Special folder** option.

- After you create the **Custom Rulebase Models** folder, the **Do not use Custom Rulebase Models** option is no longer available for this folder.

Procedure

1. Select the `CustomRulebase.class` file in the Select Modules window.



Deploy MDM Design Artifact(s)

Select Modules

Select modules you want to deploy.

Supported Modules:

Project
- [ ] Project_test
- [■] Test_P
  - [ ] Repository Models
  - [ ] Rulebase Models
  - [■] Custom Rulebase Folder
    - [✓] RulebaseCustomFunction.class
    - [ ] RulebaseCustomFunction.class
    - [ ] RulebaseCustomFunction.class

2. Select enterprise or standard folder in which you want to deploy the custom rulebase model.
3. Click **Finish**.

   The `RulebaseCustomFunction.class` file is deployed in the Rulebase folder of the selected folder.

Variables

This chapter explains different kinds of variables, properties, and declaration.

Types of Variables

- User Defined Variables
- Implicit Context Variables

User Defined Variables

User defined variables can be of two types: Data-type and Link-type.

To create a standard variable, use the data-type variable. To create a link to another object, use the link-type variable.

Data-type variables

Use this type to create a standard variable that is not defining a relationship to or attempting to access another object.

Link-type variables

Use this type to create a variable that accesses related objects during rule execution.

Links can be of different types:

- **Relationship Record**: Has a relationship and record attribute associated with it and points to a record defined by the relationship. It follows the relationship to the target record and retrieves an attribute value (if specified) or the whole record.

- **Multi Relationship Record**: Has one or more relationships and record attributes and points to records obtained by following a chain of relationships.
- **Relationship**: Has a relationship and relationship attribute associated and is used to access relationship specific attributes.
- **Record**: Points to a list of catalog product objects.
- **Catalog**: Used to write SQL statements to access Repository tables.
- **Datasource**: Used to write SQL statements to access Datasource values.
- **Classification**: Used to access classification scheme objects and its details.
- **Classification Code**: Used to access classification codes and its details.

Implicit Context Variables

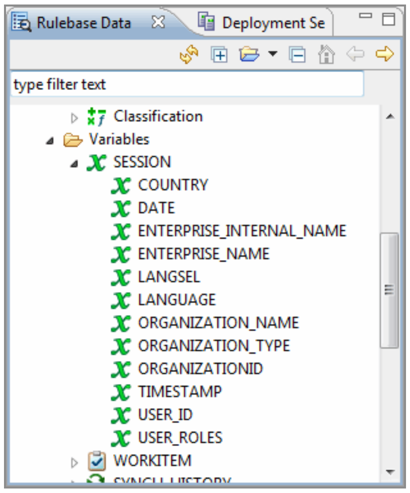
These variables are listed in the **Rulebase Data View** and can be dragged-and-dropped in the Expression Editor and used without explicit declaration (implicitly).

Most of these variables are available in specific contexts, for example, workitem variables will be available if your rulebase is executing in a workflow.

- Session Variables
- Workitem Variables
- Synch History variables
- System Variables
- Other Variables
- Workflow Variables
- Attribute History Variable
- Attribute Quality Variables
- Precedence Result Variable

Session Variables

Session variables are used to access values of system attributes of records.



Session Variables

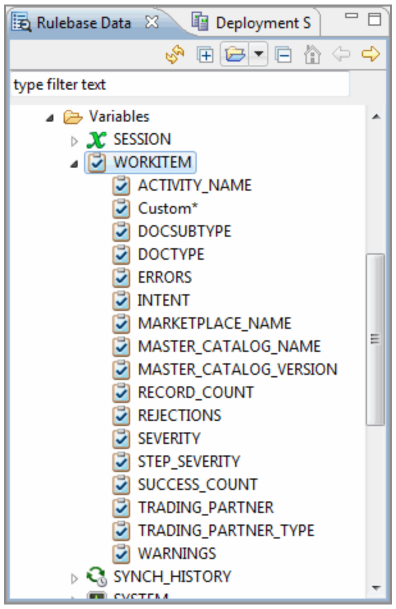
Variable	Type	Value
COUNTRY	String	Users Profile’s Country locale value.
DATE	Date	Current date.
ENTERPRISE_INTERNAL_NAME	String	Enterprise Internal Name.
ENTERPRISE_NAME	String	Enterprise Name.
LANGSEL	String	Language locale selected from login page.
LANGUAGE	String	Users profile’s Language locale value.
ORGANIZATION_NAME	String	Organization Name.
ORGANIZATION_TYPE	String	Organization Type.
ORGANIZATIONID	Number	Organization Identification number.

Variable	Type	Value
TIMESTAMP	Timestamp	Current Date and Time.
`USER_ID`	String	User ID of current user.
`USER_ROLES`	Array	Roles the user belongs to.

Workitem Variables

Each step in the workflow has dependent criteria, and requires specific variables to be defined.

The following table lists variables, their types, and values.



Workitem variables

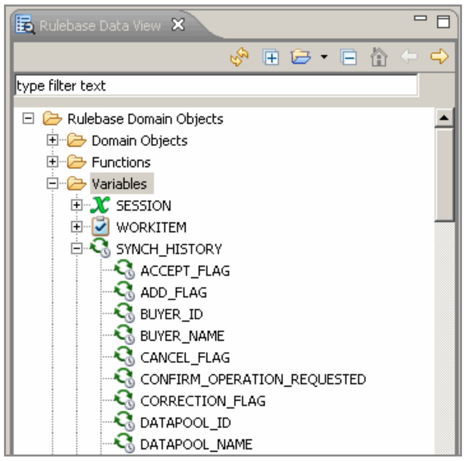
Variable	Type	Value
`ACTIVITY_NAME`	String	Current activity name.
`SEVERITY`	Number	Workitem severity.

Variable	Type	Value
STEP_SEVERITY	String	Step severity.
DOCTYPE	String	Document Type that created workitem.
DOCSUBTYPE	String	Document Sub-Type that created workitem.
ERRORS	Number	Number of errors in record bundle.
WARNINGS	Number	Number of warnings in record bundle.
REJECTIONS	Number	Number of rejections in record bundle.
TRADING_PARTNER	String	Trading Partner Name.
TRADING_PARTNER_TYPE	String	Trading Partner Organization Type.
MARKETPLACE_NAME	String	Marketplace name.
MASTER_CATALOG_NAME	String	Name of the master catalog of the record being processed.
MASTER_CATALOG_VERSION	Number	Master catalog version.
INTENT	String	Intent passed in to WorkItem activity.
RECORD_COUNT	Number	Total number of records in the bundle.
SUCCESS_COUNT	Number	Number of records with no errors.
Custom*	String	Any Parameter starting with "Custom" that is passed to Workitem Activity.

Synch History variables

Synch history variables are used to access record synchronization history.

These variables are active only during synchronization.



Synch History variables

Variable	Description
`IS_DISCONTINUED`	Is Discontinued
`IS_ADDED`	Is_Added
`HAS_BASE_ATTRIBUTE_SET_CHANGED`	Has_Base_Attribute_Set_Changed
`HAS_ATTRIBUTE_SET_CHANGED`	Has_Attribute_Set_Changed
`HAS_PARTNER_ATTRIBUTE_SET_CHANGED`	Has_Partner_Attribute_Set_Changed
`IS_PUBLISHED`	Is_Published
`IS_PUBLISHED_TO_ANY_PARTNER`	Is_Published_To_Any_Partner
`IS_LINKED`	Is_Linked
`IS_DELETED`	Is_Deleted

Variable	Description
`IS_REVIEWED`	Is_Reviewed
`IS_ACCEPTED`	Is_Accepted
`IS_REJECTED`	Is_Rejected
`IS_SYNCHRONIZED`	Is_Synchronized
`HAS_PUBLISHED_RELATION_CHANGED`	Has_Published_Relation_Changed
`HAS_LINKED_RELATION_CHANGED`	Has_Linked_Relation_Changed
`HAS_CHILDREN`	Has_Children
`HAS_PREVIOUSLY_PUBLISHED_PARENT`	Has_Previously_Published_Parent
`HAS_ANY_CHILD_CHANGED`	Has_Any_Child_Changed
`HAS_INPROGRESS_SYNC_EVENT`	Has_Inprogress_Sync_Event
`IS_ROOT_RECORD`	Is_Root_Record
`IS_PARENT`	Is_Parent
`IS_CHILD`	Is_Child
`IS_ADD_REQUESTED`	Is_Add_Requested
`IS_CORRECTION_REQUESTED`	Is_Correction_Requested
`IS_DELETE_REQUESTED`	Is_Delete_Requested
`IS_PUBLISH_REQUESTED`	Is_Publish_Requested
`IS_RELOAD_REQUESTED`	IS_RELOAD_REQUESTED
`IS_CANCEL_REQUESTED`	Is_Cancel_Requested

Variable	Description
`IS_DISCONTINUE_REQUESTED`	Is_Discontinue_Requested
`IS_INCREMENTAL_REQUESTED`	Is_Incremental_Requested
`IS_ACCEPT_REQUESTED`	Is_Accept_Requested
`IS_REJECT_REQUESTED`	Is_Reject_Requested
`IS_REVIEW_REQUESTED`	Is_Review_Requested
`IS_SYNCHRONIZE_REQUESTED`	Is_Synchronize_Requested
`VALID_NEXT_OPERATIONS`	Valid_Next_Operations
`OPERATION_REQUESTED`	Operation_Requested
`MASTERCATALOG_ID`	Mastercatalog_Id
`MASTERCATALOG_NAME`	Mastercatalog_Name
`BUYER_ID`	BUYER_ID
`BUYER_NAME`	Buyer_Name
`TRADING_PARTNER_ID`	Trading_Partner_Id
`TRADING_PARTNER_NAME`	Trading_Partner_Name
`TRADING_PARTNER_TYPE`	Trading_Partner_Type
`DATAPOOL_ID`	Datapool_Id
`DATAPOOL_NAME`	Datapool_Name
`INCREMENTAL_FLAG`	Incremental_Flag
`RELOAD_FLAG`	Reload_Flag

Variable	Description
`ADD_FLAG`	Add_Flag
`CORRECTION_FLAG`	Correction_Flag
`DELETE_FLAG`	Delete_Flag
`PUBLISH_FLAG`	Publish_Flag
`CANCEL_FLAG`	Cancel_Flag
`DISCONTINUE_FLAG`	Discontinue_Flag
`ACCEPT_FLAG`	Accept_Flag
`REJECT_FLAG`	Reject_Flag
`REVIEW_FLAG`	Review_Flag
`SYNCHRONIZE_FLAG`	Synchronize_Flag
`CONFIRM_OPERATION_REQUESTED`	Confirm_Operation_Requested
`ROOT_ADD_OPERATION`	Root_Add_Operation
`ROOT_LINK_OPERATION`	Root_Link_Operation
`ROOT_UNLINK_OPERATION`	Root_Unlink_Operation
`ROOT_PUBLISH_OPERATION`	Root_Publish_Operation

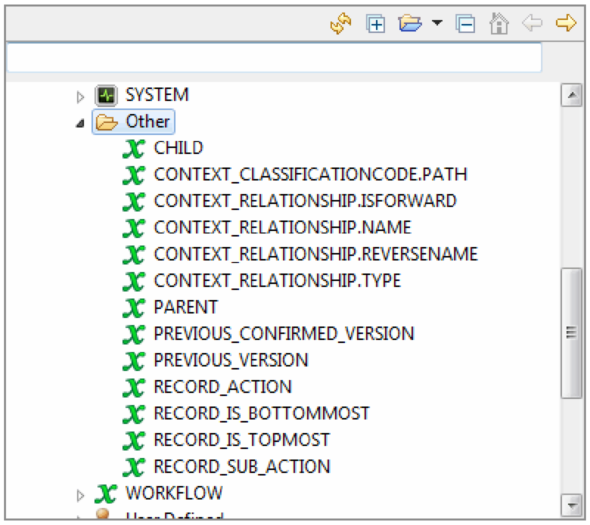
System Variables

System Variables

Variable	Description
`STATE`	Gets the product state of the current record.
`RECORD_VERSION`	Gets the product version of the current record.
`RECORD_ACTION`	Gets current record action of the record.
`RECORD_CHECKSUM`	Gets current record checksum.
`RECORD_LAST_MODIFIED_ON`	Gets current record last modified time.
`RECORD_CREATED_DATE`	Gets current record creation date.
`RECORD_ACTIVE_FLAG`	Gets current record active flag.
`RECORD_LAST_MODIFIED_BY`	Gets modify member ID of the member who modified current record in last.
`RECORD_KEY`	Gets internal bundle key. Contains product key ID and catalog ID.
`RECORD_ID`	Gets record ID for the current record.
`RECORD_IDEXT`	Gets productkey ID and Ext for the current record.
`RECORD_KEYID`	Gets ProductKey ID for the current record.
`CATALOG_NAME`	Gets the name of the catalog for the current record.
`RECORD_IS_ROOT`	Gets information on whether the record is the root of the bundle.

These variables are used to access values for system attributes of records.

Other Variables


SYSTEM
Other
CHILD
CONTEXT_CLASSIFICATIONCODE.PATH
CONTEXT_RELATIONSHIP.ISFORWARD
CONTEXT_RELATIONSHIP.NAME
CONTEXT_RELATIONSHIP.REVERSENAME
CONTEXT_RELATIONSHIP.TYPE
PARENT
PREVIOUS_CONFIRMED_VERSION
PREVIOUS_VERSION
RECORD_ACTION
RECORD_IS_BOTTOMMOST
RECORD_IS_TOPMOST
RECORD_SUB_ACTION
WORKFLOW

Child

This variable allows to access attribute values of child record during relationship catalog rulebase execution. This context variable is only available in relationship catalog rulebase execution. The parent/child record is always determined in context of forward relationship.

Context Classification Code Path

This variable is used to get the classification code path in context with current record.

Context Relationship IS Forward

This variable specifies whether the relationship is a forward relationship in context with current record.

Context Relationship Name

This variable is used to get the relationship name in context with current record.

Context Relationship Reverse Name

This variable is used to get the reverse relationship name in context with current record.

Context Relationship Type

This variable is used to get the relationship type in context with current record.

Parent

This variable allows to access attribute values of a parent record during relationship catalog rulebase execution. This context variable is only available in relationship catalog rulebase execution. The parent/child record is always determined in context of forward relationship.

Record Sub Action

Currently, the only possible value that can be assigned to this variable is RESTORE. When a record is restored, for all UI based rulebase validations for restored record, RECORD_SUB_ACTION is set to RESTORE. RECORD_SUB_ACTION is bound to RESTORE only for UI based validations and not during workflow processing.

Previous Version and Previous Confirmed Version

You can access unconfirmed and confirmed record versions with two explicitly defined contexts.

Previous Version and Previous Confirmed Version

Context	Description
`PREVIOUS_VERSION`	Latest confirmed or unconfirmed version.
`PREVIOUS_CONFIRMED_VERSION`	Last confirmed version.

Record Action

This variable is set during Record Add, Edit, or Copy functions.

Record Action

Record Action	Escaped Version
`ADD`	New record is being added.
`EDIT`	Existing record is being edited.
`COPY`	New Record is being copied from another record.
`VIEW`	Record is being viewed.

Record is Topmost

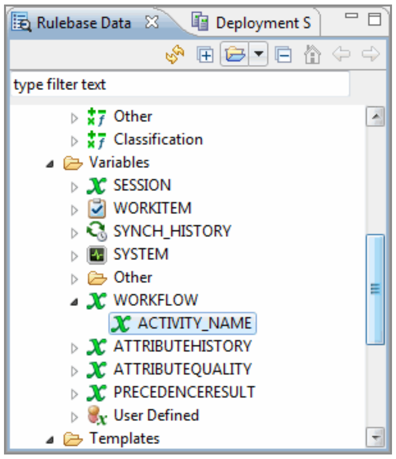
This variable allows you to check whether the record is the topmost in the hierarchy.

Record is Bottommost

This variable allows to check whether the record is bottommost in the hierarchy.

Workflow Variables

Workflow variables are used to access values of workflow.

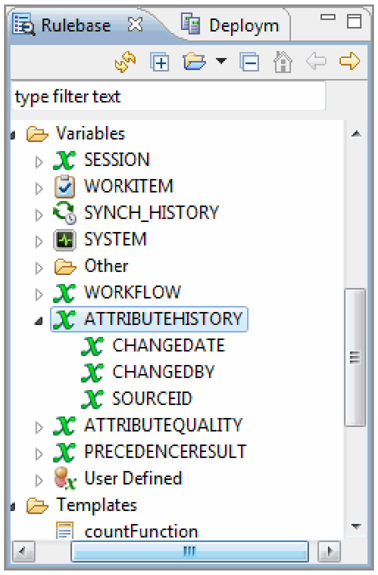


Workflow Variable

Variable	Type	Value
ACTIVITY_NAME	String	Activity Name.

Attribute History Variable

Attributes history variable is used to access the attribute history.

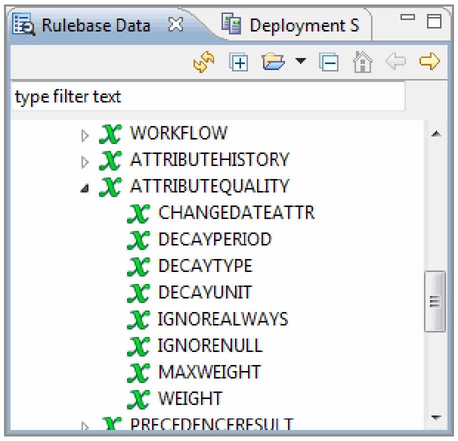


Attribute History Variable

Variable	Description
CHANGEDATE	Date when the attribute values were changed.
CHANGEBY	Name of the person who made the change.
SOURCEID	Source from where the data came from.

Attribute Quality Variables

Attribute quality variables are used to access the attribute quality.



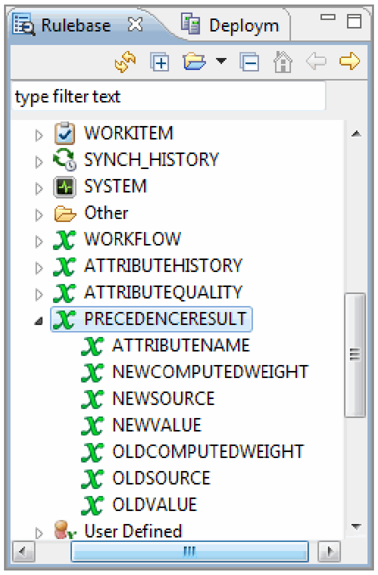
WORKFLOW
ATTRIBUTEHISTORY
ATTRIBUTEQUALITY
CHANGEDATEATTR
DECAYPERIOD
DECAYTYPE
DECAYUNIT
IGNOREALWAYS
IGNORENULL
MAXWEIGHT
WEIGHT
PRECEDENCERESULT

Attribute Quality Variable

Variable	Description
CHANGEDATEATTR	Use this attribute as change date instead of system generated change date.
DECAYPERIOD	Period of decay
DECAYTYPE	Decay Types. The available types are None, Linear, Half life.
DECAYUNIT	Unit of Measurement of period.
IGNOREALWAYS	Always ignore this attribute from this source.
IGNORENULL	If value is null, the weight is considered as zero (0).
MAXWEIGHT	Maximum weight.
WEIGHT	Weight of the attribute quality.

Precedence Result Variable

Precedence result variable is used to access the result of the precedence.



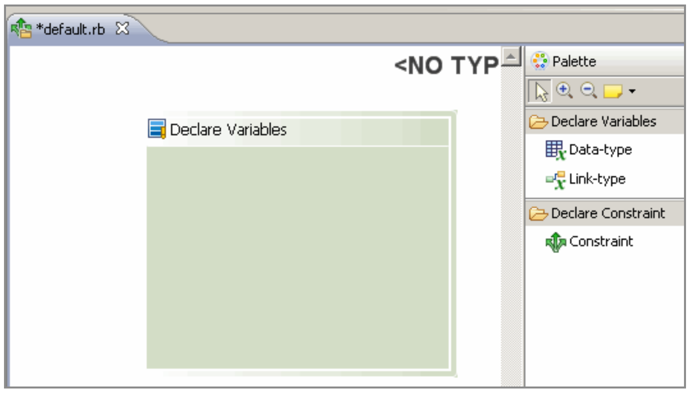
Precedence Result Variable

Variable	Description
ATTRIBUTENAME	The name of the attribute for which precedence is defined.
NEWCOMPUTEDWEIGHT	The new computed weight.
NEWSOURCE	Source from where the data came from.
NEWVALUE	The attribute’s new value.
OLDCOMPUTEDWEIGHT	The Old computed weight..
OLDSOURCE	The name of the previous source.
OLDVALUE	The attribute's previous value.

Declaring Variables

The **Declare Variables** compartment in the Rulebase diagram is used to declare variables for the rulebase.

A single rulebase contains a single **Declare Variables** compartment; any number of variables can be created in this compartment.

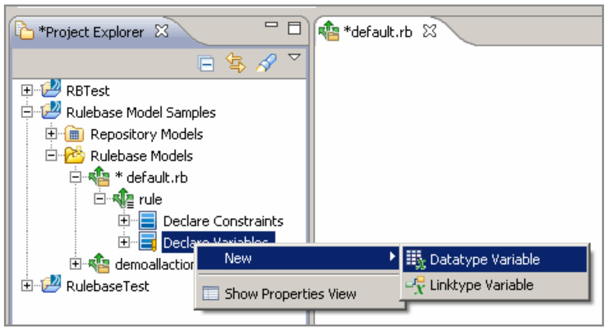


Through the Palette

Variables can be created by clicking the appropriate icon (**Data-type** or **Link-Type**) in the **Declare Variables** section of the palette and then clicking in the **Declare Variables** compartment in the Rulebase diagram.

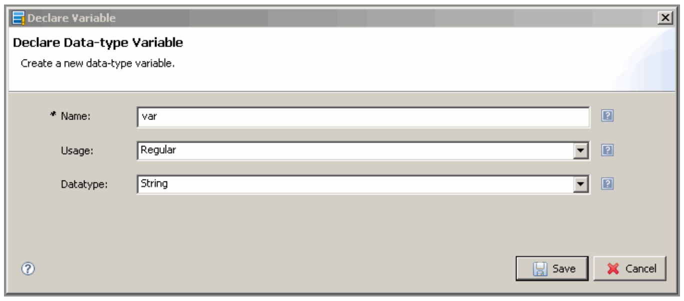
Through the Project Explorer

Variables can also be declared from the Project Explorer view.



To do this, expand the **Rulebase Models** folder, then expand the **.rul rulebase file** and the **rule** under it, and right click the **Declare Variables** node.

A dialog is displayed for variable creation with content similar to the Properties Tab (see Variable Properties).



* Name: var
Usage: Regular
Datatype: String

Editing Variables

The following can be done from the **Declare Variables** compartment.

- **Copy and paste variables**: To make a copy of a variable, select a variable and press **Control+C** and then **Control+V** to make a copy of it. Or select a variable, right click it, select **Edit > Copy** and then right click in the **Declare Variables** compartment and select **Edit > Paste.**
- **Reorder variables**: To reorder a variable, select a variable and drag to reorder its position within the variable declaration compartment.
- **Delete variables**: To delete a variable, select it and press **Delete** on the keyboard. Or select a variable, right click it and select **Delete from Model.**

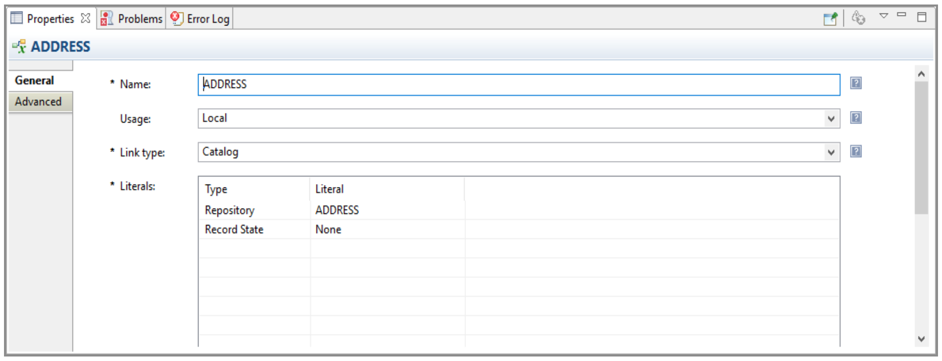
Variable Properties

Clicking a variable (in the **Declare Variables** compartment) displays its details in the **Properties** Tab.

Properties for Link type variables

This section describes the **General** and **Advanced** properties of the Link type variable.

The following details are displayed in the **General** tab and can be edited:

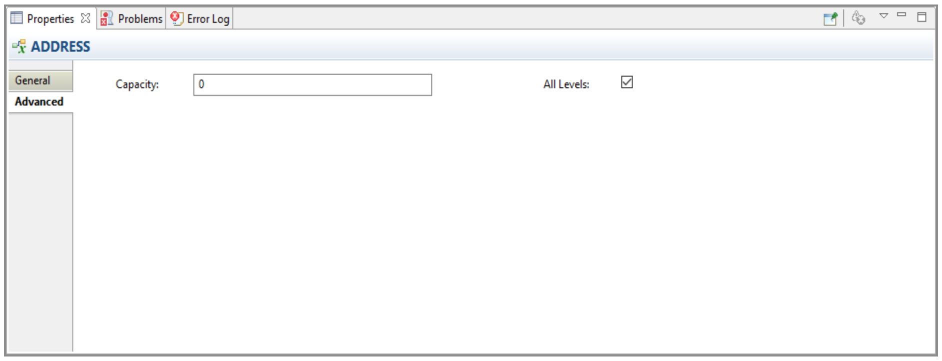


* Name: ADDRESS
Usage: Local
* Link type: Catalog
* Literals:

Type	Literal
Repository	ADDRESS
Record State	None

- **Name**: Names are case insensitive and cannot contain spaces and special characters (such as -,+,/,\,*).
- **Usage**: Describes the type of variable. Usage can be:
    - **Input**: Input by a user.
    - **Local**: Assigned through a rulebase.
    - **Output**: Local variable exported to caller.
    - **Regular**: From a catalog or session.
- **Linktype**: Link type of the variable (datasource, record, relationship, and so on)
- **Literals**: Literals are references to meta data (repository or datasource) which are bound to declared variables. For Catalog link type, the record state is also displayed whether CONFIRMED or UNCONFIRMED.

The following details are displayed in the **Advanced** tab and can be edited:



Capacity: 0
All Levels: ☑

- **Capacity**: Specify the capacity.
- **All Levels**: Select the check box to handle all related records at all levels for

relationship_record link type. The **All Levels** flag supports only "Self" relationship.

Properties for Data type variables

The following details are displayed and can be edited:

- Name: Names are case insensitive and cannot contain spaces and special characters (such as -,+,/,\,*).
- Usage: Describes the type of variable. Usage can be:
    - **Input**: Input by a user.
    - **Local**: Assigned through a rulebase.
    - **Output**: Local variable exported to caller.
    - **Regular**: From a catalog or session.
- **Datatype**: Data type of the variable (string, boolean, number, and so on)
- Variable references: Displays variable usage details.

Constraints

A constraint contains a condition and an action. The condition describes when the rule needs to be applied.

The action describes what the rule actually does and controls what attributes the rule is applicable to.

The **Declare Constraints** compartment is a default part of the Rulebase diagram.

Adding Constraints

Constraints can be added by clicking the Constraint icon.

Constraint icon in the **Declare Constraint** section of the Palette and then clicking in the **Declare Constraints** compartment of the Rulebase Diagram.

Editing Constraints

The following can be done from the **Declare Constraints** compartment:

- **Copy and paste constraints**: To make a copy of a constraint, select a constraint and press **Control+C** and then **Control+V** to make a copy of it. Or select a constraint, right click it, select **Edit > Copy** and then right click in the **Declare Constraints** compartment and select **Edit > Paste.**
- **Reorder constraints**: To reorder a constraint, select a constraint and drag to reorder its position within the **Declare Constraints** compartment.

Note: The order of constraints is of significance since constraints are executed sequentially.

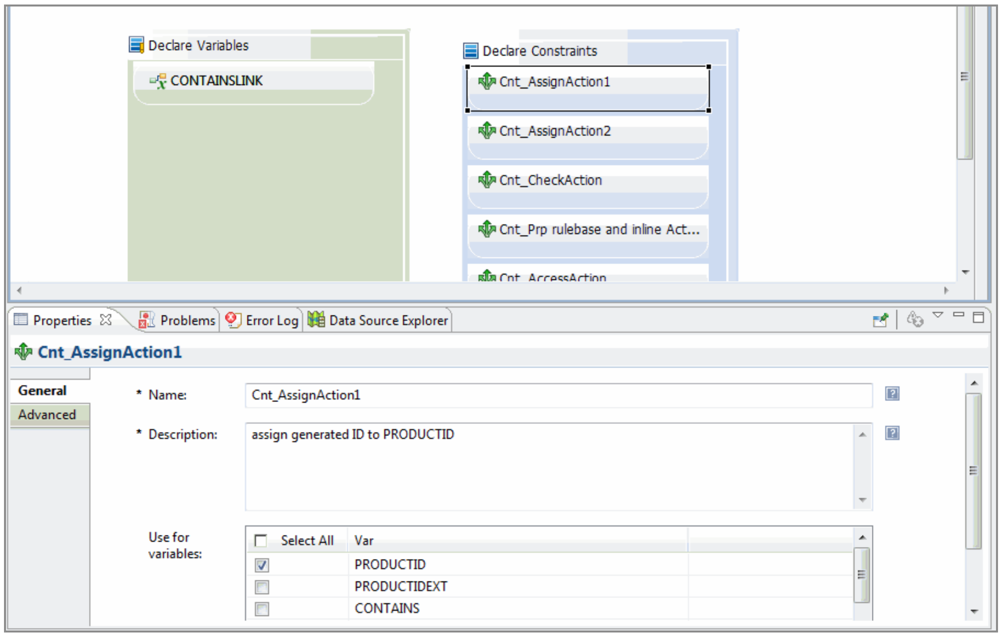
- **Delete constraints**: To delete a constraint, select it and press **Delete** on the keyboard. Or select a constraint, right click it and select **Delete from Model.**

Constraint Properties

Click an added constraint to see its properties in the **Properties** Pane.

The following properties are displayed for constraints in the Properties Pane, General tab:

- **Name**: Any logical name to identify the constraint.
- **Description**: A short text description of the constraint.
- **Use for Variables**: This specifies which attribute(s) the rule applies to.



Cnt_AssignAction1

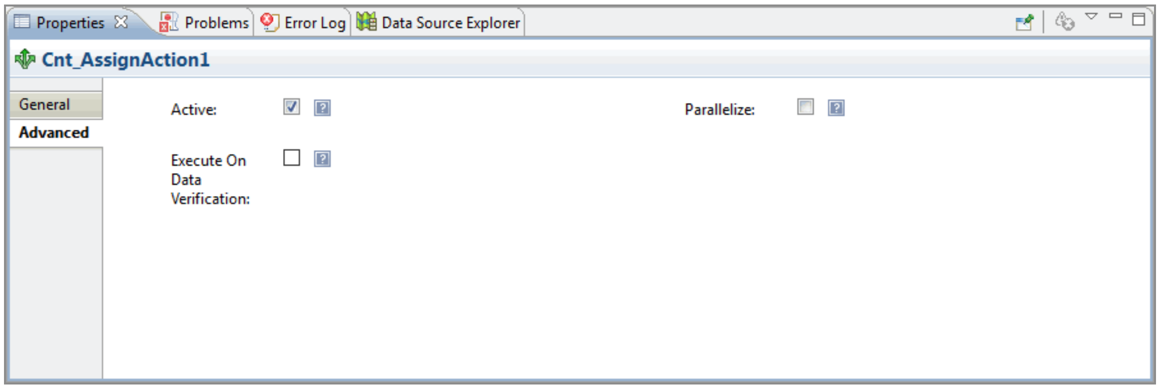
* Name: Cnt_AssignAction1

* Description: assign generated ID to PRODUCTID

Use for variables:

Select All	Var
✓	PRODUCTID
☐	PRODUCTIDEXT
☐	CONTAINS

The following properties are displayed for constraints in the Properties pane of theAdvanced tab:



Cnt_AssignAction1

Active: ✓

Parallelize: ☐

Execute On Data Verification: ☐

- **Active**: Indicates the active constraint to be evaluated during rulebase execution. To disable the constraint, uncheck the active flag.

- **Parallelize**: Indicates that this constraint does not depend on order of execution and can be executed in parallel with other constraints in rulebase.
- **Execute On Data Verification**: Identifies that the particular constraint is defined for data verification and should be run only when the data verification is executed.

Double click a constraint in the main rulebase diagram to open up the Constraint diagram which lists Conditions and Actions. For details, see Expressions and Access Action.

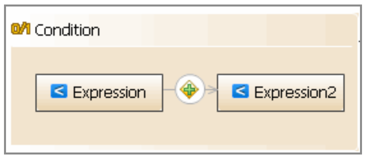
Conditions

The condition compartment in your rulebase design allows definition of expressions for the constraint.

Use the palette to add expressions in the condition compartment.

Conditions contain:

- **Expressions**: The actual expression logic.
- **Group Expression**: Expressions grouped together.
- **Link Expressions**: AND and OR operators to connect expressions and group expressions.



Working with Decision Tables

However, the condition values may vary from constraint to constraint.

The decision table allows you to manage scenarios where rules and condition templates are same, only values of the conditions changes. Decision table defines a tabular format structure similar to a Microsoft Excel sheet. Each row in the decision table represents a rule. The columns in the decision table are divided into conditions column and actions column. The actions in the action columns are same as the rulebase action.

The decision table is created in an existing rulebase file and then configured with the respective properties. The decision table configuration involves:

- Setting up the name and description
- Setting up the number of rows initially required
- Setting up the number of action columns required
- Defining and selecting the condition columns and operators
- Reordering the condition columns

Create a Decision Table

The process of creating a Decision table is same as creating Constraints. The Decision table is created in the Declare Constraint container.

After creating the Decision table their properties are defined. The mandatory fields in the properties section are denoted by asterisks. The decision table properties are as follows:

- **Name**: While creating the decision table a default name Dt, Dt1, Dt2, and so on is automatically populated. When a decision table is created for the first time the first instance in the constraint container has the name as Dt, the second instance has Dt1, the third instance has Dt2 and so on. The **Name** field is mandatory.
- **Description**: The decision table description is also populated with default description Dt, Dt1, Dt2, and so on. When a decision table is created for the first time, the first instance in constraint container has the description as Dt, the second instance has

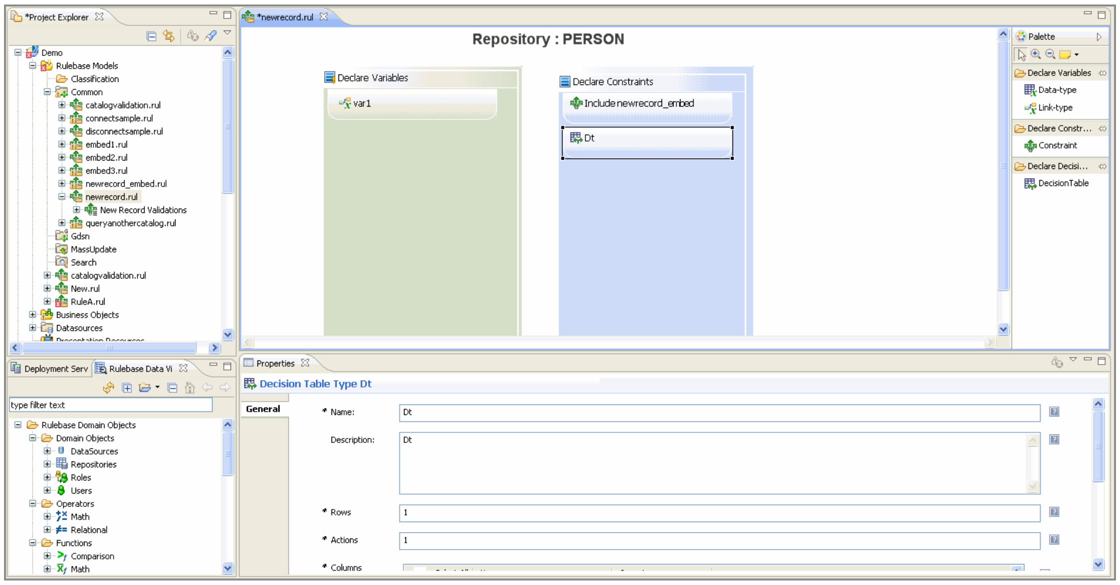
Dt1, the third instance has Dt2 and so on.

- **Rows**: The number of rows which should be available initially in a decision table is defined in the **Row** field. By default, it displays 1. There are two ways to add or delete rows in decision table. The **number of rows** field in the property section can be updated multiple times. Also addition or deletion of rows is done using the decision table editor. The **Rows** field is mandatory.
- **Actions**: The **number of actions** columns required in the decision table are defined in the **actions** field. By default, it displays 1. Maximum of ten action columns can be defined in a decision table. The actions column numbers can be updated at any time. If the updated action column number is less than the previously mentioned action column a warning message informing you that some of the existing actions columns would be deleted is displayed. The **Actions** field is mandatory.
- **Columns**: The condition columns for the decision table are defined in the columns field. The columns field is divided into three sections **Select All**, **Vars** and **Operators**. The check boxes are displayed in front of Variables for selection. Select the check boxes corresponding to the variables in the decision table.

Creating a Decision Table

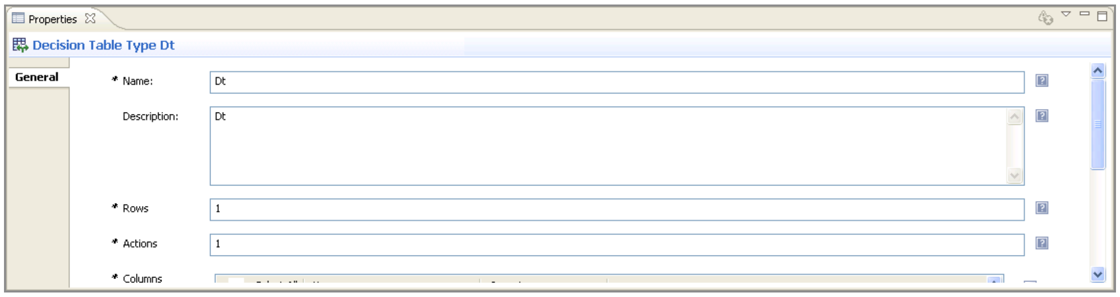
Procedure

1. Select the DecisionTable icon from the **Declare Decision Table** section of the **Palette** and click in the **Declare Constraint** compartment.



* Name:	Dt
Description:	Dt
* Rows	1
* Actions	1
* Columns	

2. The properties for the newly created decision table are displayed in the **Properties** section.



* Name:	Dt
Description:	Dt
* Rows	1
* Actions	1
* Columns	

3. Enter the appropriate Name in the **Name** field. By default, the name field is pre-populated as Dt, Dt1, Dt2, and so on. You can change the name as required. The **Name** field is mandatory.

4. Enter the appropriate description in the **Description** field. By default the description field is pre-populated with Dt, Dt1, Dt2, and so on. You can change the description as required.

5. Enter the number of rows required in the decision table in the **Rows** field. By default, the **Row** field displays 1 row. The rows can be appended using the decision table editor. If the number of rows mentioned in the **Row** field becomes less than the existing number of rows in the decision table editor, a message is displayed. If you decide to delete the rows, excessive rows in the decision table editor are deleted starting from the last row. The **Rows** field is mandatory.

6. Enter the number of actions column required in the decision table in the **Actions** field. By default, the **Action** field displays 1 action column. The decision table can have maximum of ten action column. If you decide to delete the action column, for example, if the number of action column is less than the previously mentioned action column, a message is displayed before deletion. The **Actions column** field is mandatory.

7. Select the appropriate condition columns for the decision table from the **Columns** field. By default, the first row is selected. Select the checkbox corresponding to the variables which should be part of the decision table. Select the appropriate operators to associate it with the selected variables. Based on the combination of Variable name from **Vars column** and operator image from **Operators column** the decision table column title is displayed. The conditions column is divided into three sub columns namely **Select All**, **Vars** and **Operators**.

   - **Select All**: The Select All sub column is used to select the entries that would appear as columns in the decision table. Check/Uncheck the checkbox to select and clear the selections. If you want to select all the checkboxes, check the **Select All** checkbox in the header. After the selection, if you try to uncheck the **Select All** checkbox, a warning message is displayed and all the checkboxes are cleared except for the first row checkbox. After selecting the checkbox, if you try to uncheck it, the following message is displayed This will delete column data for all the rows? Would you like to Proceed?

   - **Vars**: The variables appearing in the **Vars** sub column are derived from the repository that is linked to a given rulebase file and from Declare variable container in the rulebase file. The session variables are also displayed.

     Note: The variables with File data type and Link Type variables are not included in the Vars sub column.

   - **Operators**: The **Operators** sub column is used to associate a particular operator with a given variable in the **Vars** sub column. The **Operators** sub column provides a list of various operators. The operator listing varies depending upon the data type of the variable in the **Vars** sub column and whether variable is multi-value attribute or not. For more details on Operators, refer Operators.

8. After the variables and operators are defined, you can change the order of the condition columns. To change the order of the **conditions** columns, select an entry in the row and change the order by clicking the up arrow icon or the down arrow icon.

9. Click **Save**.

Operators

While defining the condition columns in the decision table you need to associate a particular operator with a given variable.

The **Operators** column provides a drop-down list which contains various operators. The operator listing varies depending upon the data type of the variable in the **Vars** column and whether variable is multi-value attribute or not.

The following operators are available in the drop-down list:

- Eq (equals)
- Neq (not equals)
- Gt (greater than)
- Geq (greater than or equal to)
- Lt (less than)
- Leq (less than or equal to)
- In
- Bet (Between)
- Contains
- Contains All
- Custom

Data Type - Operator Listings

Data Type - Operator Listings

Variable Data Type	Supported Operators
String	In, Eq, Neq, Custom
Number	In, Bet, Eq, Neq, Geq, Gt, Leq, Lt, Custom,
Boolean	Eq, Neq, Custom
Date	In, Bet, Eq, Neq, Geq, Gt, Leq, Lt, Custom,

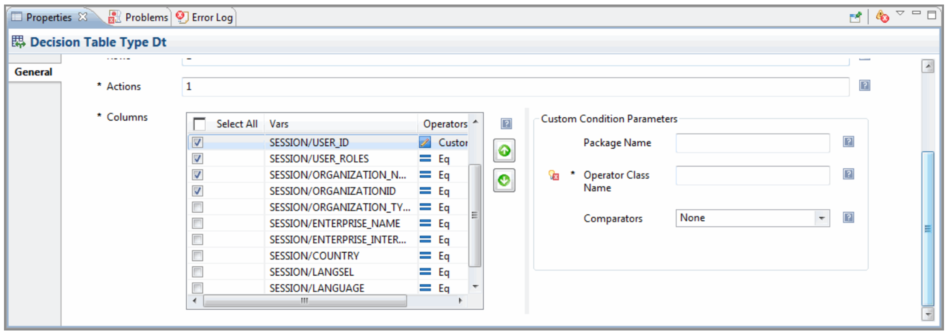
For multi-value attribute, following operators are supported

- Contains
- Contains All
- Eq
- Neq
- Custom

Custom Operator

If the available operators do not suffice the need, the custom operator can be used. The custom operator contains custom conditions which you can configure.

On selecting the custom operator the custom condition parameters are displayed.



* Actions: 1

* Columns

Select All	Vars	Operators
✓	SESSION/USER_ID	Custor
✓	SESSION/USER_ROLES	Eq
✓	SESSION/ORGANIZATION_N...	Eq
✓	SESSION/ORGANIZATIONID	Eq
	SESSION/ORGANIZATION_TY...	Eq
	SESSION/ENTERPRISE_NAME	Eq
	SESSION/ENTERPRISE_INTER...	Eq
	SESSION/COUNTRY	Eq
	SESSION/LANGSEL	Eq
	SESSION/LANGUAGE	Eq

Custom Condition Parameters

Package Name

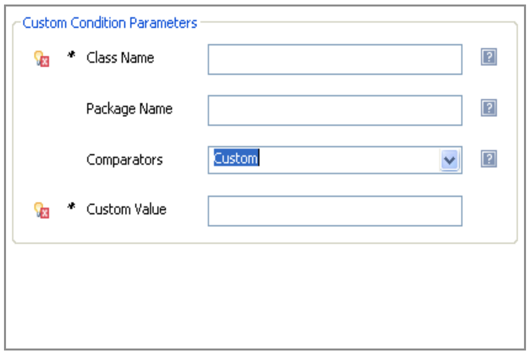
* Operator Class Name

Comparators: None

Enter the **Package Name**, **Operator Class Name** and select the **Comparator** from the drop-down list. The comparators are as follows:

- Boolean Comparator
- String Comparator
- Date Comparator
- Number Comparator
- MV Comparator
- None
- Custom

The Boolean, String, Date, Number and MV are the existing comparators. If you do not want to provide any comparator, select the None comparator. If you want to want customize and provide your own comparator, use the custom comparator. Enter your comparator in the custom value field.


Custom Condition Parameters

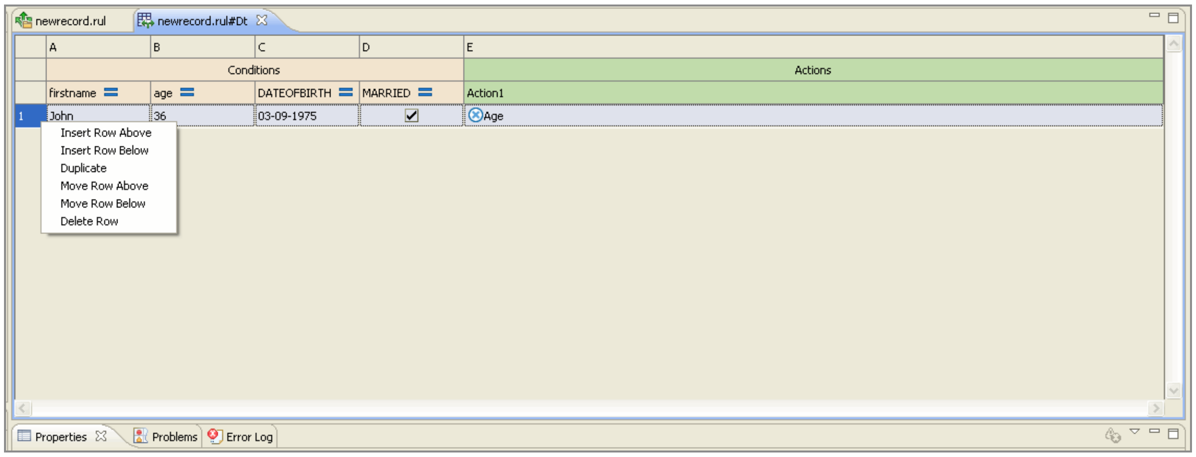
* Class Name	
Package Name	
Comparators	Custom
* Custom Value	

Decision Table Editor

Once you have created the decision table and configured the properties of the decision table, it is time to use the decision table.

On double clicking the newly created decision pallet, it opens the decision table editor.

The decision table editor UI appears similar to Microsoft Excel sheet.



	A	B	C	D	E
	Conditions				Actions
	firstname =	age =	DATEOFBIRTH =	MARRIED =	Action1
1	John	36	03-09-1975	✓	Age

The decision table displays the **Conditions and the Actions** columns. Each variable along with the operators symbol is displayed in the **Conditions** section. The Action section list the actions column configured during creation of decision table. By default, the number of rows displayed are as per the rows configured in the property section.

To add more rows to the decision table, select the row and right-click mouse. The following actions are displayed:

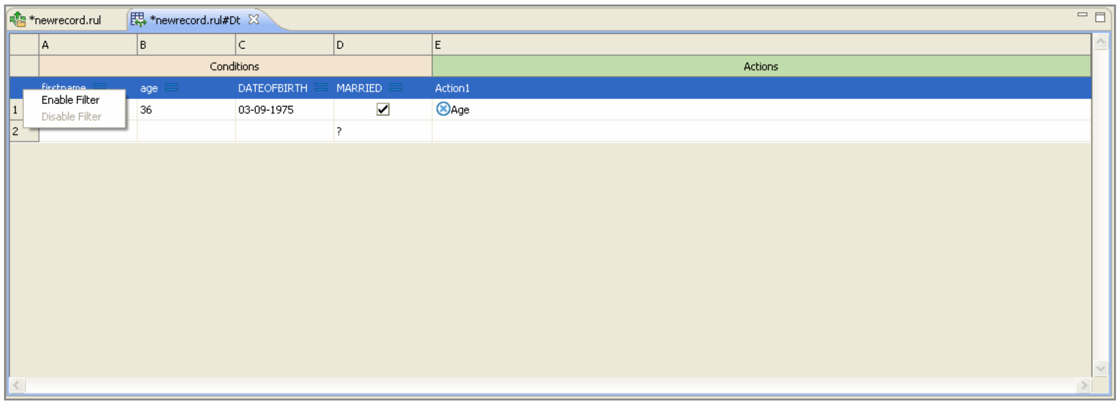
- **Insert Row Above**: Insert a row above the selected row.
- **Insert Row Below**: Insert a row below the selected row.
- **Duplicate**: Insert a duplicate row.
- **Move Row Above**: Move the selected row above the previous row. This is disabled for the first row.
- **Move Row Below**: Move the selected row below the next row.
- **Delete Row**: Delete a row.

Similarly, you can filter and sort the rows.

Filtering and Sorting Rows

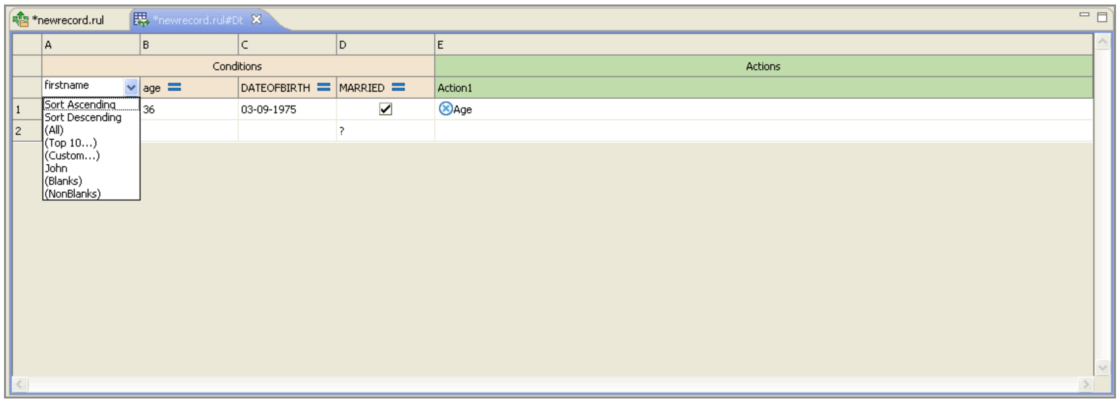
Procedure

1. Select the Header containing the variable name along with operator symbol and right-click mouse. Click **Enable Filter**.



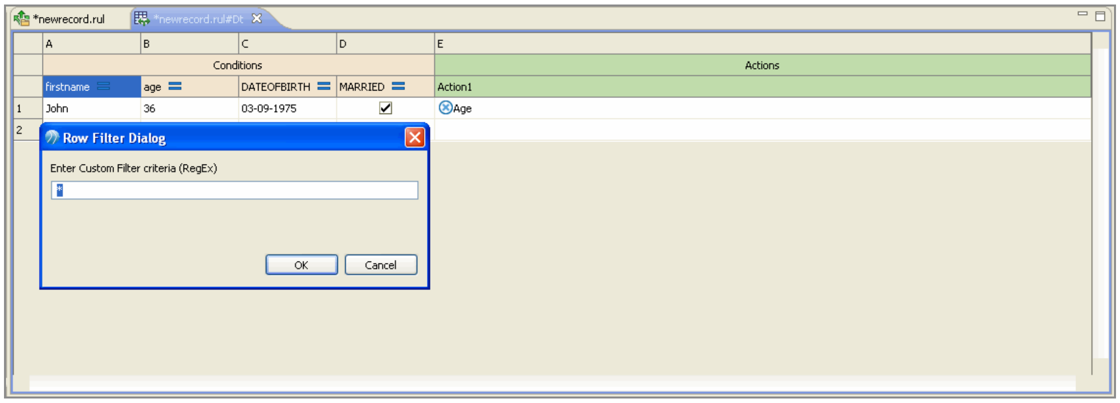
	A	B	C	D	E
	Conditions				Actions
	firstname	age	DATEOFBIRTH	MARRIED	Action1
1		36	03-09-1975	✓	Age
2				?	

2. Double-click the column header which you want to sort or filter. The decision table displays the drop-down list with the sort order and the filtering options.



	A	B	C	D	E
	Conditions				Actions
	firstname	age	DATEOFBIRTH	MARRIED	Action1
1		36	03-09-1975	✓	Age
2				?	

3. You can sort by ascending or descending order. You can filter by ALL, Top 10, By values, Blanks or NonBlanks, and by Custom filter. If you select custom filter, a custom **Row Filter Dialog** box is displayed.



	A	B	C	D	E
	Conditions				Actions
	firstname	age	DATEOFBIRTH	MARRIED	Action1
1	John	36	03-09-1975	✓	Age
2					

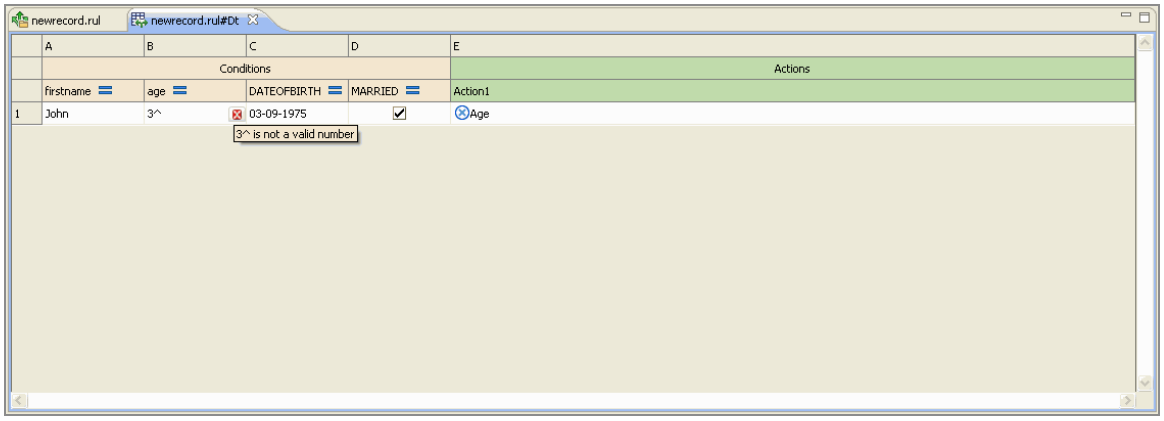
4. Enter the custom filter criteria in the **Enter Custom Filter Criteria (RegEx)** field. The filter criteria must be a regular expression.

5. Click **OK**.

Cell Validation

Each and every value entered in decision table is validated. If you enter or modify a value in the cell, a validation check is performed.

In case the validation check fails, a cross mark appears in the right corner of the cell. When you hover over the cross mark, a tool tip with the reason for validation failure is displayed.



	A	B	C	D	E
	Conditions				Actions
	firstname	age	DATEOFBIRTH	MARRIED	Action1
1	John	3^	03-09-1975	✓	Age

3^ is not a valid number

The following cell validations are performed:

- **Syntax Validation** -The syntax validation is performed while entering multiple values. A comma delimiter must be used while entering multiple values.
- **Validation of values based on operator** - The number of values entered in the text cell for a particular column depends upon the combination of type of operator used for that column and the variable (whether its multi-value or not). The table below shows the number of values required for various operators.

  Cell Validation

Operator	Number of Values required for multi-value attribute	Number of Values required for Non multi-value attribute
Eq	*	1

Operator	Number of Values required for multi-value attribute	Number of Values required for Non multi-value attribute
Contains	*	NA
ContainsAll	NA	NA
Customs	NA	NA
In	NA	*
Neq	*	1
Gt	NA	1
Lt	NA	1
Geq	NA	1
Leq	NA	1
Between	NA	2

- **Validation of values as per the variable data type** - After performing the Syntax and number of values validation, the variable data type validation is performed. The following date type formats are supported by decision table:
    - mm/dd/yyyy (default date format)
    - dd-mon-yyyy
    - mm/dd/yy
    - ddmmyyyy
    - yyyy-mm-dd
    - yyyy/mm/dd
    - dd-mm-yyyy
    - dd/mm/yyyy

- dd-mm-yy
- timestamp

Cell Skipping

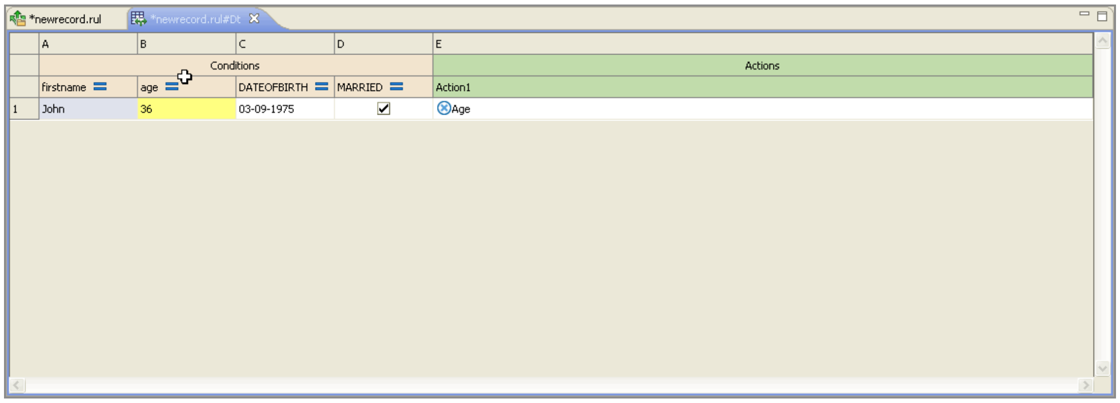
In decision table, skipping a cell means the condition which is of the form [Variable] [Operator] [cell Value] is not evaluated for a given row whose cell is skipped.

The cells which are skipped are marked in Yellow.

Skipping a Cell

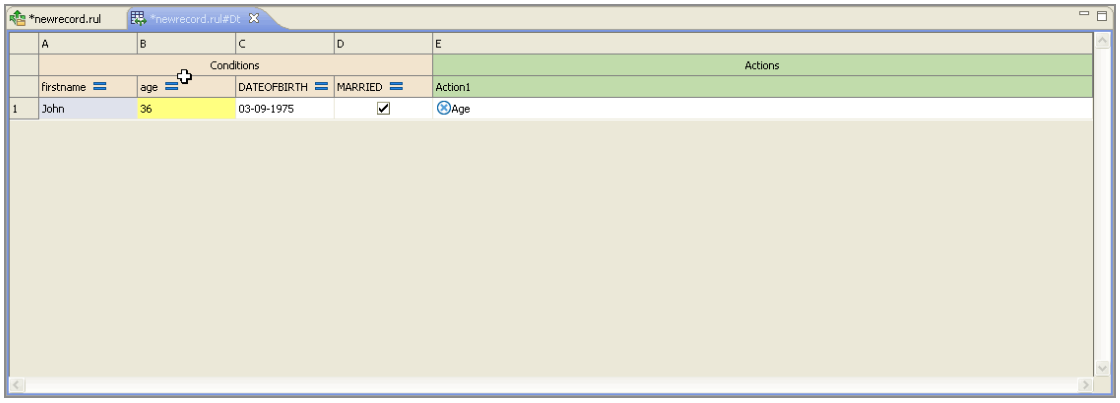
Procedure

1. Select the cell that you want to skip. For example, select the cell where age=36.



	A	B	C	D	E
	Conditions				Actions
	firstname =	age =	DATEOFBIRTH =	MARRIED =	Action1
1	John	36	03-09-1975	✓	Age

2. Right-click on the cell and select **Skip**. The cell which is skipped is marked in yellow.



	A	B	C	D	E
	Conditions				Actions
	firstname	age	DATEOFBIRTH	MARRIED	Action1
1	John	36	03-09-1975	✓	Age

3. To include a cell you have skipped earlier, select the cell marked in yellow, right-click on the cell, and select **Include**.

Decision Table Export

Decision table is exported as a part of the rulebase process export. During export, the following files are generated:

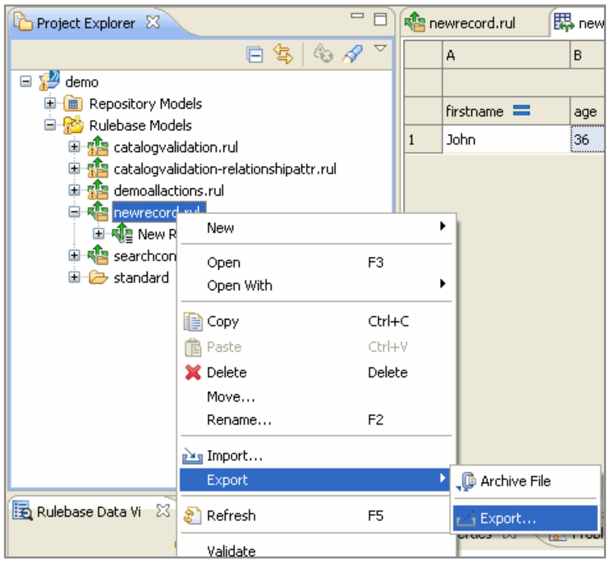
- **DT XML file**: The DT XML files contains all the conditions defined for each row in the decision table.
- **Actions XML file**: The Action XML file contains all the actions defined for each row in the decision table.
- `<rulebase filename>.xml` file: This is the main rulebase file.

You can export only those decision tables which do not have any validations errors in it.

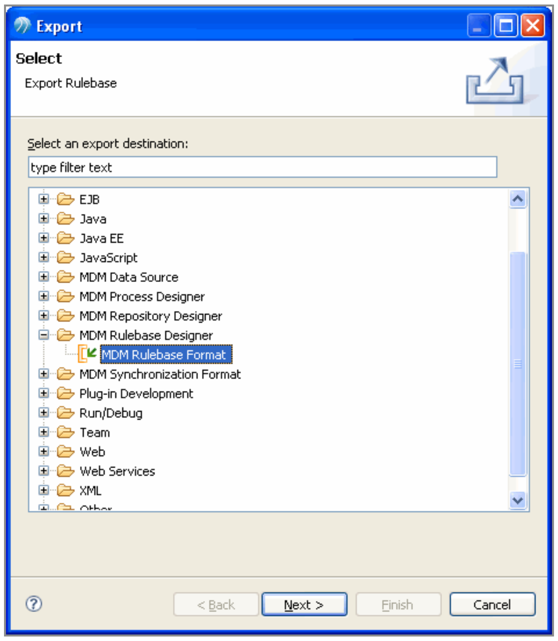
Exporting a Decision Table

Procedure

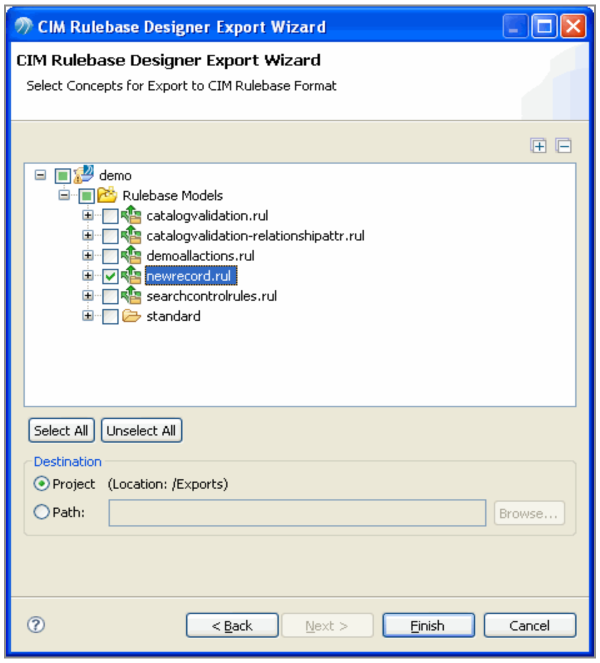
1. Right-click the **rulebase file** in the Project Explorer and click **Export > Export**.



2. The Export Wizard is displayed. Expand **MDM Rulebase Designer** from export destination list, select **MDM Rulebase Format**, and click **Next**.



3. Select the rulebase file which you want to export. By default, the rulebase file is exported to /Exports directory which is in the same project. You can change the location by specifying a different destination path. Select the **Path** option and browse to the folder in which you want to export the rulebase XML file and click **Finish**.


CIM Rulebase Designer Export Wizard

CIM Rulebase Designer Export Wizard

Select Concepts for Export to CIM Rulebase Format

- demo
  - Rulebase Models
    - [ ] catalogvalidation.rul
    - [ ] catalogvalidation-relationshipattr.rul
    - [ ] demoallactions.rul
    - [✓] newrecord.rul
    - [ ] searchcontrolrules.rul
    - [ ] standard

Select All | Unselect All

Destination

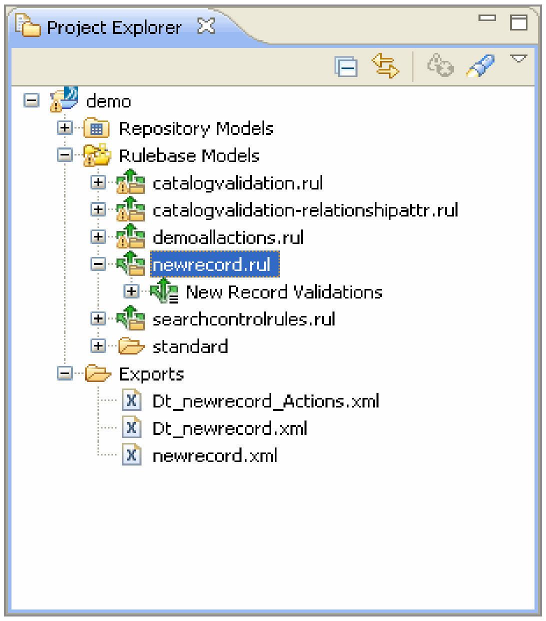
[✓ Project] (Location: /Exports)

Path:

Browse...

< Back | Next > | Finish | Cancel

4. The **Exports** folder in the explorer displays the DT XML, Actions XML, and the `<rulebase filename>.xml` files.


Project Explorer

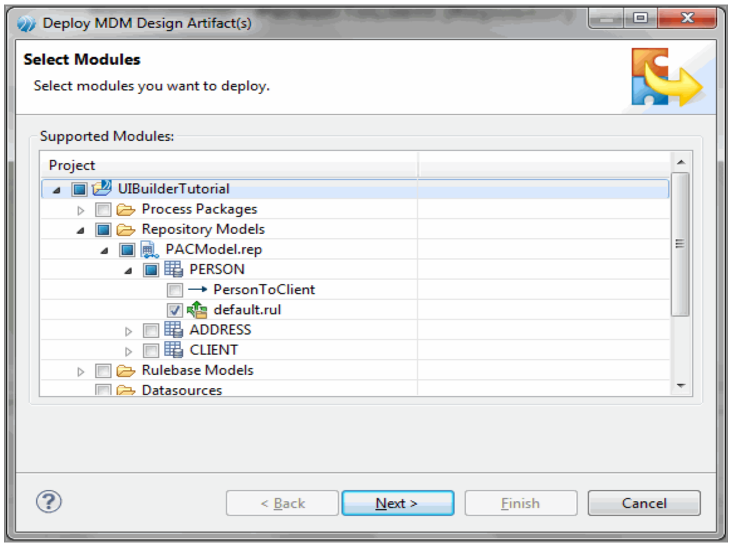
- demo
  - Repository Models
  - Rulebase Models
    - catalogvalidation.rul
    - catalogvalidation-relationshipattr.rul
    - demoallactions.rul
    - newrecord.rul
      - New Record Validations
    - searchcontrolrules.rul
    - standard
  - Exports
    - Dt_newrecord_Actions.xml
    - Dt_newrecord.xml
    - newrecord.xml

Direct Deploying the Decision Table

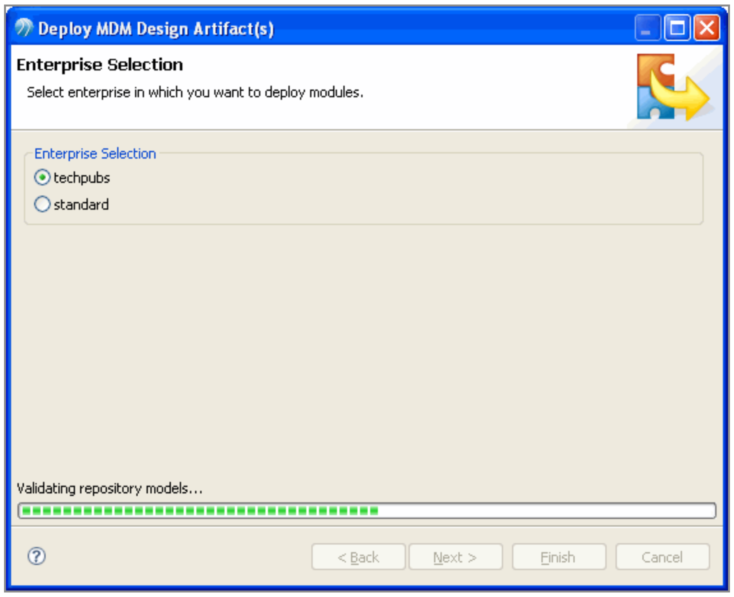
Procedure

1. In the Deployment pane, right click the < TIBCO MDM Server and select **Deploy Module.**

2. Select the rulebase module to deploy. Click **Next**.



3. Select the enterprise to deploy the rulebase to (either the current enterprise or standard). Click **Finish**.



Enterprise Selection

Select enterprise in which you want to deploy modules.

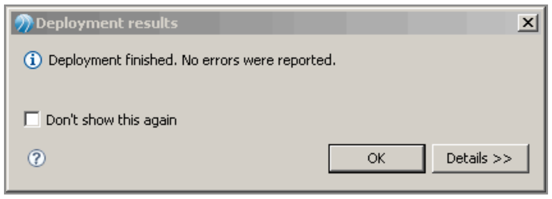
Enterprise Selection

(●) techpubs

( ) standard

Validating repository models...

On successful deployment a confirmation message is displayed. Similarly if an error is encountered and error is displayed.



Deployment finished. No errors were reported.

☐ Don't show this again

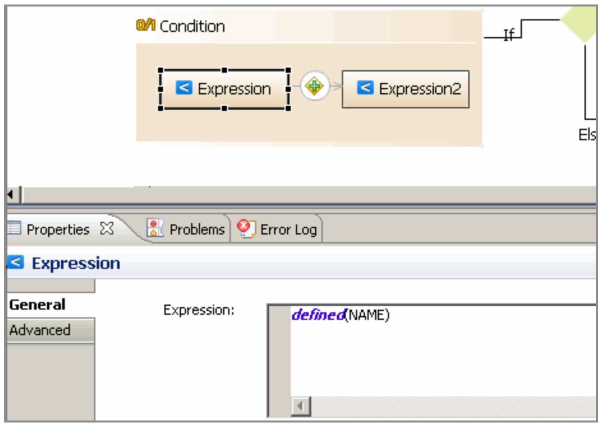
Result

Once successfully deployed, you can log onto the TIBCO MDM Server and check if your decision tables have got included.

Expressions

Note: Expressions are displayed in the Condition section of the Constraint diagram, accessed by double clicking constraints in the **Declare Constraints** compartment in the main rulebase diagram.

Expressions in a condition must evaluate to Boolean. Expressions created in the rulebase diagram have a corresponding property tab, which in turn contains an expression editor.



Expression: *defined*(NAME)

Creating Expressions

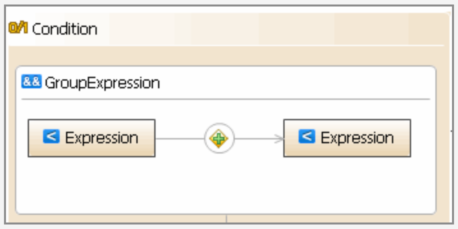
Expressions can be created using the following icons from the **Condition** section of the Palette and dropping in the Condition compartment:

- Expression
- Use this **Expression** icon to create standalone expressions.
- Group Expression
- Use this **Group Expression** icon to create group expressions.

For details, see Expression Editor Palette.

Group expressions allow for a bracket effect to the expression, for example (a==1 && (b==2 or c==5)). Group expressions can contain expressions and nested group expressions.

Note: Note: A group expression has two expressions per row and one group expression per row. Use **CTRL+SHIFT+F** to arrange the expression appropriately.



Expression Editor

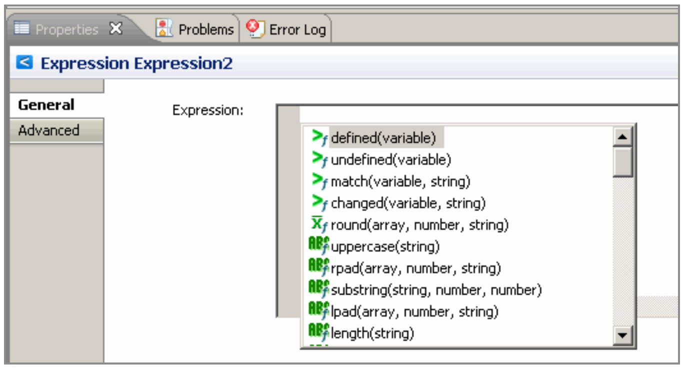
The Expression Editor provides user friendly expression definition functionality such as content assist, syntax completion, and syntax coloring (all functions and reserved words are colored).

Select the **Expression** box (within the condition) in the constraint diagram to see its properties in the **Properties** tab. On selecting the expression box, the **Properties** tab displays an **Expression** textbox to add, define, or modify the expression logic.

Content Assist

Click in the **Expression** textbox and use the **Ctrl+Spacebar** keyboard shortcut.

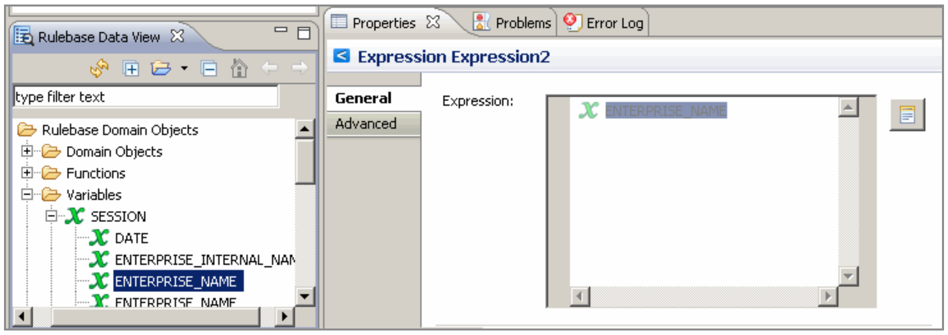
This activates context sensitive coding assistance and displays a list of applicable elements for the location content assist was activated for.



defined(variable)
undefined(variable)
match(variable, string)
changed(variable, string)
round(array, number, string)
uppercase(string)
rpad(array, number, string)
substring(string, number, number)
lpad(array, number, string)
length(string)

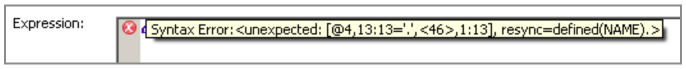
Use the mouse or the keyboard (Up Arrow, Down Arrow, Page Up, Page Down, Home, and End) to navigate and select elements in the list. Press Enter on a selected element in the list to insert the selection into the editor.

You can also drop and drop variables from the **Rulebase Data View** into the Expression Editor.



Syntax Errors

An error marker is displayed when there is a syntax error in the expression editor.



```
Syntax Error:<unexpected: [@4,13:13='.',<46>,1:13], resync=defined(NAME).>
```

Templates

Templates are a structured description of coding patterns that reoccur in the expression editor.

The expression editor supports the use of templates to fill in commonly used source patterns. Templates can be inserted using content assist (Ctrl+Space). Reoccurring expression syntax can be saves as a template by using the **Save as Template** button (to the right of the expression editor). On clicking this button, you will be prompted to provide a name for the template.

Note: While defining a constant string values use "Single Quote(')" in the expression editor.

Restrictions on SQL Expressions

The following restrictions apply for the use of SQL expressions:

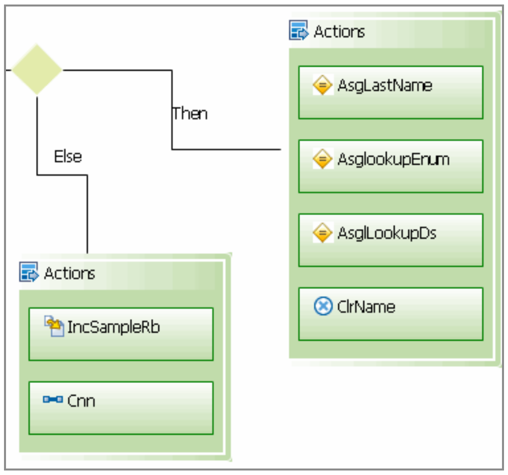
- The following logical expressions are supported: **and, or, not, in, defined, undefined, like**.
- The following operands are supported: **eq, neq, leq, lt, geq, gt**.
- You have to use a **where** clause if you have a table where source is sql.
- Multi-column results can only be used in **select** and **slice** actions. All other functions require a single column result (that is, a list of values).
- Built-in functions such as **concat**, **count**, and so on cannot be used within an SQL expression.
- In case of rulebase evaluation with relationship attribute values, the access modifiers on records based on relationship attribute are not applied on the record list. For example, if you try to hide some records from a record list based on their relationship attribute value for a particular relationship, the records are not hidden.
- The multi-value attribute can only be used in where clause and not supported in a select column list.
- Only supported operators for multi-value are **eq**, **neq**, **in** and **like**.

Actions

Note: Actions are displayed in the Constraint diagram, accessed by double clicking constraints in the **Declare Constraints** compartment in the main rulebase diagram.

Actions specify what a rule actually does. The action executed depends on whether the condition evaluates to **true** or **false**. Actions can be of two kinds: Then actions and Else actions.

The Constraint diagram contains **Then Actions** and **Else Actions** in appropriate Action containers, at the end of the logic flow.



The **Actions** group in the Palette contains icons for actions that can be dropped in the Then Actions or Else Actions sections. To define actions, select action icons from the palette and drop it in the **Actions** container.

Logic Implemented for Action

The following logic is implemented for Actions:

- If a condition is specified, a THEN Action is mandatory. ELSE Actions are optional.

- THEN Actions are executed if the expressions in the Condition section evaluate to true.
- ELSE Actions are executed if the expressions in the Condition section evaluate to false.
- If an ELSE action is specified, an associated condition is mandatory.
- A THEN Action can be specified by itself without any ELSE Action or associated Condition.
- Action tags (if specified within the Action compartment) are executed sequentially if the condition is true or if no conditions are present.
- In case of any undefined variables, the condition may not be evaluated and no actions will execute.

Access Action

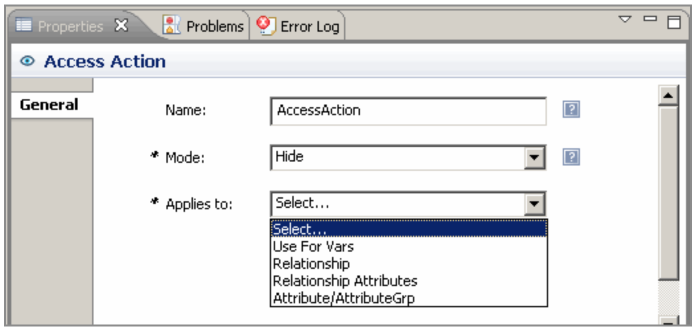
The **Access** action provides control over viewing, modifying, and visibility of Attributes, Attribute Groups and Records.

To define an Access action, click the Access icon.

Access action icon from the palette and drop it the appropriate (Else or Then) Action container.

Access Action Properties

The following properties can be provided for the Access Action in the **Properties** window, **General** tab:



Name: AccessAction

* Mode: Hide

* Applies to: Select...

Select...
Use For Vars
Relationship
Relationship Attributes
Attribute/AttributeGrp

- **Name**: Any logical name for the Access action.
- (Mandatory) **Mode**: The Mode controls the visibility and access to a record, attribute, or attribute group. The following values can be set here:
    - **View**: Attribute/Attribute group will appear in the UI as read-only.
    - **Hide**: Attribute/Attribute group will not be visible in the UI.
    - **Modify**: Attribute/Attribute group will be visible in the UI and can be modified by users.
    - **View Record**: Record will appear in record lists but cannot be modified.
    - **Hide Record**: Record will not appear in record lists or relationships.
    - **Modify Record**: Record will appear in record lists and can be modified.
    - **Hide Relationship**: Hide Relationship is used to hide the relationship from UI (RecordHierarchy). This mode works only with relationship catalog validations.

Note: View, Hide, and Modify modes apply for Attribute/AttributeGrp, Use for Vars, Relationship, Relationship Attributes.

View Record, Hide Record, and Modify Record apply for records.

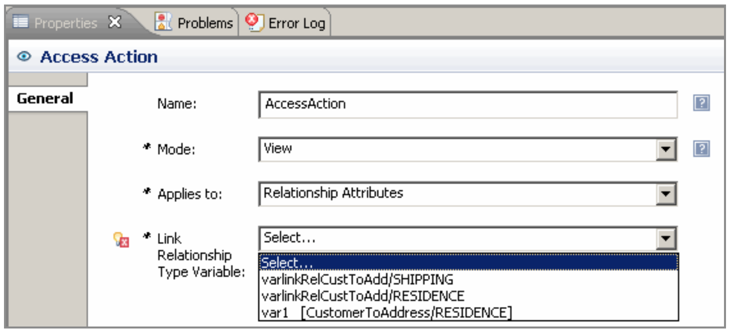
- (Mandatory)**Applies To**: This indicates what the Access action applies to. Values are:
    - Attribute/AttributeGrp
    - Use for Vars
    - Relationship
    - Relationship Attributes

Note: This option is disabled if the selected mode is View Record, Hide Record, Modify Record, Skip Merge, Allow Merge.

Access Action Validation

If **Applies To** is set to a value other then **Use For Vars**, it is mandatory to select the associated **Relationship Type Variable** or **Attribute Group**.

If the associated drop-down is not selected, an error marker with an appropriate error message is displayed.



Name: AccessAction

* Mode: View

* Applies to: Relationship Attributes

* Link Relationship Type Variable: Select...

Select...
varlinkRelCustToAdd/SHIPPING
varlinkRelCustToAdd/RESIDENCE
var1 [CustomerToAddress/RESIDENCE]

ApplyPrecedence Action Properties

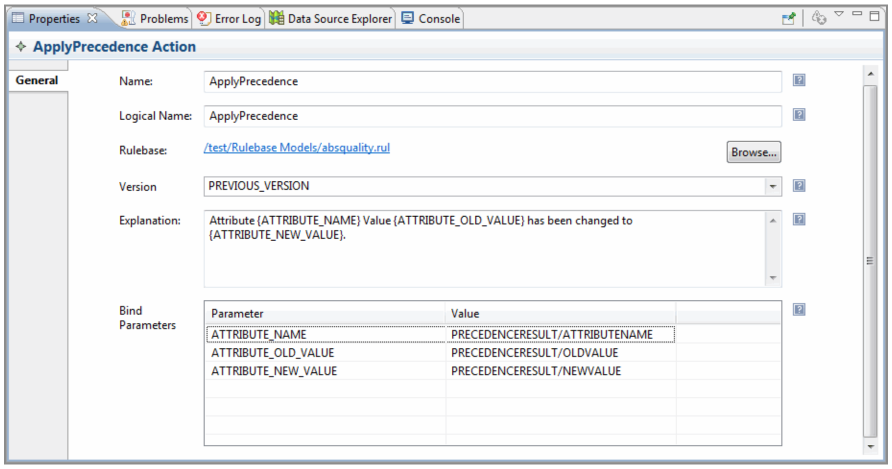
The following properties can be provided for the **ApplyPrecedence** Action in the **Properties** window.

General tab properties

- **Name**: Any name for the ApplyPrecedence Action, default name is ApplyPrecedence.
- **Logical Name**: Any logical name for the ApplyPrecedence Action.
- **Rulebase**: Browse and specify the rulebase to embed or type the relative path of the rulebase.
- **Version**: Select the version for ApplyPrecedence Action to identify with which version you want to compare. The available options are **Blank**, **PREVIOUS_VERSION**, and **PREVIOUS_CONFIRMED_VERSION**. The default value is blank.
- **Explanation**: Specify text to be displayed after applying precedence. Dynamic text can be specified using the place holder and then binding variables in Bind Parameter table.
    - To Bind Parameter specify the dynamic values, enter the text with place holders. For example: Attribute {ATTRIBUTE_NAME} Value {ATTRIBUTE_OLD_VALUE} has been changed to {ATTRIBUTE_NEW_VALUE}.
    - After entering the text, click anywhere on the property section, the Bind

Parameters table is populated with the specified parameters from the explanation text. For example, if you have entered the following in the explanation text field Attribute {ATTRIBUTE_NAME} Value {ATTRIBUTE_OLD_VALUE} has been changed to {ATTRIBUTE_NEW_VALUE}.

- **Bind Parameters**: The bind parameters table has four fields.
  - **Parameter**:- The name of the parameter is populated from the explanation text.
  - **Value**: Select the appropriate value from the **Value** drop-down list. The available options are **PRECEDENCERESULT/ATTRIBUTENAME, PRECEDENCERESULT/OLDSOURCE, PRECEDENCERESULT/OLDCOMPUTEDWEIGHT, PRECEDENCERESULT/OLDVALUE, PRECEDENCERESULT/NEWSOURCE, PRECEDENCERESULT/NEWCOMPUTEDWEIGHT, PRECEDENCERESULT/NEWVALUE**.



Name: ApplyPrecedence

Logical Name: ApplyPrecedence

Rulebase: /test/Rulebase Models/absquality.rul

Version: PREVIOUS_VERSION

Explanation: Attribute {ATTRIBUTE_NAME} Value {ATTRIBUTE_OLD_VALUE} has been changed to {ATTRIBUTE_NEW_VALUE}.

Bind Parameters

Parameter	Value
ATTRIBUTE_NAME	PRECEDENCERESULT/ATTRIBUTENAME
ATTRIBUTE_OLD_VALUE	PRECEDENCERESULT/OLDVALUE
ATTRIBUTE_NEW_VALUE	PRECEDENCERESULT/NEWVALUE

ApplyPrecedence Action Validation

The following validations are performed at the time of Export.

- The rulebase embed path file must have a valid path or empty field.
- Each Parameter must have value assigned to it or parameter value must not have an empty or null value.

Apply Precedence Action

The **ApplyPrecedence** action decides when the precedence should be applied.

To define a **ApplyPrecedence** action, select the ApplyPrecedence icon from the palette and drop it in the appropriate (Else or Then) Action container. The ApplyPrecedence action can be implemented only if the precedence management flag is set to true in the repository property section action icon from the palette and drop it the appropriate (Else or Then) Action container.

ApplyPrecedence Action Properties

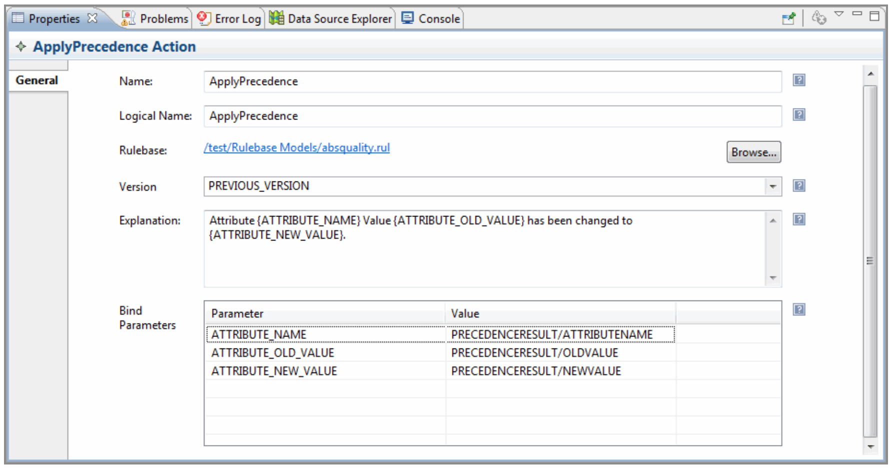
The following properties can be provided for the **ApplyPrecedence** Action in the **Properties** window.

General tab properties

- **Name**: Any name for the ApplyPrecedence Action, default name is ApplyPrecedence.
- **Logical Name**: Any logical name for the ApplyPrecedence Action.
- **Rulebase**: Browse and specify the rulebase to embed or type the relative path of the rulebase.
- **Version**: Select the version for ApplyPrecedence Action to identify with which version you want to compare. The available options are **Blank**, **PREVIOUS_VERSION**, and **PREVIOUS_CONFIRMED_VERSION**. The default value is blank.
- **Explanation**: Specify text to be displayed after applying precedence. Dynamic text can be specified using the place holder and then binding variables in Bind Parameter table.
    - To Bind Parameter specify the dynamic values, enter the text with place holders. For example: Attribute {ATTRIBUTE_NAME} Value {ATTRIBUTE_OLD_VALUE} has been changed to {ATTRIBUTE_NEW_VALUE}.
    - After entering the text, click anywhere on the property section, the Bind Parameters table is populated with the specified parameters from the explanation text. For example, if you have entered the following in the

explanation text field   Attribute {ATTRIBUTE_NAME} Value {ATTRIBUTE_OLD_VALUE} has been changed to {ATTRIBUTE_NEW_VALUE}.

- **Bind Parameters**: The bind parameters table has four fields.
    - **Parameter**:- The name of the parameter is populated from the explanation text.
    - **Value**: Select the appropriate value from the **Value** drop-down list. The available options are **PRECEDENCERESULT/ATTRIBUTENAME, PRECEDENCERESULT/OLDSOURCE, PRECEDENCERESULT/OLDCOMPUTEDWEIGHT, PRECEDENCERESULT/OLDVALUE, PRECEDENCERESULT/NEWSOURCE, PRECEDENCERESULT/NEWCOMPUTEDWEIGHT, PRECEDENCERESULT/NEWVALUE**.



ApplyPrecedence Action

Name:	ApplyPrecedence
Logical Name:	ApplyPrecedence
Rulebase:	/test/Rulebase Models/absquality.rul
Version	PREVIOUS_VERSION
Explanation:	Attribute {ATTRIBUTE_NAME} Value {ATTRIBUTE_OLD_VALUE} has been changed to {ATTRIBUTE_NEW_VALUE}.

Bind Parameters

Parameter	Value
ATTRIBUTE_NAME	PRECEDENCERESULT/ATTRIBUTENAME
ATTRIBUTE_OLD_VALUE	PRECEDENCERESULT/OLDVALUE
ATTRIBUTE_NEW_VALUE	PRECEDENCERESULT/NEWVALUE

ApplyPrecedence Action Validation

The following validations are performed at the time of Export.

- The rulebase embed path file must have a valid path or empty field.
- Each Parameter must have value assigned to it or parameter value must not have an empty or null value.

Assign Action

The **Assign** action allows for assignation of values to variables.

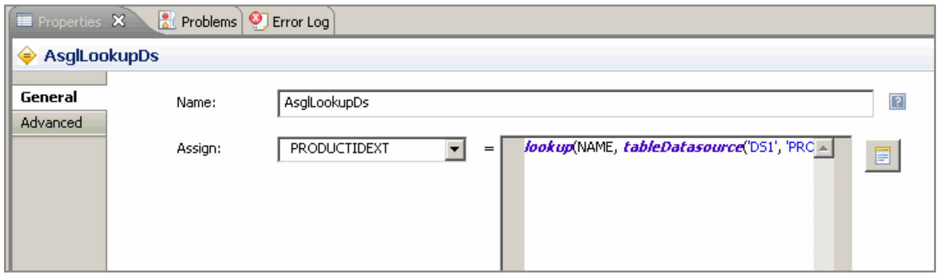
Values can be assigned to declared variables and repository attributes.

To define an **Assign** action, select the Assign **Assign** action icon from the palette and drop it in the appropriate (Else or Then) Action container.

Assign Action Properties

The following properties can be provided for the Assign Action in the **Properties** window, **General** tab:

- **Name**: Any logical name for the Assign action.
- **Assign**: This displays a list of all variables, and provides an expression editor to type the expression. Select the variable to be assigned in the **Assign** drop down and specify the expression in the expression editor to the right of the assigned variable.



Name: AsglLookupDs

Assign: PRODUCTIDEXT = lookup(NAME, tableDatasource('DS1', 'PRC

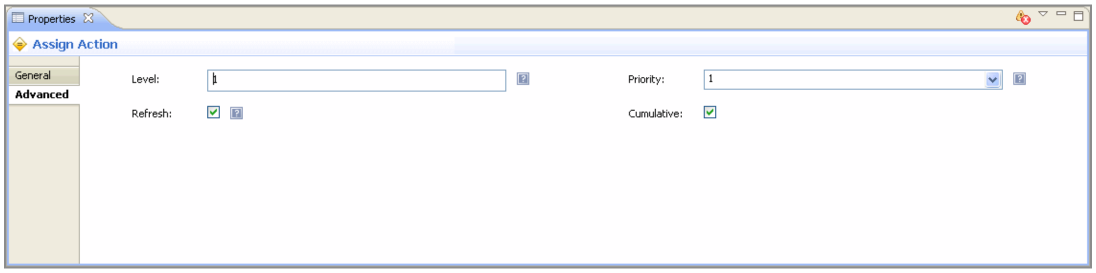
The expression can be a simple mathematical expression like (a+b*(c+d)) or as complex as assigning an array {1,2,3} or ({'abc', 'xyz'}, 2) or tableSql({col1,col2}, where col1='John' and col2='D').

Note: If you are assigning a true or false value to a boolean type variable, the value must be written in lower case otherwise the assignment does not work correctly. For example, true. This is treated as constant value assignment.

Advanced tab properties

- **Level**: For details, see Level.
- **Refresh**: For details, see Refresh.

- **Priority**: For details, see Priority.



Level: 1
Priority: 1
Refresh: ✓
Cumulative: ✓

Assign Action Validation

If the assignment variable (in the **Assign** drop-down) is not selected and if there are errors in expression, an error marker is shown on the figure.

Assign Identity Action

The **Assign Identity** action allows you to identify an external key and map it to the internal key.

If the data that is being imported does not have PRODUCTID and PRODUCTIDEXT, use the Assign Identity action.

The Assign Identity action can also be used when data is modified from web services or from User Interface. However, the assign identity rule should only be used when data is being saved for the first time, that is when TIBCO MDM has not already assigned an identity and product key to it.

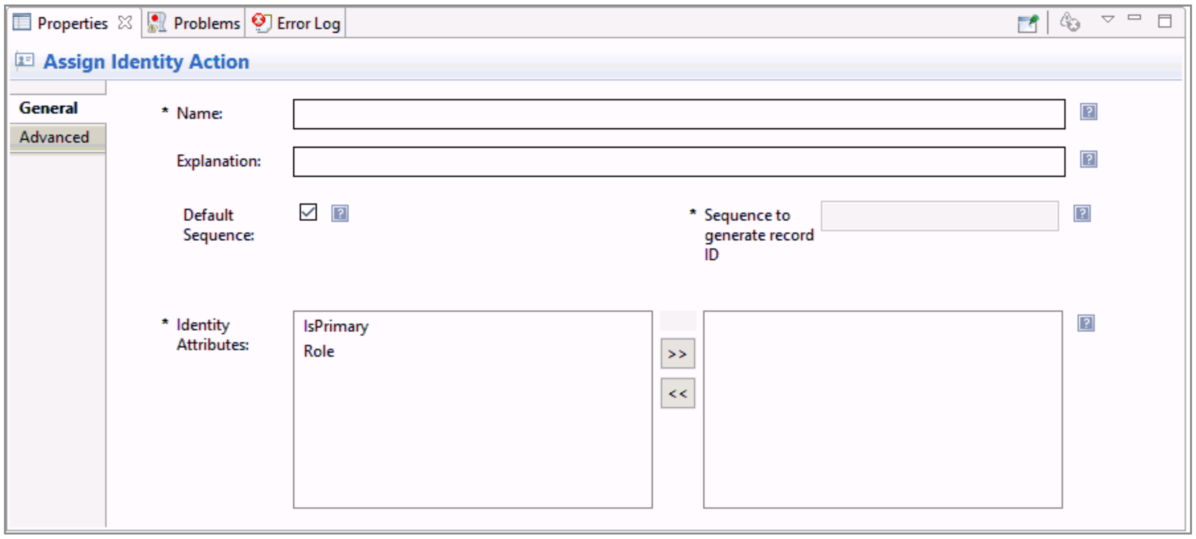
If Assign Identity rule is used during new record creation, and any existing matching record is found, the add will be rejected as duplicate. The assign Identity rule does not work with SaveRecord activity in workflow as product ID and Ext are required for this activity to merge the incoming data with existing data.

To define an **Assign Identity** action, select the Assign Identity **Assign Identity** action icon from the palette and drop it in the appropriate (Else or Then) Action container.

Assign Identity Action Properties

The following properties can be provided for the Assign Identity Action in the Properties window, **General** tab:

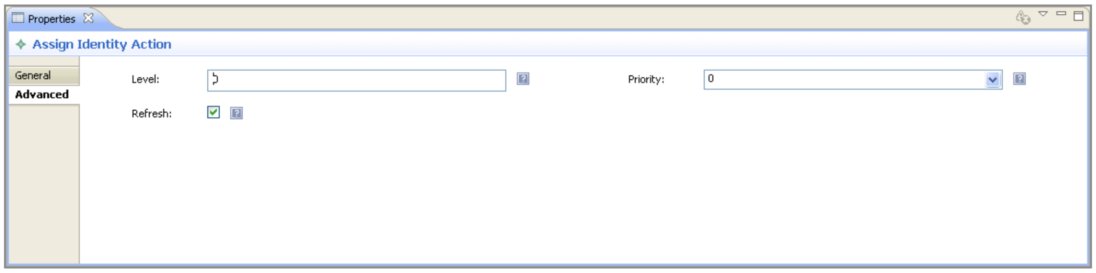
- (Mandatory)**Name**: Any logical name for the AssignIdentity action.
- **Explanation**: A detailed description of the AssignIdentity action.
- **Default Sequence**: Indicates to use default repository sequence to generate record ID. By default, the check box is selected. Clear the check box to use another sequence specified in the **Sequence to generate record ID** field. If the default sequence check box is selected, the `%DEFAULT_SEQUENCE%` variable in the sequence tag is exported or deployed.
- **Sequence to generate record ID**: The Sequence in which the record ID is generated.
- (Mandatory)**Identity Attributes**: Select the attribute which you want as the business key to uniquely identify the record. The multi value attribute cannot be specified as business key.



Advanced tab properties

- **Level**: For details, see Level.
- **Refresh**: For details , see Refresh.

- **Priority**: For details, see Priority.



Level:

Refresh: ✓

Priority: 0

Assign Identity Action Validation

If the **Name**, **Sequence to generate record ID**, and **Identity Attributes** are not selected and if there are errors, an error marker is shown on the figure.

Categorize Action

This action allows to categorize the record, it can categorize the record in one or more categorizes.

The categorize action supports incremental flag, by default incremental flag value is set to "True".

To define an **Categorize** action, select the Categorize **Categorize** action icon from the palette and drop it in the appropriate (Else or Then) Action container.

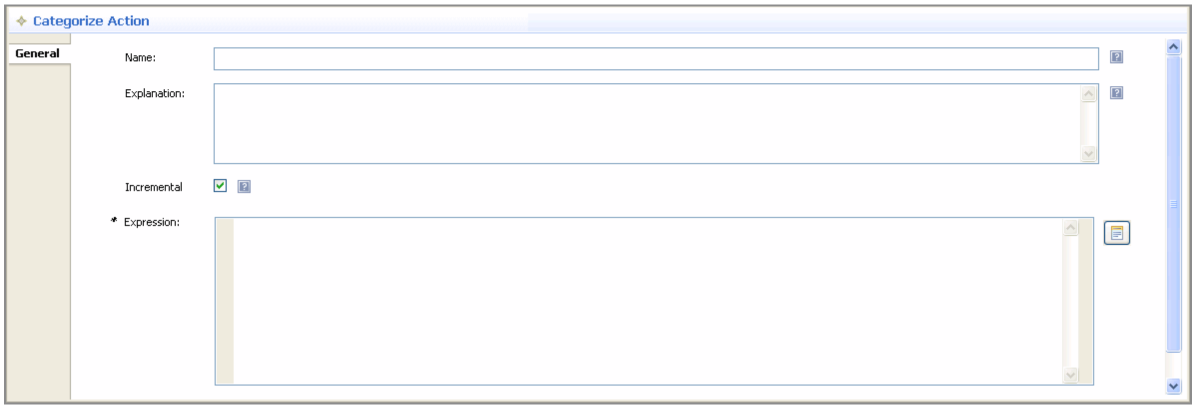
Categorize Action Properties

The following properties can be provided for the Categorize Action in the **Properties** window, **General** tab:

- **Name**: Any logical name for the Categorize action.
- **Explanation**: A detailed description of the categorize action.
- **Incremental**: True - to keep record’s existing classification along with the current action. False - to remove record’s existing classification and to categorize it into

provided classification codes in this action.

- (Mandatory) **Expression**: Input parameters must be an array of link types classification code or any expression that evaluates to classification code. For example, the following are the input parameters for categorize action: getClassificationCodeByCode(CODEVAR_SOCIAL, 'P07') - This is a classification function that returns classification code. CODEVAR_DESKTOP - This is link type classification code. CODEVAR_DB - This is link type classification code.



Categorize Action Validation

If the **Expression** variable is not specified and if there are errors in expression, an error marker is shown on the figure.

Check Action

The **Check** action evaluates an expression as **true** or **false**. If the expression is true, the attribute is in compliance. If it is false, the check failed and an explanation is displayed.

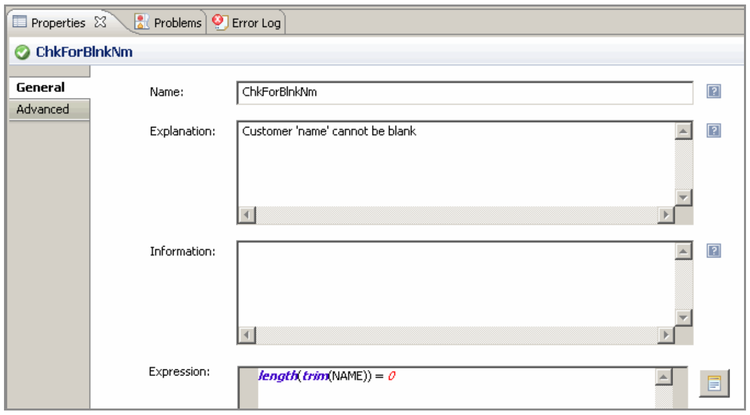
To define a **Check** action, select the Check **Check** action icon from the palette and drop it in the appropriate (Else or Then) Action container.

Check Action Properties

The following properties can be provided for the **Check** Action in the **Properties** window.

General tab properties

- **Name**: Any logical name for the Check action.
- **Explanation**: Specify text to be displayed if the expression evaluates to false.
- **Information**: Information about the check action.
- **Expression**: Expression that evaluates to true or false.



ChkForBlnkNm

Name: ChkForBlnkNm

Explanation: Customer 'name' cannot be blank

Information:

Expression: length(trim(NAME)) = 0

Advanced Tab Properties

- **Severity**: Set the Severity level for this action.

Check Action Validation

If there is an error in the expression, an error marker is shown on the figure.

Clear Action

The **Clear** action is used to clear a variable. Select the variable to be cleared.To define a **Clear** action, select the Clear **Clear** action icon from the palette and drop it in the appropriate (Else or Then) Action container.

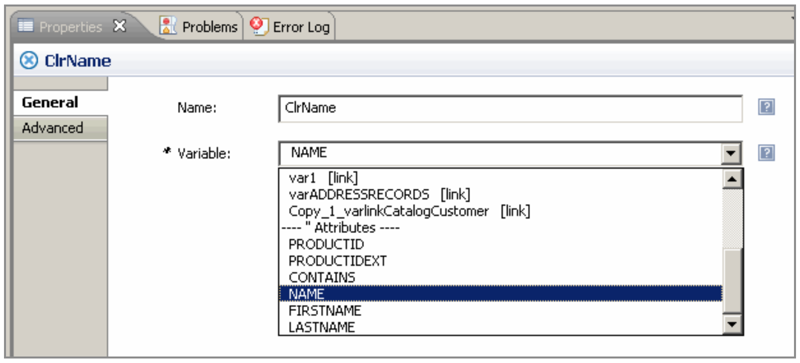
Clear Action Properties

The following properties can be provided for the **Clear** Action in the **Properties** window.

General tab properties

Name: Any logical name for the Check action.

Variable: Select the variable value to be cleared. Choose from a list of declared variables and repository attributes.



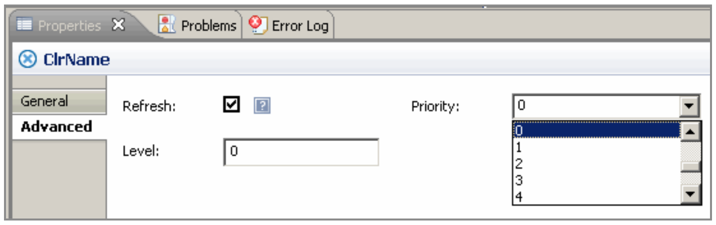
Name: ClrName

* Variable: NAME

```
var1   [link]
varADDRESSRECORDS   [link]
Copy_1_varlinkCatalogCustomer   [link]
---- " Attributes ----
PRODUCTID
PRODUCTIDEXT
CONTAINS
NAME
FIRSTNAME
LASTNAME
```

Advanced tab properties

- **Refresh**: Refreshes the variable.For details , see Refresh.
- **Priority**: For details, see Priority.
- **Level**: For details, see Level.



Refresh: ☑

Level: 0

Priority: 0

```
0
1
2
3
4
```

Clear Action Validation

If a variable is not selected from the drop-down list, an error marker is displayed on the figure to indicate that a variable to clear has not been selected.

Connect Action

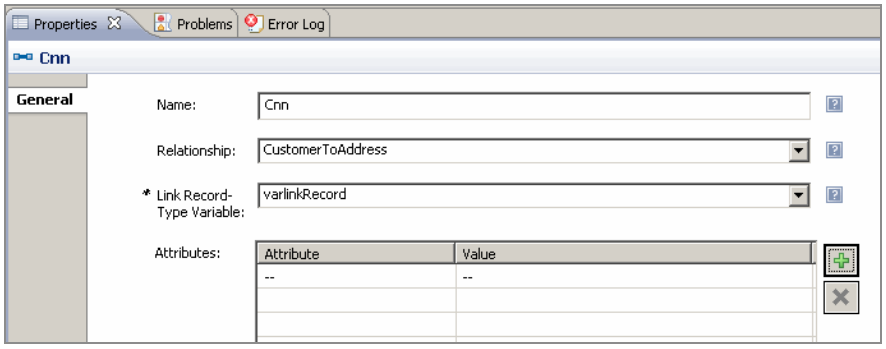
The **Connect** action is used to establish a relationship between records. It establishes a relationship between the record being processed and accessed records

To define a **Connect** action, select the Connect **Connect** action icon from the palette and drop it in the appropriate (Else or Then) Action container.

Connect Action Properties

The following properties can be provided for the **Connect** Action in the **Properties** window, **General** tab:

- **Name**: Any logical name for the Connect Action.
- **Relationship**: Select a Relationship name using which a relationship will be established. Relationships for the associated repository are displayed here.
- **Link Record -Type Variable**: Select from the list of Declared link variables (with Link type as Record) that are displayed here.
- **Attributes**: Click the appropriate icons to add or delete relationship attributes and provide values. Multiple relationship attributes can be passed to the Connect action.



Name: Cnn

Relationship: CustomerToAddress

* Link Record-Type Variable: varlinkRecord

Attributes:

Attribute	Value
--	--

Connect Action Validation

If the Relationship and Link Record-Type Variable are not specified, an error is displayed on the figure.

Disconnect Action

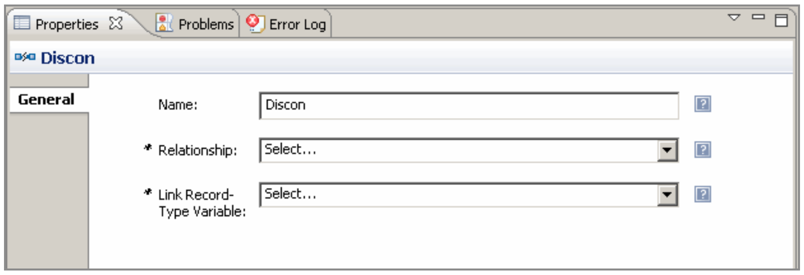
The **Disconnect** action is used to remove relationships between records. Any relationship attributes defined with the relationship are also removed.

To define a **Disconnect** action, select the Disconnect **Disconnect** action icon from the palette and drop it in the appropriate (Else or Then) Action container.

Disconnect Action Properties

The following properties can be provided for the **Disconnect** Action in the **Properties** window, **General** tab:

- **Name**: Any logical name for the Disconnect action.
- **Relationship**: Select the relationship to be deleted.
- **Link Record-Type Variable**: Select from the list of Declared link variables (with Link type as Record) that are displayed here.



Name:	Discon
* Relationship:	Select...
* Link Record-Type Variable:	Select...

Disconnect Action Validation

If the Relationship and Link Record-Type Variable are not specified, an error is displayed on the figure.

Include Action

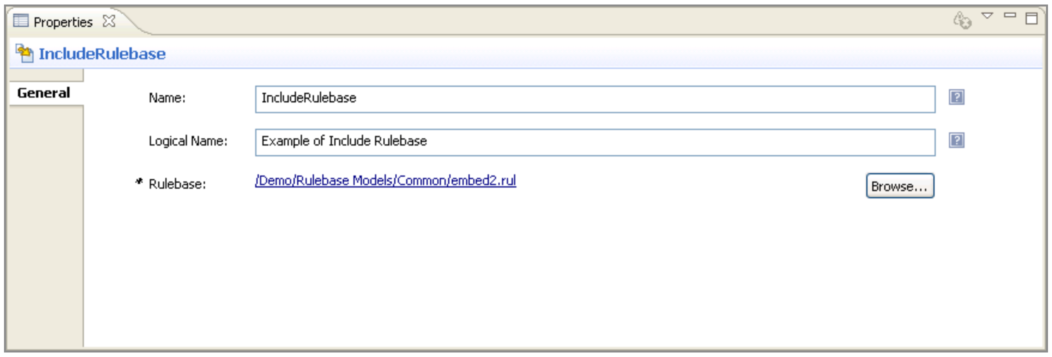
The **Include** action allows embedding one rulebase file into another.

To define a **Include** action, select the Include **Include** action icon from the palette and drop it in the appropriate (Else or Then) Action container.

Include Action Properties

The following properties can be provided for the **Include** Action in the **Properties** window, **General** tab:

- **Name**: Any name for the Include action.
- **Logical Name**: Any logical name for the Include action.
- **Rulebase**: Browse and specify the Rulebase to embed.



Name: IncludeRulebase

Logical Name: Example of Include Rulebase

* Rulebase: /Demo/Rulebase Models/Common/embed2.rul

Include Action Validation

If the relative path name for the Rulebase is not specified, an error is displayed on the figure.

Include Rulebase

A rulebase file can be included in another rulebase using include action.

Using the Browse button in the include action property section helps to include a rulebase file. The include rulebase allows you:

- To include a rulebase file present only in a workspace.
- To include a rulebase file in .rul format.
- To open up the included rulebase file in the rulebase editor which is available as a link.

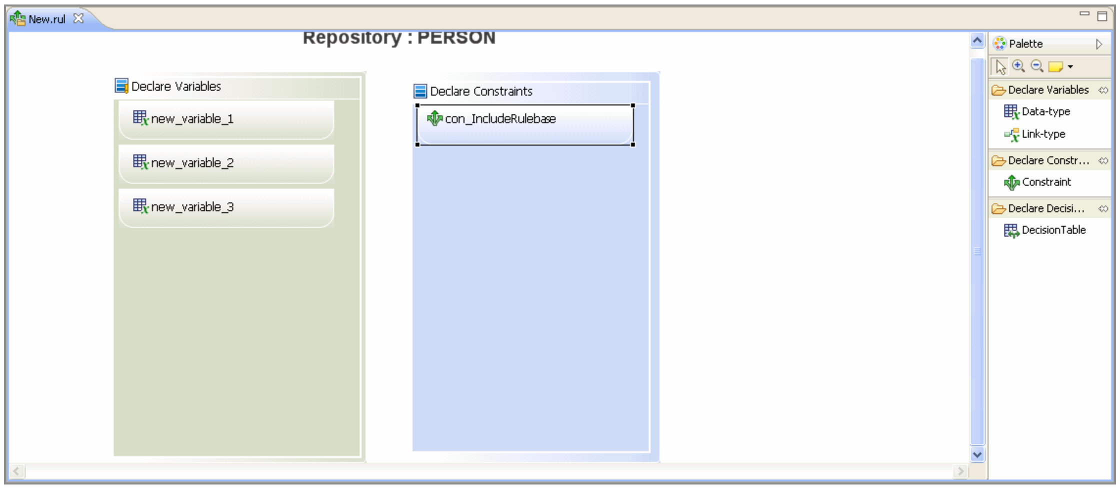
To list the variables defined in the included rulebase file in the variable listing of current rulebase file in the rulebase data view. In addition it is also available in the content assist of the expression editor.

Including a Rulebase in a Current Rulebase File

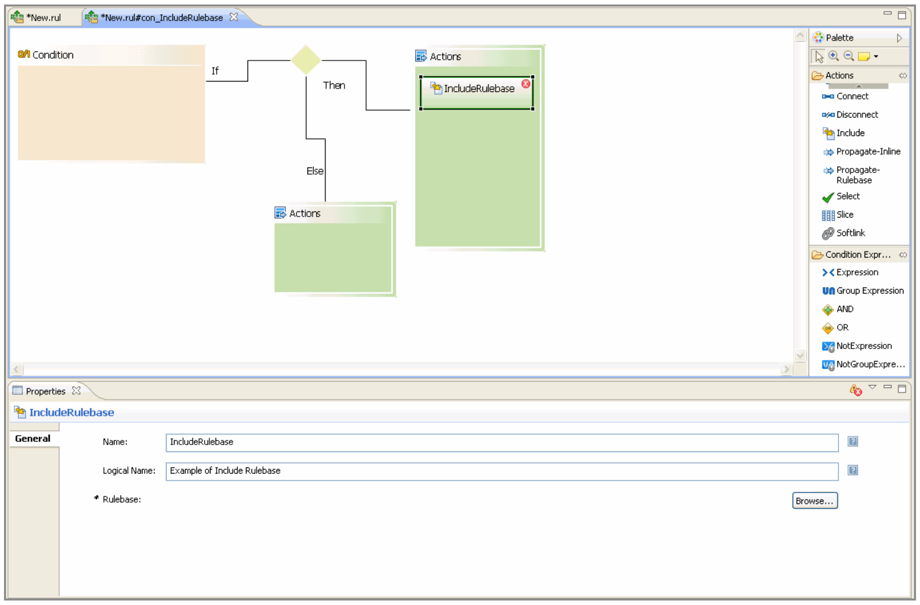
To include a rulebase in a current rulebase file perform the following steps:

Procedure

1. Create a rulebase file and declare the variables and constraints.



2. Add Include Action to the constraint.



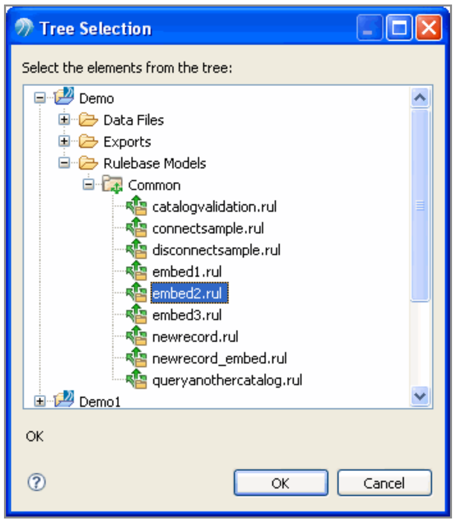
IncludeRulebase

Name: IncludeRulebase

Logical Name: Example of Include Rulebase

* Rulebase:

3. Click **Browse** in Include action properties section. The pop up window

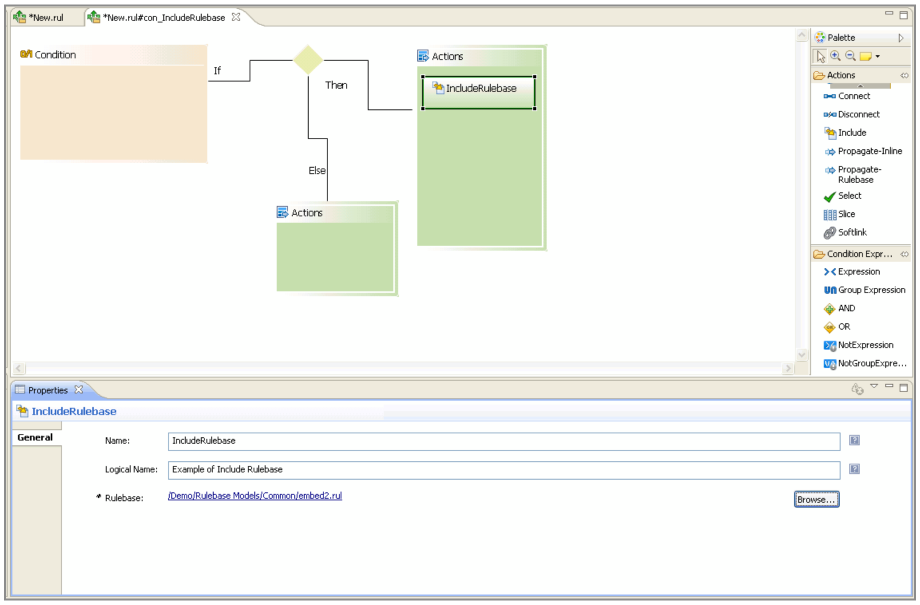


Select the elements from the tree:

```
Demo
  Data Files
  Exports
  Rulebase Models
    Common
      catalogvalidation.rul
      connectsample.rul
      disconnectsample.rul
      embed1.rul
      embed2.rul
      embed3.rul
      newrecord.rul
      newrecord_embed.rul
      queryanothercatalog.rul
Demo1
```

OK

4. The pop up tree selection window is displayed. Navigate to the common folder and select the rulebase which you want to include and click **OK**.

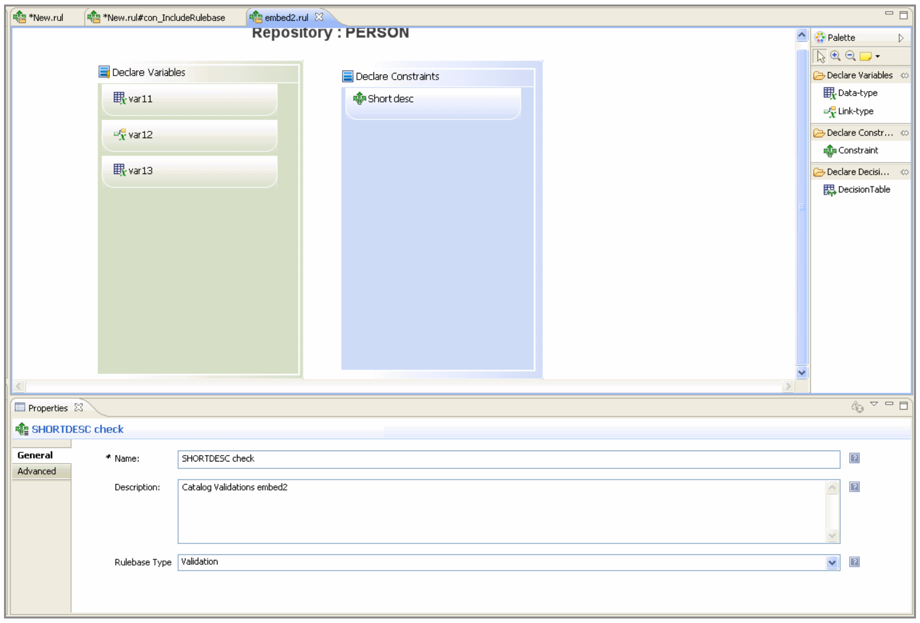


Name: IncludeRulebase

Logical Name: Example of Include Rulebase

* Rulebase: /Demo/Rulebase Models/Common/embed2.rul

5. Click the newly added rulebase link. The included rulebase will open in the rulebase editor.

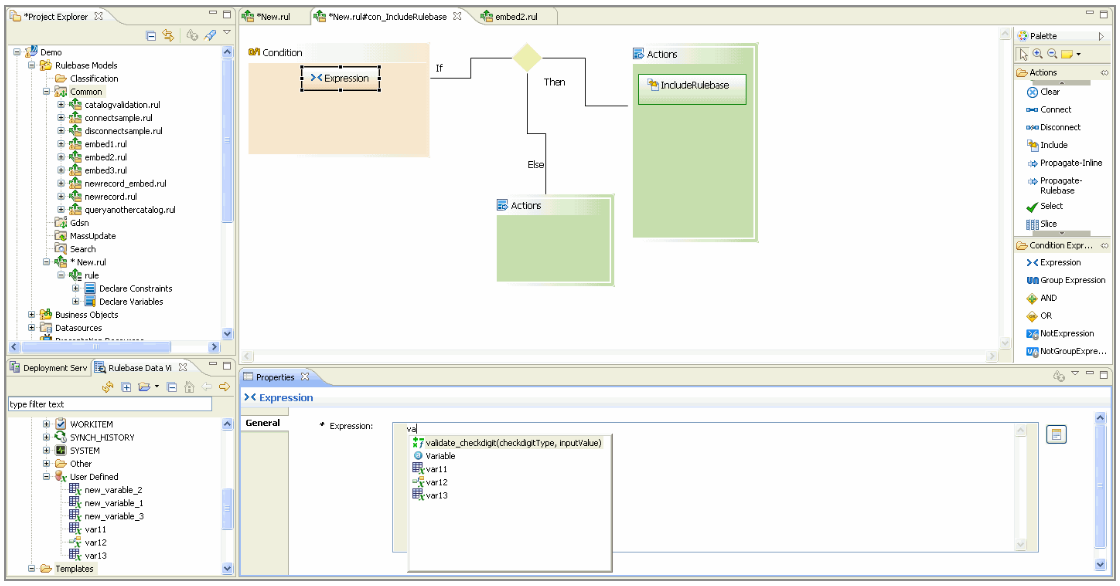


* Name: SHORTDESC check

Description: Catalog Validations embed2

Rulebase Type: Validation

6. The new variables defined in the included rulebase appears in the variable listing of current rulebases in the rulebase data view and also in the content assist of expression editor.



Propagate Inline Action

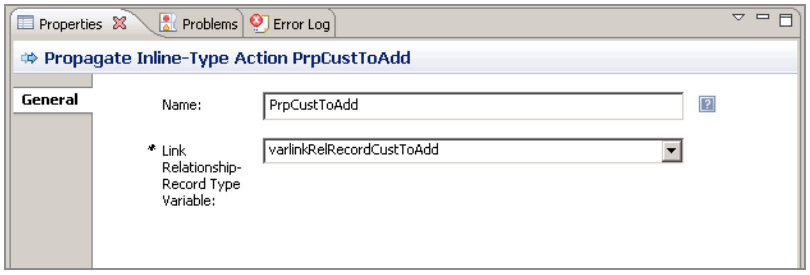
The **Propagate-Inline** action propagates values from parent records to child records. It uses the Assign Action to assign the attribute of the related record.

To define a **Propagate-Inline** action, select the Propagate-Inline **Propagate-Inline** action icon from the palette and drop it in the appropriate (Else or Then) Action container.

Propagate-Inline Action Properties

The following properties can be provided for the **Propagate-Inline** Action in the **Properties** window, **General** tab:

- **Name**: Any logical name for the Propagate action.
- **Link Relationship-Record Type Variable**: Select the variable to propagate a value for. This drop-down lists all Declared link variables with Link type as Relationship-Record.


Properties | Problems | Error Log

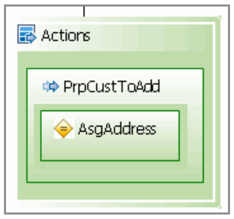
Propagate Inline-Type Action PrpCustToAdd

General

Name: PrpCustToAdd

* Link Relationship-Record Type Variable: varlinkRelRecordCustToAdd

Click the **Assign** action within the **Propagate-Inline** action box in the Actions compartment to view the Properties for the contained Assign action.



In the **Assign** action Properties, select the attribute of the related record from the drop-down and enter the expression to access the parent record values. For example: select <AttributeName> in the **Assign** drop-down and provide link.<parentrecord> in the Expression Editor.

Propagate Rulebase

The **Propagate-Rulebase** action propagates values from parent records to child records.

Unlike the Propagate Inline Action, it does not use an inline Assign Action, but requires you to provide the rulebase file in the **Properties**.

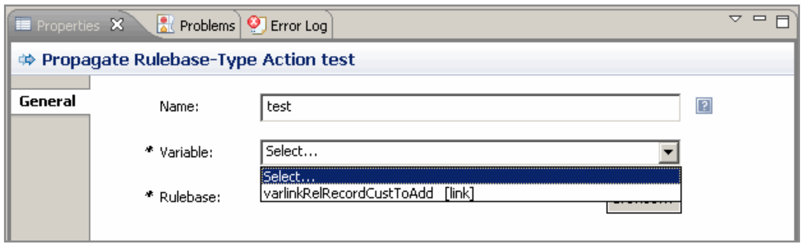
To define a **Propagate-Rulebase** action, select the Propagate-Rulebase **Propagate-Rulebase** action icon from the palette and drop it in the appropriate (Else or Then) Action container.

Propagate-Rulebase Action Properties

The following properties can be provided for the **Propagate-Rulebase** action in the **Properties** window, **General** tab:

- **Name**: Any logical name for the Propagate Rulebase action.

- **Variable**: Select the variable to propagate a value for. This drop-down lists all Declared link variables with Link type as Relationship-Record.
- **Rulebase**: Browse to select the Rulebase.



Propagate Rulebase-Type Action test

Name: test

* Variable: Select...

Select...
varlinkRelRecordCustToAdd [link]

* Rulebase:

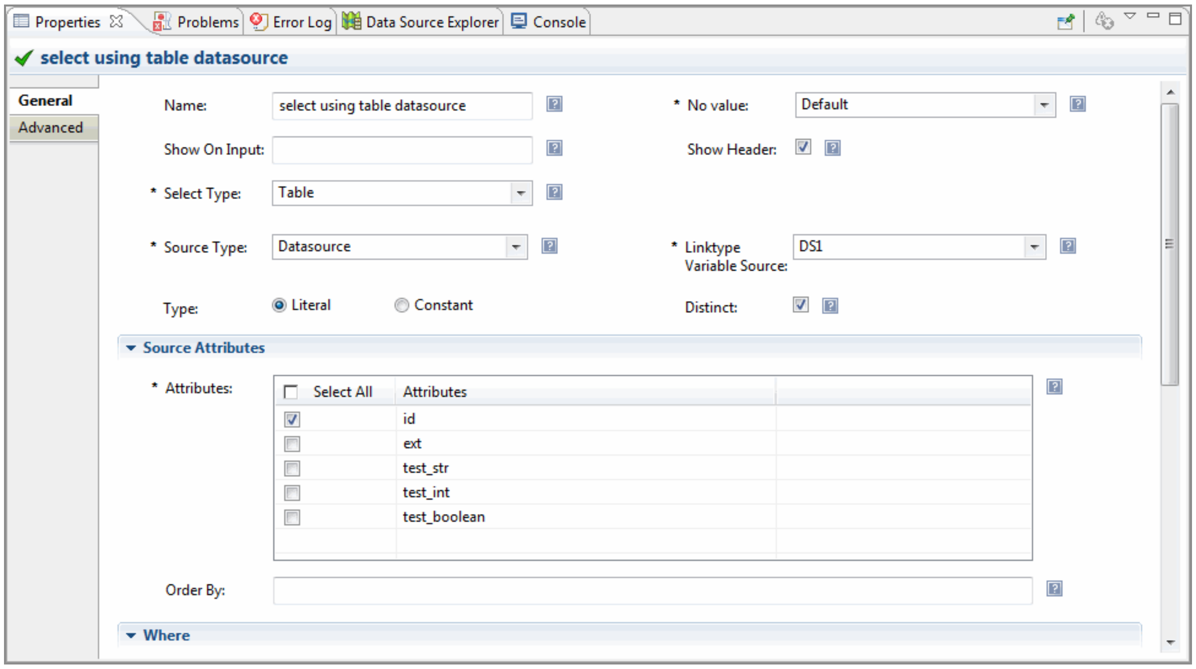
Propagate-Rulebase Validation

Values for the Variable and Rulebase are mandatory; an error marker will be displayed on the figure if these values are not specified.

Select Action

The **Select** action allows you to create a drop down with a list of values (LOV)for any attribute in the TIBCO MDM record UI.

The attribute can be specified using Use for Variables (see Constraints). To do this, select the Constraint in the main **Declare Constraints** compartment and in the **Properties**, select the variable in the **Use for variables** list.



select using table datasource

Name:	select using table datasource	* No value:	Default
Show On Input:		Show Header:	✓
* Select Type:	Table		
* Source Type:	Datasource	* Linktype Variable Source:	DS1
Type:	● Literal ○ Constant	Distinct:	✓

* Attributes:

Select All	Attributes
✓	id
	ext
	test_str
	test_int
	test_boolean

Order By:

The list of values can be populated from static constants and dynamic source. Static constant values can be specified using **Select Type** as Enum and dynamic values using **Select Type** as Table.

To define a **Select** action, select the ✓ Select **Select** action icon from the palette and drop it in the appropriate (Else or Then) Action container.

Select Action Properties

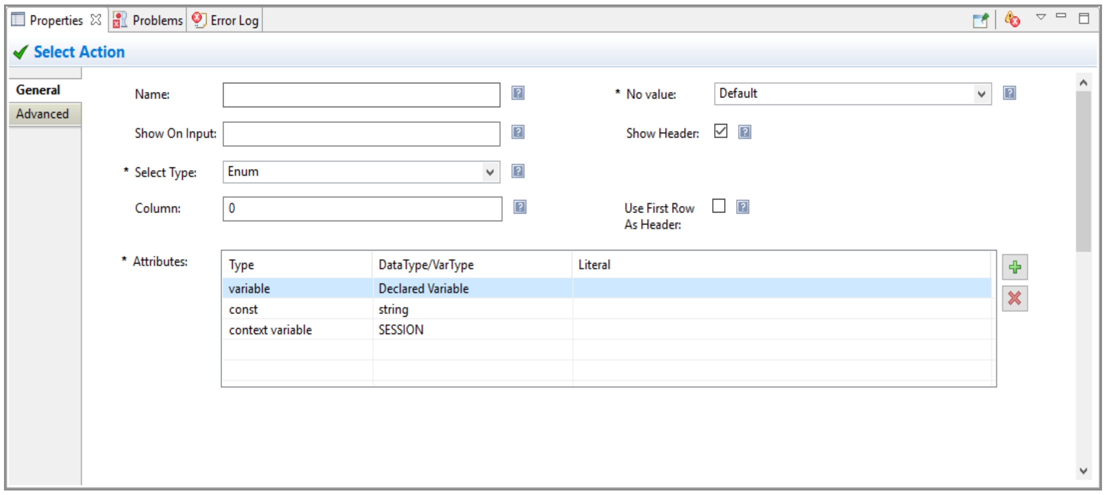
The following properties can be provided for the **Select** action in the **Properties** window, **General** tab:

- **Name**: Any logical name for the Select action.
- **No Value**: This specifies where an empty value appears (in the list). The following options are available:
  - **Default**: The empty value appears as the first value in the list.
  - **Option**: The empty value appears at the bottom of the list.
  - **No**: The empty value does not appear in the list.
- **Showoninput**: Specify the columns to display in a multi column drop down list. Enter single or comma separated values. For example, if shownoninput=1,3 then columns 1

and 3 will be displayed in the drop down in the record UI.

- **Show Header**: By default, there is no header in the data (header="no"). Select this checkbox to display as the column header.
- **Select Type**: Type can be Enum or Table. Each type has a different set of properties displayed in the Property window.

  Enum Type: The following properties are displayed for the Enum Type:



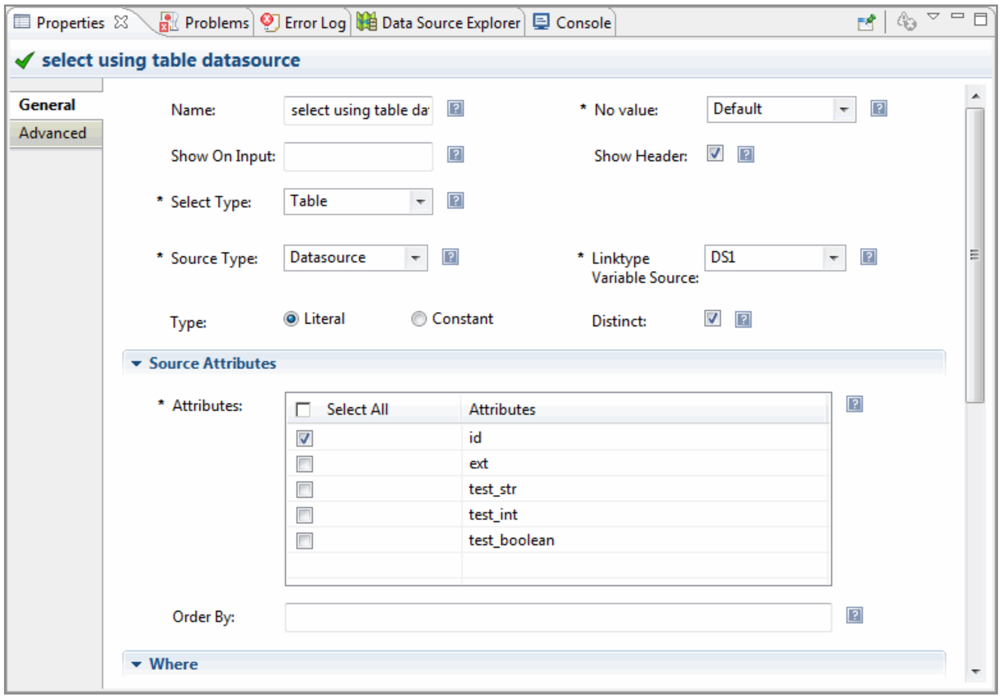
Name:		* No value:	Default
Show On Input:		Show Header:	☑
* Select Type:	Enum		
Column:	0	Use First Row As Header:	☐

* Attributes:

Type	DataType/VarType	Literal
variable	Declared Variable	
const	string	
context variable	SESSION	

- **Column**: The number of columns is solely determined by the col attribute.
- **Use First Row As Header**: If you have select Show Header checkbox, then select this check-box if you want the first row of data to be displayed as the column header.
- **Attributes**: Add constant values or declared variables to the table using the Add button on the right of the table. Select **const** or **variable** or **context variable** from the **Type** drop-down list and enter the value corresponding to the type in the **Literal** field. Use the delete button to delete values from the table.

  Table Type: The following properties are displayed for the Table Type:



select using table datasource

General
Advanced

Name:	select using table da	* No value:	Default
Show On Input:		Show Header:	✓
* Select Type:	Table		
* Source Type:	Datasource	* Linktype Variable Source:	DS1
Type:	● Literal ○ Constant	Distinct:	✓

Source Attributes

* Attributes:

Select All	Attributes
✓	id
	ext
	test_str
	test_int
	test_boolean

Order By:

Where

- **Source Type**: Defines the source to retrieve data from. Can be either Datasource or SQL.
- **Linktype Variable Source**: Lists repositories and datasources based on the selected source type.
- **Type**: Type can be either Literal or Constant.
- **Distinct**: Select this checkbox to filter duplicates and display only distinct or unique values from retrieved values. For example, if a column for "city" has a city repeated twice, the result set will display the city only once.
- **Attributes**: Displays attributes associated with the defined source type variable.
- **Order By**: Allows sorting the result set based on columns specified. Multiple columns can be specified (comma separated).

  The value for order must be specified in the following format. order=" [-]<columnpostion1>[,[-]<columnposition2>,..]"

  where: - indicates descending order <columnpositon> specifies column position based on selected columns.
- **Where**: Type in an SQL where clause syntax.

Slice Action

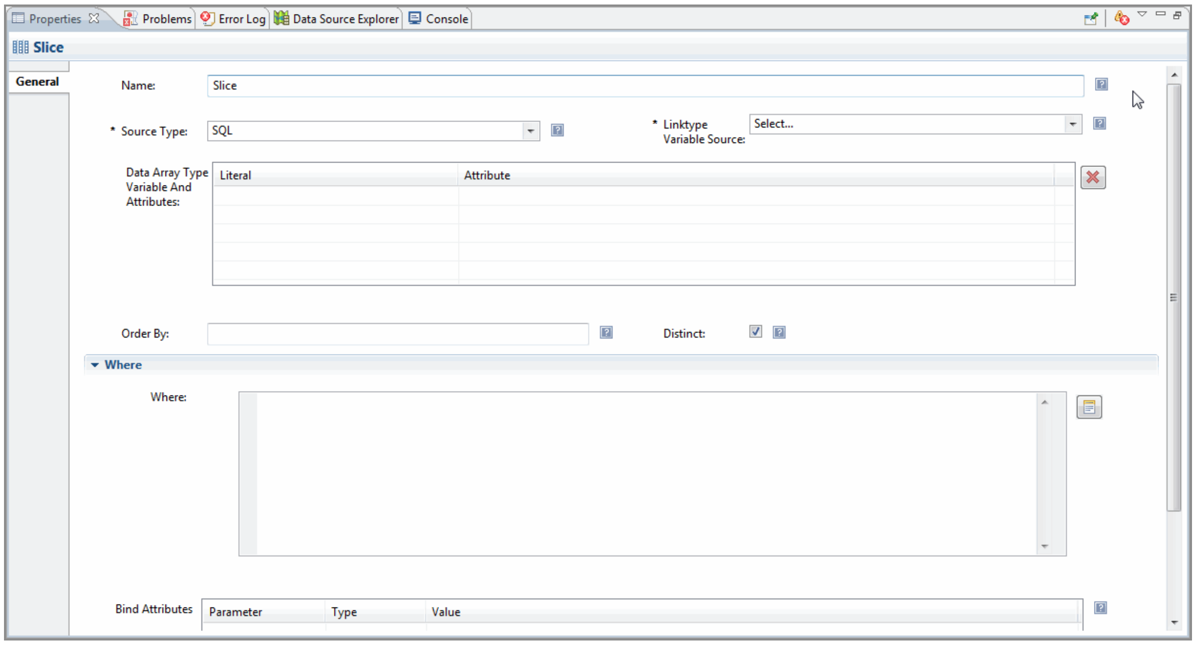
The **Slice** action enables slicing a table into columns. Each column is assigned to a different variable.

These variables are usually passed into the rulebase for further processing.

To define a **Slice** action, select the Slice **Slice** action icon from the palette and drop it in the appropriate (Else or Then) Action container.

Slice Action Properties

The following properties can be provided for the **Slice** Action in the **Properties** window, **General** tab:



Name: Slice

* Source Type: SQL

* Linktype Variable Source: Select...

Data Array Type Variable And Attributes:

Literal	Attribute

Order By:

Distinct: ✓

Where:

Bind Attributes

Parameter	Type	Value

- **Name**: Any logical name for the Slice action.
- Source Type: Defines the source to retrieve data from. Can be either Datasource or SQL.
- **Linktype Variable Source** : Lists repositories and datasources based on the selected source type.

- **Data Array Type Variable and Attributes**: You can select one of these variables to hold the result of the Slice action. If the rulebase has array type output or local variables then the rows of the table are enabled. Only the output or local variables are used.
    - Click on any row in the **Literal** column. The drop-down list displays all the available array type variables. Select the appropriate variable.
    - Click on the row corresponding to the selected literal in the **Attribute** column. The drop-down list displays all the attributes of the selected Linktype variable source. Select the attribute which you want to assign to selected variable.
    - You can select the variable only once. The selected variable is not displayed in drop-down list of other literals.
    - To delete a row, double click on that particular row and click ✖.
- **Order By**: Allows sorting the result set based on columns specified.
- **Distinct**: Filter and display only distinct retrieved values.
- **Where**: Enter SQL where clause syntax.
- **Bind Attributes**: Enter the value for the statement parameter.

Softlink Action

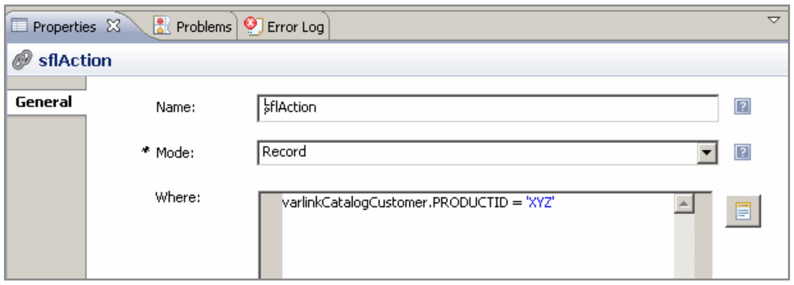
The **Softlink** action allows establishing a link between records without permanent binding.

This is needed when records stored in different repositories need to be linked. For example, it may be required to link records for vendors stored in one repository with customers in another repository. Links will be evaluated on access, and evaluation could have different results depending on any data changes.

To define a **Softlink** action, select the Softlink **Softlink** action icon from the palette and drop it in the appropriate (Else or Then) Action container.

Softlink Action Properties

The following properties can be provided for the **Softlink** Action in the **Properties** window, **General** tab:



Name: sflAction

* Mode: Record

Where: varlinkCatalogCustomer.PRODUCTID = 'XYZ'

- **Name**: Any logical name for the Softlink action.
- **Mode**: The Softlink action supports two modes:
  - **View** (display mode) - Creates a hyperlink which executes a query when clicked, and returns a list of records matching the selection criteria.
  - **Record** (non-display mode) - Executes a query and returns a list of records matching the selection criteria. Additionally, populates a record link variable and the Usefor variable points to the record link variable.
- **Where**: Allows to type in SQL where clause syntax. Specify in case of Slice use the Catalog link type variable in the expression editor e.g. for Declared varCustomerRepo use varCustomerRepo.PRODUCTID='11' and varCustomerRepo.PRODUCTIDEXT='11' in the expression editor

UnCategorize Action

This action allows to uncategorize the record from all the categories or selected categories.

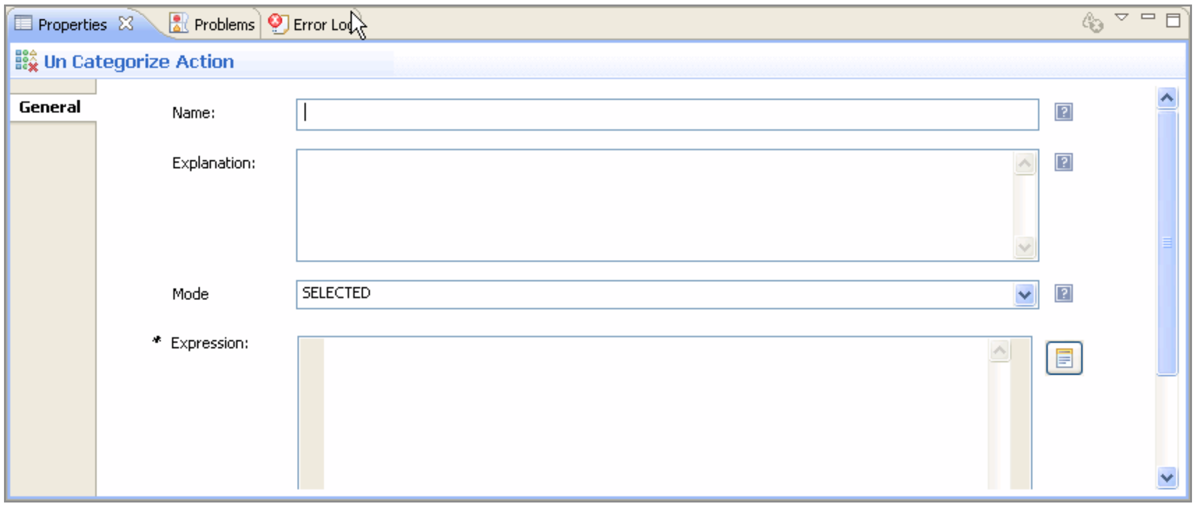
Based on the mode the record is uncategorized. If the mode is set to 'ALL' the record is uncategorized from all the categorizes. Similarly if the mode is set to "SELECTED" the record is uncategorized from the provided categories.

To define an **UnCategorize** action, select the UnCategorize **UnCategorize** action icon from the palette and drop it in the appropriate (Else or Then) Action container.

UnCategorize Action Properties

The following properties can be provided for the UnCategorize Action in the **Properties** window, **General** tab:

- **Name**: Any logical name for the UnCategorize action.
- **Explanation**: A detailed description of the UnCategorize action.
- **Mode**: If the mode is set to 'ALL' the record is uncategorized from all the categorizes. Similarly if the mode is set to "SELECTED" the record is uncategorized from the provided categories.
- **Expression**:Input parameters must be an array of link types classification code or any expression that evaluates to classification code. For example, the input parameters for categorize action: getClassificationCodeByCode(CODEVAR_SOCIAL, 'P07') - This is a classification function that returns classification code. CODEVAR_DESKTOP - This is link type classification code. CODEVAR_DB - This is link type classification code.



Mode SELECTED

UnCategorize Action Validation

By default, the Mode is **ALL**, if the mode is **SELECTED**, and **Expression** is not specified, an error marker is shown on the figure.

Severity Priority Refresh and Level

Severity

Validation can have a severity level associated with it. Severity levels can range from 1 (most critical) to 99 (least critical).

The lower the severity, the more serious the error. For example, a severity level of **2** is more serious than a severity of **4**.

Severity is specified as a **Severity** option (applicable for the **Check** and **Select** actions, displayed in the **Properties** window, **Advanced** tab). You can specify the level of severity that must be reached for an operation to fail. For example, if you specify the error severity as 4, all validations with severities less than or equal to 4 cause the operation to fail. All other errors appear as warning or information messages.

Priority

Validation can have a priority level associated with it. Priority values can be between -9 and 9. If no value is specified, the priority is 1.

The value assigned to a variable is that of the highest priority assignment. Priority is specified as a **Priority** option (applicable for the ___ and ____ actions, displayed in the **Properties** window, **Advanced** tab).

Refresh

When the user enters a new value in a drop-down list or in a text field, all attributes that use that value in their computation are refreshed.

This refresh involves a trip to the server.

Sometimes, however, the benefit of having an updated value does not compensate for the delay incurred in having to wait for that value. If that is the case, the system allows you to set the “refresh” flag to “no”. This means that the dependent attribute value is not refreshed. By default the "refresh" flag is set to "yes".

The refresh flag can be added to the following actions:

- <assign>
- <select>

- <clear>

Level

Level is used to control the display of information messages in the UI.

All assignment messages with level less than or equal to information_threshold are displayed on the UI. The default value for level is 1.

Rulebase Data View

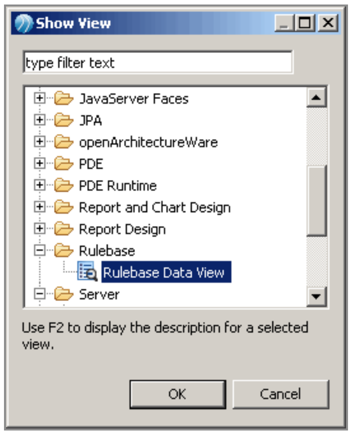
The **Rulebase Data View** contains elements that can be used when building rulebases.

It contains artifacts imported through metadata import (such as users, roles, and repositories), functions, variables and templates.

Rulebase Data View

The Rulebase Data View is usually present in the bottom left of the screen.

If not visible, you can add this view by going to **Window** > **Show View** > **Other** and then selecting **Rulebase Data View** under **Rulebase**.



Navigating through the Rulebase Data View

Icons are present at the top of the view to help navigate and filter components.

Click the **Refresh** button to refresh the view, **Expand All** to expand all components, Collapse All to minimize components, and choose the project whose details you want to display in the current view.

Components

The following are available here:

Rulebase Data View components

	Components	**Details**
Domain Objects	DataSources	For details, see Domain Objects.
	Repositories	
	Roles	
	Users	
Operators	Math Operators	For details, see Operators.
	Comparison Operators	
Functions	Comparison	For details, see Functions.
	Math	
	String	
	Other	

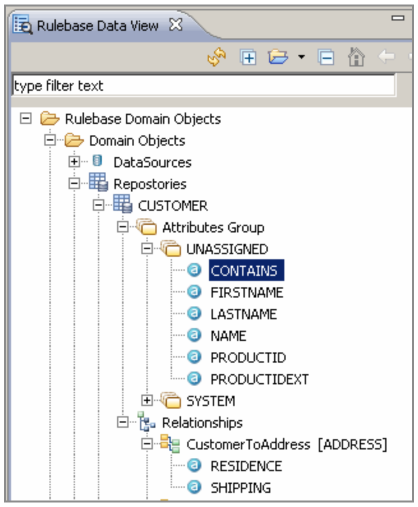
	Components	Details
Variables	Session	For details, see Implicit Context Variables.
	Workitem	
	Synch_History	
	System	
	Other	
	Workflow	
	AttributeHistory	
	AttributeQuality	
	PrecedenceResult	
	User Defined	
Templates		For details, see Templates.

Domain Objects

This section contains Datasources, Repositories, Roles, and Users imported into TIBCO MDM Studio.

See Importing Users and Roles, and Defining or Importing Repository Data.

Any of these components can be expanded; for example expand Repositories to see individual repositories, attribute groups, attributes, and relationships.



Attributes and relationships can be directly dragged from the Rulebase Data View and dropped into expressions for ease of use.

Note: Data Source in the same project are automatically updated into the Rulebase Data View.

Operators

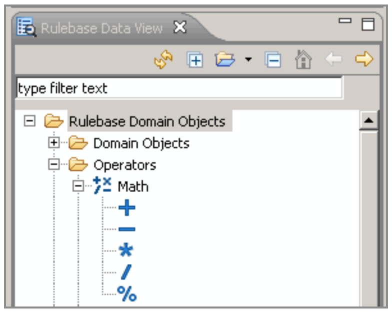
This section contains Math and Relational Operators that can be used in expressions.

To add a math or relational operator, expand **Operators**, select the required icon under **Math** or **Relational** and drag it into an expression.

Math Operators

- ⊟
- Minus
- ⁄
- Div

- *
- Mult
- %
- Percent
- +
- Plus



Minus

Description	Operands	Returns	Example
Calculates subtraction of values.	1...n numeric expressions.	Calculated numerical.	Arg1 - Arg2 - Arg3 - ... Argn

Div

Description	Operands	Returns	Example
Calculates	1...n numeric	Calculated	Arg1 / Arg2 /

Description	Operands	Returns	Example
division of values.	expressions.	numerical. Division by 0 returns null. By default, results are rounded to 8 decimal places.	Arg3 / ... Argn

Mult

Description	Operands	Returns	Example
Multiplies values.	1...n numeric expressions.	Calculated numerical	Arg1 * Arg2 * Arg3 * ... Argn

Percent

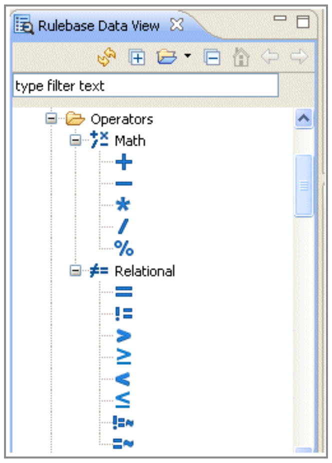
Description	Operands	Returns	Example
Computes percentage.	1...n numeric expressions.	Calculated numerical.	{(Arg2 - Arg1) / Arg1}* 100

Plus

Description	Operands	Returns	Example
Calculates addition of values	1...n numeric expressions.	Calculated numerical.	Arg1 + Arg2 + Arg3 +... Argn

Relational Operators

- !=
- Not Equal to
- !=~
- Scalar Matching - Not equal to
- <
- Less Than
- ≤
- Less than equal to
- =
- Equal to
- =~
- Scalar Matching - Equal to
- >
- Greater Than
- ≥
- Greater Than Equal To



Not Equal to

Description	Operands	Returns	Example
Checks if first operand is not equal to the second.	2 expressions.	**true** — if first operand is not equal to second operand. **false** — if first operand is equal to second operand.	Arg1 != Arg2

Scalar Matching - Not equal to

Description	Operands	Returns	Example
To be used in conjunction with NEQ (See Not Equal to) for scalar matching=false.	2 expressions.	**true** — if expression Arg1 does not match the Regular Expression Arg2. **false** — if expression	Arg1 !=~ Arg2

Description	Operands	Returns	Example
		Arg1 matches the Regular Expression Arg2..	

Less Than

Description	Operands	Returns	Example
Checks if first operand is lesser than the second.	2 expressions.	**true** — if first operand is < second operand. **false** — if first operand is > second operand.	Arg1 < Arg2

Less than equal to

Description	Operands	Returns	Example
Checks if first operand is less than or equal to the second.	2 expressions.	**true** — if first operand is <= second operand. **false** — if first operand is > second operand.	Arg1 <= Arg2

Equal to

Description	Operands	Returns	Example
Checks if first operand is equal to the second.	2 expressions.	true — if first operand is equal to second operand. **false** — if first operand is not equal to second operand	Arg1 = Arg2

Scalar Matching - Equal to

Description	Operands	Returns	Example
To be used in conjunction with EQ (See Equal to) for scalar matching=false.	2 expressions.	**true** — if expression Arg1 matches the given regular expression Arg2. **false** — if expression Arg1 does not match the given regular expression Arg2.	Arg1 !=~ Arg2

Greater Than

Description	Operands	Returns	Example
Checks if first operand is greater than the second.	2 expressions.	**true** — if first operand is > second operand. **false** — if first operand is < second operand.	Arg1 > Arg2

Greater Than Equal To

Description	Operands	Returns	Example
Checks if first operand is greater than or equal to the second.	2 expressions.	**true** — if first operand is >= second operand. **false** — if first operand is < second operand.	Arg1 >= Arg2

Functions

This section contains Comparison, Math, String, and Other functions that can be directly dragged and dropped while building expressions.

- Comparison Functions
- Math Functions
- String Functions
- Other Functions
- Classification Function

Note: Custom functions can also be created and saved as templates. For more information on creating custom functions, refer Custom Functions.

Functions

Comparison Function	Math Functions	String Functions	Other Functions	
changed	round	concat	checkdigit	validate_checkdigit
defined		length	count	toMultivalue
match		lpad	distinct	nvl
undefined		rpad	duplicate	sequence
		substring	enum	strip
		trim	filter	synchstatus
		uppercase	invokeJavaAPI	syncOperationAttribute
			max	tableDatasource
			min	tableSql

Comparison Functions

changed

Description	Syntax	Parameters	Returns	Example
If Variable is specified then attribute compare will be done and if the Variable is not specified then record compare will be done.	changed (variable, keyword)	variable: Variable (optional)keyword (optional): PREVIOUS_VERSION (default), PREVIOUS_ CONFIRMED_ VERSION) (Any one from the parameter list is mandatory.)	**true** — if attribute value has changed from its previous version. **false** — if record value has changed from its previous version.	`changed (SHORTDESC, 'PREVIOUS_ VERSION')`

defined

Description	Syntax	Parameters	Returns	Example

Determines if value has been defined.	defined (variable)	variable: Variable(1 to n)	**true** — if <var> has a non-null, non-empty value.
			
			false — if otherwise
				defined(SIZE_ UOM)
				
				`defined(SIZE_ UOM, PRICE)`

in

Description	Syntax	Parameters	Returns	Example
Checks if operand is contained in a list.	in(variable, array)	variable: Variable array: List of values.	**true** — if arg1 in (arg2, arg3, ...,argn). **false** — otherwise.	`in(PRODUCTID, {'XYZ%','CCCCC'})`

like

Description	Syntax	Parameters	Returns	Example
Used to search for a specified pattern in a variable.	boolean like (variable, constantString)	variable: Variable for which pattern has to be searched. constantString: Pattern string.	The boolean value of the pattern.	`like (PRODUCTID, 'XYZ%')`

Note: Note: The Like operator does not support attributes. It can only be used in sql expression.

match

Description	Syntax	Parameters	Returns	Example
Regular expression match.	match(variable, regexStr)	variable: Variable to matchregexStr: Regular	**true** — if first operand matches	`match (GTIN,'/^\\d {14}$/’)`

Description	Syntax	Parameters	Returns	Example
		Expression	second operand. **false** — if first operand does not match second operand.	

undefined

Description	Syntax	Parameters	Returns	Example
Determines if a value is undefined.	Undefined (variable)	variable: Variable(1 to n)	**true** — if the variable is null or empty. **false** — otherwise	`undefined (FIRST_NAME)`

Math Functions

round

Description	Syntax	Parameters	Returns	Example

Rounds to a defined set of decimal places.	round(num, decimalPlaces, roundingMethod)	num: Number or array of numbers.
		
		decimalPlaces: Number of decimal places to round to (Optional). Default to 0.
		
		roundingMethod: Rounding method (Optional)
		
		HALF_UP - default, round up towards "nearest neighbor".
		
		CEILING - round towards positive infinity.
		
		DOWN - truncate and round down towards zero.
		
		FLOOR - round towards negative infinity.
		
		HALF_DOWN - round down towards "nearest neighbor"
		
		HALF_EVEN - round towards even "nearest neighbor"
		
		UNNECESSARY - assert an exact
			The array of number values.	`round((11 / 9))`
				
				`equivalent to round((11 / 9), 0, 'HALF_UP')`
				
				`round((11 / 9), 2)`
				
				`equivalent to round((11 / 9), 2, 'HALF_UP')`
				
				`round((11 / 9), 2, 'CEILING')`
				
				`round({(11 / 9), 10.234, 99.455}, 2, 'CEILING')`

Description	Syntax	Parameters	Returns	Example
		result. UP - increment and round up to zero.		

String Functions

concat

Description	Syntax	Parameters	Returns	Example

Concatenates string values.	concat(strExpr)	strExpr: string expressions(1 to n)	String, which is a concatenation of all input strings.	`concat('PREFIX-','STRING','-SUFFIX')`
				`concat('PREFIX-',rpad(PRODUCT_NUMBER, 9, 'A'),'-SUFFIX')`

length

Description	Syntax	Parameters	Returns	Example
Computes string length.	length(strExpr)	strExpr: string expression	Length of the specified string.	`length ('PREFIX-STRING-SUFFIX')`

lpad

Description	Syntax	Parameters	Returns	Example

Pads a string on the left.	lpad(strToPad, paddedStrLen, charToPad)	strToPad: string or array of strings.
		
		paddedStrLen: final length of padded string.
		
		charToPad: character to pad with (Optional, Default is SPACE).
			String.	`lpad('ABC’, 4)`
				
				`lpad({'ABC', 'DEF'}, 4)`

rpad

Description	Syntax	Parameters	Returns	Example

Pads a string on the right.	`rpad(strToPad, paddedStrLen, charToPad)`	strToPad: string or array of strings.
		
		paddedStrLen: final length of padded string.
		
		charToPad: character to pad with (Optional, Default is SPACE).
			String.	`rpad('ABC’, 4)`
				
				`rpad({'ABC', 'DEF'}, 4)`

substring

Description	Syntax	Parameters	Returns	Example
Substring of current string.	`substring (strTosubset, start, numChars)`	strTosubset: string. start: start position. numChars: number of characters (Optional, default is remaining length).	Returns string as the result of substring operation.	`substring ('ABC', 1)`

trim

Description	Syntax	Parameters	Returns	Example
Trims leading and trailing spaces.	`trim(strExpr)`	strExpr: string expression.	String with leading and trailing spaces removed.	`trim(CUSTOMER_NAME)`

uppercase

Description	Syntax	Parameters	Returns	Example
Uppercase of input string.	`uppercase (strExpr)`	strExpr: string expression.	String with all characters converted to uppercase.	`uppercase ('aBcDeF')`

Other Functions

checkdigit

Description	Syntax	Parameters	Returns	Example
Checks the number of digits in an attribute.	`checkdigit (checkdigitType, inputValue)`	checkdigitType: Number of digits to be checked in the input value. inputValue: string input value.	String.	`checkdigit ('14', GTIN)`

count

Description	Syntax	Parameters	Returns	Example

Counts number of (non-null, non-false) entries in array.	`count(val)`	val: Value or array of values.	Number.	`count (CHILDCITIES)`
				
				`count ({'PREFIX', NULL, 'ABCD’})`

distinct

Description	Syntax	Parameters	Returns	Example

Returns distinct values.	`distinct (values)`	values: Array of values	Distinct values.	`distinct (CHILDCITIES)`
				
				`count(distinct (SIBLING_ REL.UOM))`

duplicate

Description	Syntax	Parameters	Returns	Example
Checks for records with duplicate values	`duplicate (variable, isCaseSensitive)`	variable: value or array of values. isCaseSensitive: whether to compare attribute value irrespective of case of the string or not. Default is true. (Optional)	**true** : if duplicate **false**: otherwise.	`duplicate (variable, false)`

enum

Description	Syntax	Parameters	Returns	Example

Defines a list of values.	`enum (values, col, header)`	values: array of strings (1 to n).
		
		col: number of columns (optional).
		
		header: specifies if first row of data is description of columns (optional)
		
		HEADER_YES, HEADER_NO.
			Array of values.	`enum ({'FA<sep/>Farenheit', 'CE<sep/>Celcius', 'KA<sep/>Kelvin'}, 2)`
				
				`enum({’TEMP_UNIT’, 'FA<sep/>Farenheit', 'CE<sep/>Celcius', 'KA<sep/>Kelvin'}, 2, HEADER_YES)`
				
				`enum ({'FA<sep/>Farenheit', 'CE<sep/>Celcius', 'KA<sep/>Kelvin'}, 2, HEADER_NO)`

filter

Description	Syntax	Parameters	Returns	Example

Filters a list of records.	`filter(records, selectionCriteria)`	records: array of records.
		
		selectionCriteria: WHERE Clause.
			Array of records that match filter criteria.	`filter(CHILD_RECORDS, where [PERISHABLE = 'YES' and DD = 'X'])`
				
				`<op func="filter">`
				
				`<var>CHILD_RECORD_LIST</var>`
				
				`<condition> <and>`
				
				`<eq>`
				
				`<var>EFFECTIVE_DATE</var>`

Description	Syntax	Parameters	Returns	Example
				`<var>ROOT/EFFECTIVE_DATE</var>` `</eq>` `</and>` `</condition>` `</op>`
Filters relationship attributes.	`filter_relationshiprecords(relationships, selectionCriteria)`	relationship: array of relationships. selectionCriteria: WHERE Clause.	Array of relationship that match filter criteria.	`filter_relationshiprecord (ccr_relationship, where ( AddressType = 'Y' )))` `<op func="filter_relationshiprecord"> <var>ccr_relationship</var> <condition> <eq> <var>AddressType</var> <const type="string">Y</const>` `</eq> </condition> </op>`

invokeJavaAPI

Description	Syntax	Parameters	Returns	Example
Invoke in-built java methods.	invokeJavaAPI (constantString ,variable)	constantString: Java function to be called. variable:	Result of function on variable.	`invokeJavaAPI ('java.lang.String.length', Variable)`

Description	Syntax	Parameters	Returns	Example
		Parameter to Java Function.		

lookup

Description	Syntax	Parameters	Returns	Example
Lookup value in database.	`lookup( exprToLookup, tableExpr)`	exprToLookup: Expression to lookup. tableExpr: Table Expression	A string value as a result of lookup operation.	`lookup(UOM, tableDatasource ('UOMCODES', 'VALUE', 'CODE'))`

max

Description	Syntax	Parameters	Returns	Example
Maximum value.	`max(values)`	values: value or array of values(1 to n).	Maximum value encountered. Nulls are ignored.	`max (CHILDPRICES)`

min

Description	Syntax	Parameters	Returns	Example
Minimum value.	`min(values)`	values: value or array of values(1 to n).	Minimum value encountered. Nulls are ignored.	`min (CHILDPRICES)`

toMultivalue

Description	Syntax	Parameters	Returns	Example
Allows assignment for multi value variables (used only for multi value attributes).	`toMultivalue ({variable})`	variable: array of values.	Array of values.	`toMultivalue ({'1', '2'})`

nvl

Description	Syntax	Parameters	Returns	Example
Substitutes values when NULL values encountered.	`nvl(expr1, expr2)`	expr1: expression 1.expr2: expression 2.	If expression1 is null, returns expression2; otherwise returns expression1.	`nvl(expression1, expression2)` `nvl(expression1, 'expression2')`

sequence

Description	Syntax	Parameters	Returns	Example
Used to return the next value from a database sequence.	`sequence (dbSequenceName)`	dbSequenceName: name of the database sequence.	Next sequence value converted to string.	`sequence('MQ_ SEQUENCE_1')` `sequence (mysequence)` `concat('HS=' lpad(sequence ('MQ_ SEQUENCE_1'), 9, '0'), '- JAL')`

strip

Description	Syntax	Parameters	Returns	Example
Removes all null values from an array.	`strip(Values)`	Values: array of values.	Array with all null values removed.	`strip (CHILDWEIGHT)`

synchstatus

Description	Syntax	Parameters	Returns	Example
Gets record synchronizat ion status.	`Synchstatus (marketpla ce,`	marketplace: Marketplace/Data pool Organization	**true** - if record has ever been Added/Published/Linked/Co	`synchstat us ('UCCne`

Description	Syntax	Parameters	Returns	Example

	`operation)`	name.
		
		operation: Operation name (Optional, Default is ADD. PUBLISH, LINK, COMMIT are other options).
			mmitted to this Marketplace/Datapool.
			
			false - otherwise.
				`t')`
				
				`synchstatus('UCCnet', 'PUBLISH')`

syncOperationAttribute

Description	Syntax	Parameters	Returns	Example
Used to get the operation date (GDSN only).	`SyncOperationAttribute(operation, datapool, tradingPartner)`	operation: Operation name (only ADD supported currently). datapool (Optional).: marketplace or datapool name. tradingPartner (Optional): trading partner name.	Latest synchronization Date for the operation.	`SyncOperationAttribute('ADD', Datapoolname, TradingPartnerName)`

tableDatasource

Description	Syntax	Parameters	Returns	Example
Selects values from a datasource.	`tableDatasource (datasourceName, columns, whereClause, DISTINCT)`	datasourceName: name of the datasource. columns (1 to n): array of columns. whereClause: where clause (optional). DISTINCT (Optional): The valid values are DISTINCT_TRUE and DISTINCT_ FALSE.	Array of values.	`tableDatasource (COUNTRYCODES, COUNTRYCODE, COUNTRYNAME, where like (COUNTRYNAME, 'Arg%'))`

tableSql

Description	Syntax	Parameters	Returns	Example
Selects values from a repository.	`tableSql( variable.column, whereClause, DISTINCT)`	variable.column: array of columns from a repository variable. whereClause: where clause (optional). DISTINCT	Array of records.	`tableSql (var.COUNTRYNAME, where like (COUNTRYNAME,'Arg%'))` Where var is a link type variable pointing to the repository.

Description	Syntax	Parameters	Returns	Example
		(Optional): The valid values are DISTINCT_TRUE and DISTINCT_ FALSE.		

toDate

Description	Syntax	Parameters	Returns	Example
Converts string to date.	`toDate (datestr)`	datestr: Date in string format.	String converted to date.	`toDate (’12/10/2010’)`

validate_checkdigit

Description	Syntax	Parameters	Returns	Example
Validates a set of digits against a format.	`validate_ checkdigit (checkdigitType, inputValue)`	checkdigitType: Standard to verify against. inputValue: String.	**true** - string gets validated against the format. **false** - otherwise.	`validate_ checkdigit (GTIN-14, GTIN)`

Classification Function

There are two types of custom functions for classification in the rulebase.

- Record based: The record parameter is implicit and mandatory for these custom functions. The following are the record based custom functions:

  -
    ```
    isRecordCategorizedUnderScheme
    isRecordCategorizedUnderCodesPath
    isRecordCategorizedUnderCodeNamesPath
    isRecordCategorizedUnderMultipleCodePaths
    isRecordCategorizedUnderMultipleCodeNamePaths
    getClassificationCodePathsForRecord
    getClassificationCodeNamePathsForRecord
    getClassificationCodesForRecord
    getClassificationCodeNamesForRecord
    isRecordCategorized
    isRecordCategorizedUnderAll
    ```
- Metadata based: The metadata parameter is not implicit parameter. The following are the metadata based custom functions:

  -
    ```
    getClassificationScheme
    getClassificationCodeByCode
    getClassificationCodeByName
    getClassificationCodeForCodesInPath
    getClassificationCodeForCodeNamesInPath
    getClassificationCodeLevel
    isSubCategoryOfCode
    isSubCategoryOfCodeName
    stringTreepathOfCodeToClassificationCode
    stringTreepathOfCodeNamesToClassificationCode
    ```

For more information, refer getClassificationScheme.

AddressCleansing Function

The address cleansing function is introduced in rulebase for a single record cleansing on record add or update operations.

Geocode

The `geocode` function with no parameters can be used to call the GeoAnalytics service for record. The function retrieves all the required parameters from the configuration specified in configurator for the Geo Analytics properties.

For information, refer to the section, "Configuration Properties of TIBCO GeoAnalytics" in *TIBCO MDM System Administration*.

Description	Syntax	Parameters	Returns	Example
The function prepares a map, for a single geocoding request by having the attribute name-value pair in the current record. This function uses `Geo Analytics URL` property from Configurator to perform geocoding with the configured record address fields. **Note: Note:** If the `geocode` function is invoked multiple times, it involves an external Geocoding service and network communication, which both might be expensive in the terms of usage of the service and the time efficiency. To avoid this concern, use the `RECORD_ACTION` context variable to ensure that the `geocode` function is executed only once in the Record Add or Record Modify request.	NA	NA	• true: if successful service response, validation and update of record is done. • false: if the service is failed	`geocode()`

Built-in Functions

The built-in function is called with the following syntax:

```
<op func="funcname"></op>
```

Child values are considered arguments to the function. In turn, each function returns a result value which can be used by another function.

Custom Functions

In addition to the **built-in** functions, you can write your own functions.

During rulebase execution, if an unknown function is encountered, the application looks for the custom function definition from RulebaseCustomFunction.class.

It is located in the following directory:

`$MQ_COMMON_DIR/<internal_enterprise_name>/rulebase`

Creating a Custom Function

Procedure

1. Copy the sample RulebaseCustomFunction.java file from `$MQ_HOME/common/standard/rulebase`.

   This class has a predefined method, `execCustomFunction`, with the following signature.

   ```
   public HashMap execCustomFunction(HashMap args)
   ```

   This method takes one argument, which is a HashMap and expects HashMap in return. The following is a list of predefined constants that can be used:

Input HashMap

The input **HashMap** has the following entries:

Rulebase Constant	Equivalent String Constant	Description
FUNCTION_NAME	FUNC_NAME	The name of the function to execute.
FUNCTION_ARGUMENTS	FUNC_ARGUMENTS	ArrayList of input arguments.

FUNCTION_ARGUMENTS are passed in an ArrayList with the same order as that specified in the rulebase constraint. The following is a list of data types in the rulebase and corresponding Java types:

Rulebase Type	Java Type
string	String
number	Long BigDecimal
boolean	Boolean
date	java.util.Date
array	java.util.ArrayList

Output HashMap

The output **HashMap** has the following entries:

Rulebase Constant	Equivalent String Constant	Description
`FUNCTION_SUCCESS`	`FUNC_SUCCESS`	Set to Boolean. TRUE, if function found and executed successfully else Boolean. FALSE.
`FUNCTION_RETURN_VALUE`	`FUNC_RETURN_VALUE`	Output of the function.
`FUNCTION_ERROR_MESSAGE`	`FUNC_ERROR_MESSAGE`	Error message in case error occurred in function execution. This error message is logged in the `$MQ_HOME/elink.log` file.

Implement custom function as a separate method and call it from execCustomFunction depending upon the FUNCTION_NAME passed in.

For example:

```
if (funcName.equals("checkNumber")) {retValue = checkNumber(inArgs);}
```

Compile the `RulebaseCustomFunction.java`. For example:

```
javac RulebaseCustomFunction.java -classpath $MQ_HOME/lib/mq/AllECMClasses.jar:$MQ_
HOME/lib/external/log4j-1.2.17.jar
```

Note: Ensure that `$MQ_HOME/lib/mq/AllECMClasses.jar` and `$MQ_HOME/lib/external/log4j-x.x.jar` are in the classpath for compilation.

Copy the `RulebaseCustomFunction.class` file to `$MQ_COMMON_DIR/<internal_enterprise_name>/rulebase` folder and restart the application server.

Compile the `RulebaseCustomFunction.java` in the Windows system by using the following command:

```
javac RulebaseCustomFunction.java -classpath %MQ_
HOME%/lib/mq/AllECMClasses.jar;%MQ_HOME%/lib/external/log4j-1.2.17.jar
```

Custom Rulebase Class Example

For an example, refer to `$MQ_HOME/common/standard/rulebase/RulebaseCustomFunction.java`.

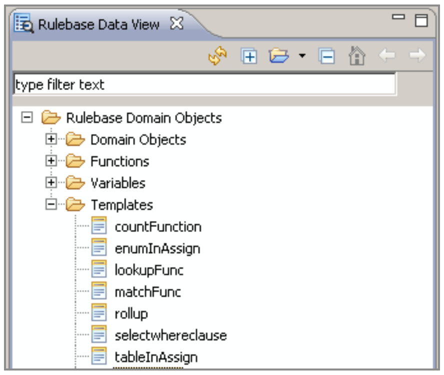
Variables

For details, see Implicit Context Variables.

Templates

The Rulebase Designer provides some basic ready to use templates.

These are present under **Templates** in the **Rulebase Data View**. Drag and drop directly into expressions.



Templates

Template	Description	Syntax
countFunction	This template uses the **count** and **filter** functions (count and filter). The result of filter is assigned to a temporary variable, i.e. PERISHABLE_LIST, that the count function operates on.	`count( PERISHABLE_LIST := filter ( CHILD_RECORDS, where PERISHABLE = 'YES' and DD = 'X' )) > 1`
lookupFuncUsingEnum	This template uses the **enum** and **lookup** functions (enum and lookup) to look for TEMPCODE in the enum list.	`lookup(TEMPCODE, enum({'FA<sep />Fahrenheit','CE<sep />Celsius'},2))`
lookupFuncUsingTableDatasource	This template uses the **lookup** and **tableDatasource** functions (See lookup and tableDatasource) to look for a specified distinct name from the datasource.	`lookup(NAME, tableDatasource ('DS1',{ PRODUCTID,PRODUCTIDEXT}, where (PRODUCTID = 'abc' and PRODUCTIDEXT = 'abc'),DISTINCT_TRUE ))`

Template	Description	Syntax
lookupFuncUsingTableSql	This template uses the **lookup** and **tableSql** functions (See lookup and tableSql) to look for a specified distinct name from a repository.	`lookup(NAME, tableSql({ var1.PRODUCTID,var1.PRODUCTIDEXT}, where (PRODUCTID = 'abc' and PRODUCTIDEXT = 'abc'),DISTINCT_TRUE ))`
matchFunc	This template matches variables defined in useforvars with the provided regular expression to find non-negative numbers.	`match(useforvars, '/^((\\d+(\\.\\d*)?)|(\\d*\\.\\d+))$/')`
rollup	This template uses the **nvl** function (See nvl) to get the value of NETWEIGHT of the child record. The result of its product (multiplied) with quantity is rolled up. CONTAINSREL and CONTAINS_RECORD are Linktype-Relationship type user defined	`+(CONTAINSREL.QUANTITY * nvl(CONTAINS_RECORD.NETWEIGHT,100))`

Template	Description	Syntax
	variables.	
selectWhereClause	This template is for use with the Select action. It shows the use of a simple expression in the where clause.	`mc1.PRODUCTID = '1111' and mc1.PRODUCTIDEXT = '33333'`
tableInAssign	This template uses the tableSql function (See tableSql) . col1 and col2 specify the columns that the table function provides values for. The second parameter provides the selection criteria for records.	`tableSql({COL1,COL2}, where COL1='ddddddd' and COL2='sssssssss')`
toDateWithIn	This template uses the toDate function (See toDate) to convert given strings to date format and then searches for DateOfBirth in them.	`in(DateOfBirth,{toDate ('10/2/86'),toDate('12/3/56')})`

Deployment

This chapter explains direct deployment of rulebases created in TIBCO MDM Studio.

Deployment Overview

MDM Studio supports direct deployment of defined rulebases to MDM.

This direct deployment of rulebases created in MDM Studio provides a quick way to deploy the graphically defined components - rather than the conventional and slower method of using the export wizard and importing/moving the exported file to the MDM server manually.

Network deployment is a web service, through which you can deploy and undeploy rulebases. Internally, MDM Studio stores rulebases in XML Metadata Interchange format (XMI). Before transporting to the MDM server, this is validated, translated into native MDM format, and then executed.

Before you attempt direct deployment, first establish a connection to a MDM Server.

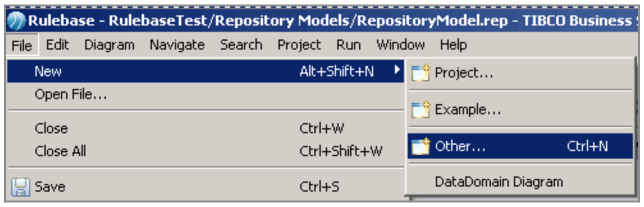
Creating an TIBCO MDM Deployment Server

Note: TIBCO MDM network deployment service requires administrative privileges.

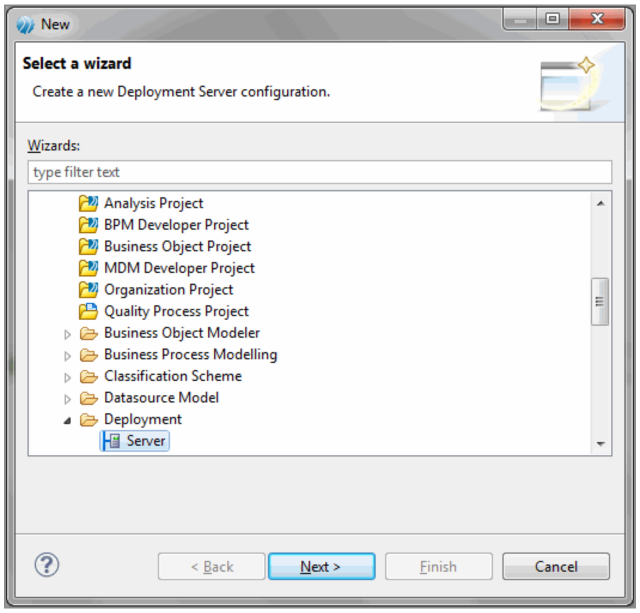
In case you do not see the Deployment Server Pane, go to **Window** > **Show view** > **Other** and select **Deployment Server** under **Studio**. Click **OK**.

Procedure

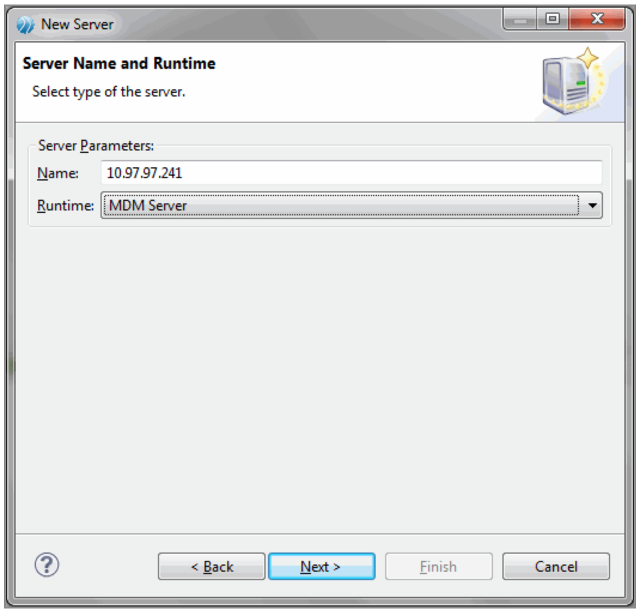
1. Select **File > New > Other.**


Rulebase - RulebaseTest/Repository Models/RepositoryModel.rep - TIBCO Business
File Edit Diagram Navigate Search Project Run Window Help
New Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Project...
Example...
Other... Ctrl+N
DataDomain Diagram

2. Select **Business Modeling > Deployment > Server**.



3. Provide the Server Name; select **MDM Server** as **Runtime**.


New Server

Server Name and Runtime

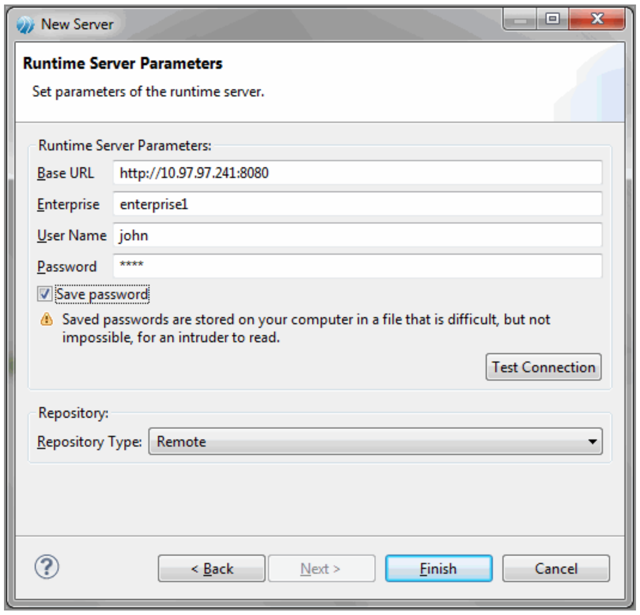
Select type of the server.

Server Parameters:

Name: 10.97.97.241

Runtime: MDM Server

4. Enter the **BaseURL, Enterprise, User Name, Password** and select **Remote** as the Repository Type. If you select the **Save password** option, you will not be prompted for a password in the following step. Click **TestConnection**. You will received successful connection message. Click **Finish**.


New Server

Runtime Server Parameters

Set parameters of the runtime server.

Runtime Server Parameters:

Base URL http://10.97.97.241:8080

Enterprise enterprise1

User Name john

Password ****

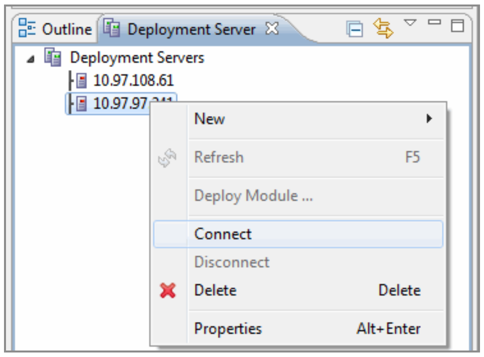
☑ Save password

Saved passwords are stored on your computer in a file that is difficult, but not impossible, for an intruder to read.

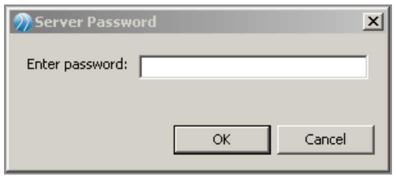
Repository:

Repository Type: Remote

5. In the Deployment Server pane, right click the created MDM Server and click **Connect**.

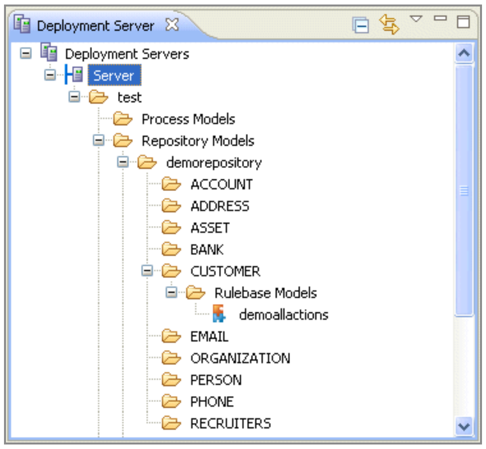


If you did not choose to save the password, you will be prompted to enter the password.



Result

This establishes a connection between the TIBCO MDM Server and TIBCO MDM Studio client and shows all deployed modules on the MDM server.



Deploying TIBCO MDM Studio where SSL is Enabled

To deploy TIBCO MDM Studio where SSL is enabled follow the steps:

Procedure

1. Copy keystore file on the same machine where TIBCO MDM Studio is installed.
2. Copy -Djavax.net.ssl.trustStore parameter in the studio.ini file. The studio.ini file is located in `$TIBCO_HOME/studio-mdm/4.1/eclipse/TIBCOBusinessStudio.ini`.
3. Specify the keystore file path in the newly added parameter. For example, -Djavax.net.ssl.trustStore=C:/app/foo.keystore

Result

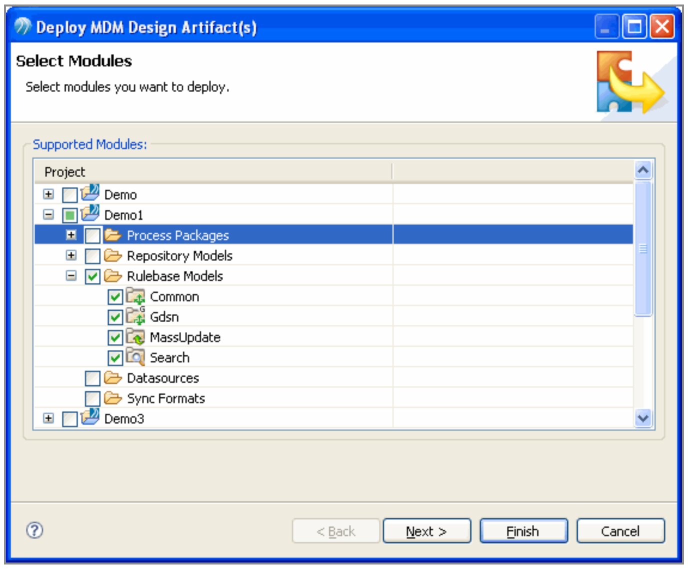
Note: Note: If you want to use SSL enabled URL, enter the URL in **Base URL** field as "https://<hostname>:<port>".

Direct Deploying of Rulebases

The following are the steps involved in directly deploying rulebases:

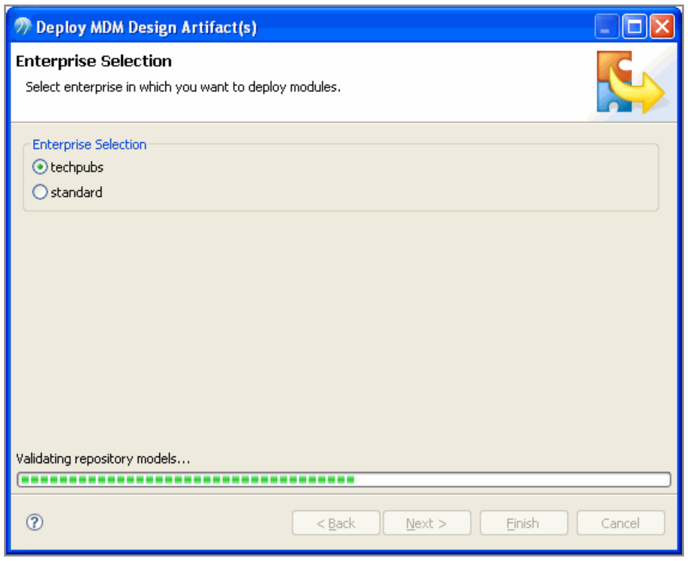
Procedure

1. In the Deployment pane, right click the <**MDM Server** and select **Deploy Module.**
2. Select the modules to deploy. Click **Next**.



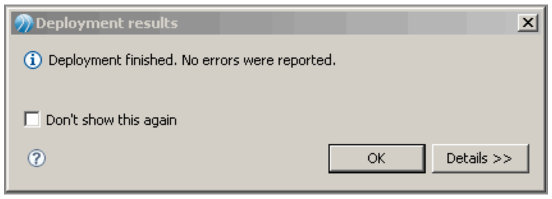
Note: Select the rulebase (.rul) file to deploy. This is listed under the relevant repository model (.rep file). Expand the .rep file and repositories under it to see associated .rul files.

3. Select the enterprise to deploy the rulebase to (either the current enterprise or standard). Click **Finish**.


Deploy MDM Design Artifact(s)
Enterprise Selection
Select enterprise in which you want to deploy modules.
Enterprise Selection
techpubs
standard
Validating repository models...
< Back
Next >
Finish
Cancel

4. If deployment is successful, you will get a message confirming it. Errors, if any, will be displayed.

Result



Once successfully deployed, you can log onto the TIBCO MDM Server and check if your rulebases and repositories have got included.

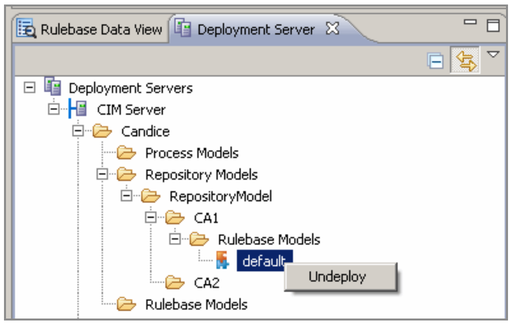
Undeploying Rulebases

To undeploy a rulebase from the server:

Procedure

1. In the Deployment Server view, expand Deployment Server
2. Then expand <MDM Server><EnterpriseName><Rulebase Models>**.rul** file.
3. Right click the deployed **.rul** file and select **Undeploy**.
4. You will get a message to confirm undeployment. Click **Yes** to undeploy.

Result



This undeploys the selected component, and a backup of it is internally renamed and stored.

Import and Export Rulebases

This chapter explains importing TIBCO MDM rulebases into TIBCO MDM Studio and exporting rulebases created in TIBCO MDM Studio.

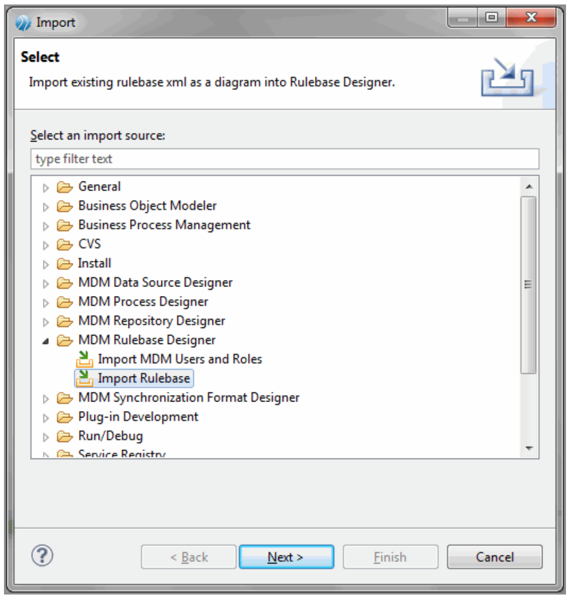
Importing Rulebases

Existing TIBCO MDM rulebases can be imported into the Rulebase Designer. Follow these steps to import rulebases:

Procedure

1. Select the project you want to import the rulebase into; right click the **Rulebase Models** folder in that project and select **Import**.

2. Select **Import Rulebase** under **MDM Rulebase Designer**. Click **Next**.

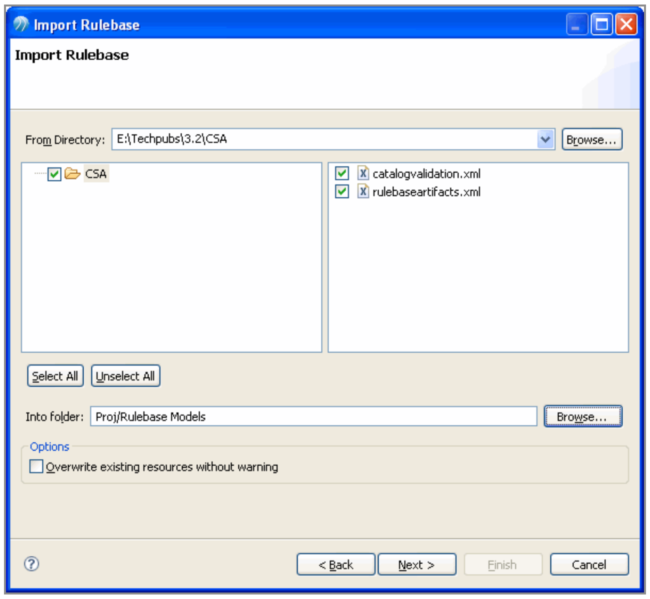


Select

Import existing rulebase xml as a diagram into Rulebase Designer.

3. Browse to the location (folder) of the Rulebase (xml) and select the checkbox against the appropriate file. You can also Browse to change the folder that the Rulebase

should be imported into. Click **Next**.



From Directory: E:\Techpubs\3.2\CSA

☑ CSA

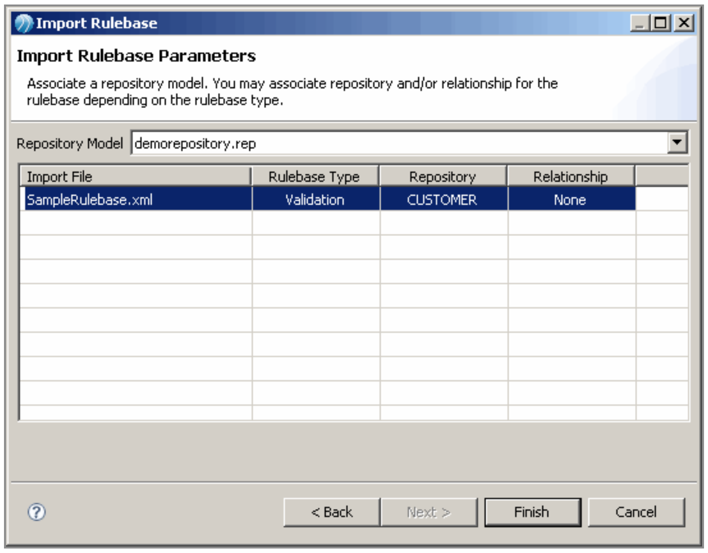
☑ catalogvalidation.xml
☑ rulebaseartifacts.xml

Into folder: Proj/Rulebase Models

☐ Overwrite existing resources without warning

4. The selected rulebase will be displayed along with its type. Select a .rep file (model) to associate (from the Repository Model dropdown) and then click in the Repository column and select the repository to associate or click in the Relationship column and select the relationship to associate. Click **Finish**.

Result



Import Rulebase Parameters

Associate a repository model. You may associate repository and/or relationship for the rulebase depending on the rulebase type.

Repository Model demorepository.rep

Import File	Rulebase Type	Repository	Relationship
SampleRulebase.xml	Validation	CUSTOMER	None

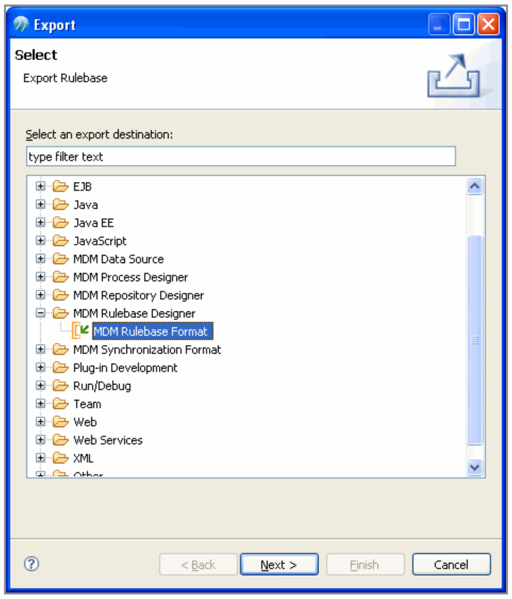
The imported rulebase will then be displayed in the appropriate folder (the one selected for import into.)

Exporting Rulebases

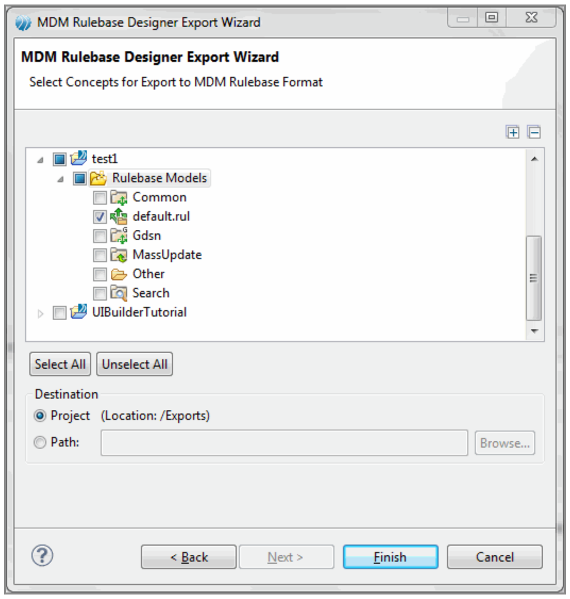
Follow this process to export rulebases designed into the Rulebase Designer into TIBCO MDM.

Procedure

1. In the Project Explorer, right click the rulebase file (.rul) to export and select **Export > Export**.
2. Select **MDM Rulebase Format** under **MDM Rulebase Designer** and click **Next**.



3. Select the rulebase file to export by selecting its checkbox. Accept the default destination (an Exports folder under the project) or provide a path.



4. Click **Finish**.

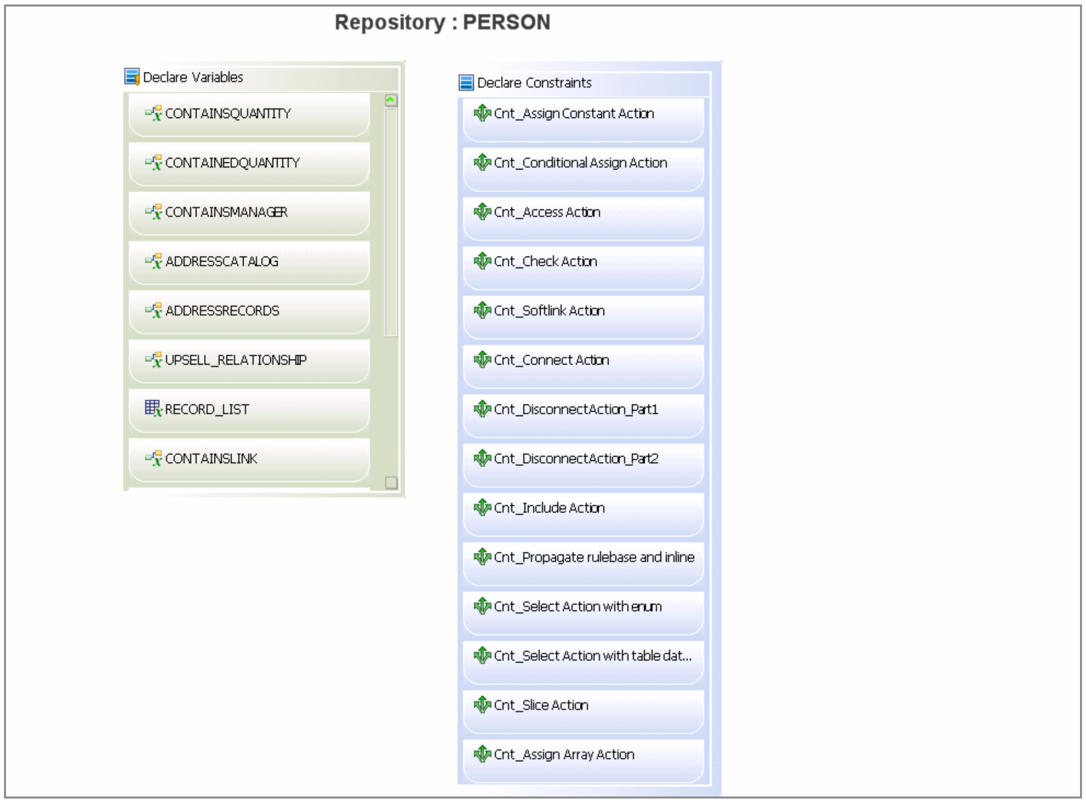
Rulebase Examples

This section has extensive rulebase code examples divided into functional segments (modules) to call attention to each of their functions.

Sample - 1

The following is a sample rulebase of catalog validations for Person repository. The Rulebase type used in this sample is of type validation.

Rulebase Sample

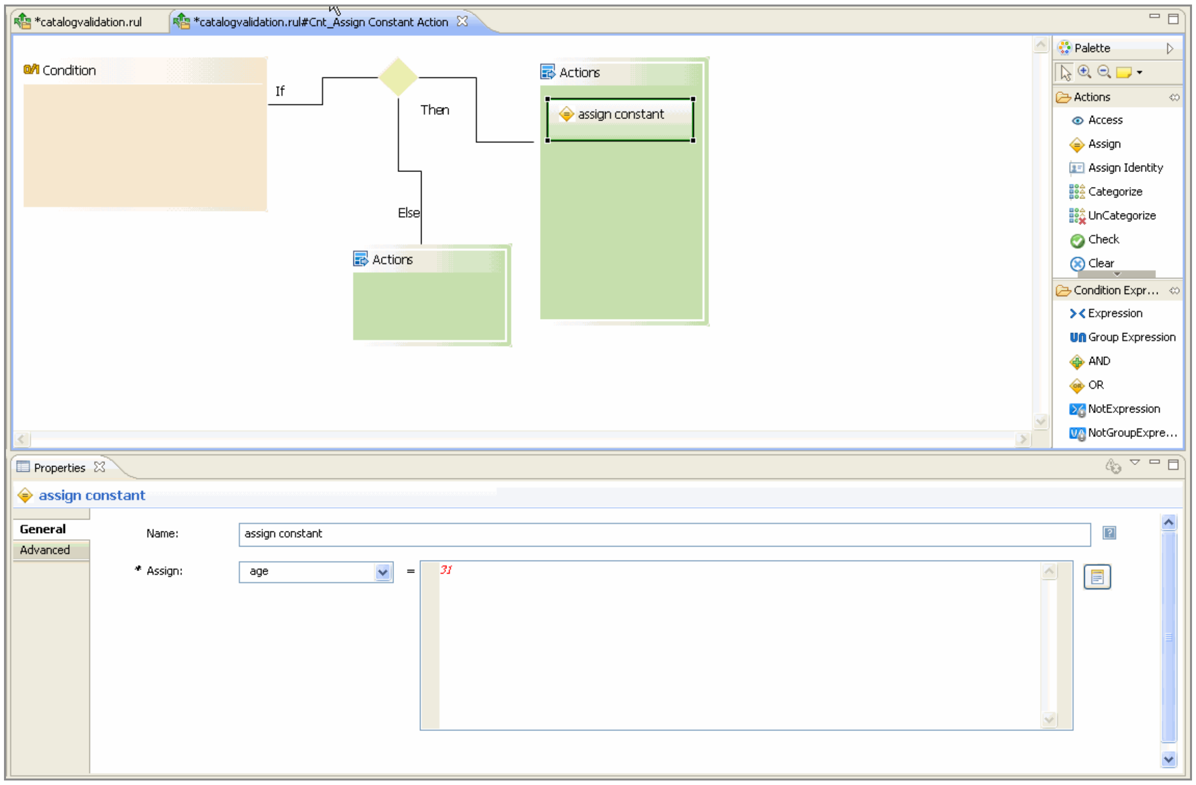


Assign Action Constant

In this example, a constant value is assigned to a variable. Double click on the Cnt_Assign Constant Action container, the rulebase opens in a new editor.

A constant value "31" is assigned to the variable "age".

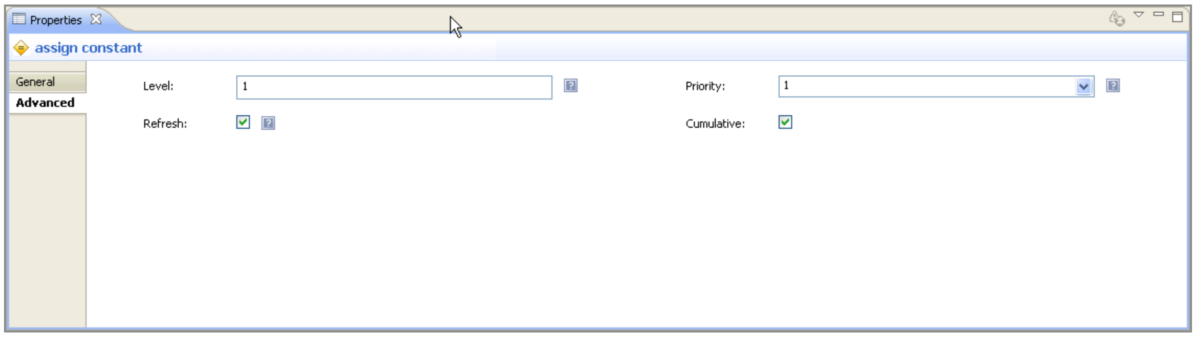
Assign Action with General Properties



Name: assign constant

* Assign: age = 31

Assign Action Rule Advanced Properties



Level: 1

Priority: 1

Refresh: ✓

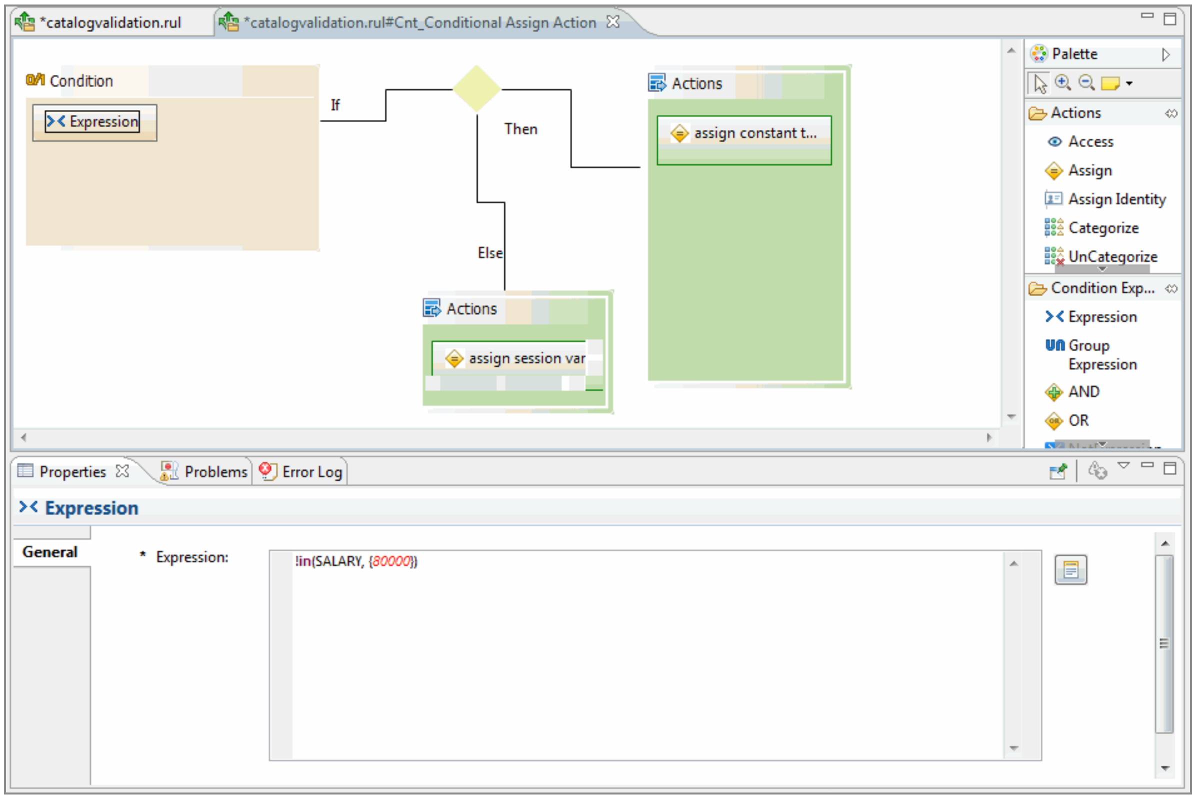
Cumulative: ✓

Assign Action Conditional

In this example a conditional value is assigned, if salary is other than 80000, then productidext is Manager else assign session variable constant.

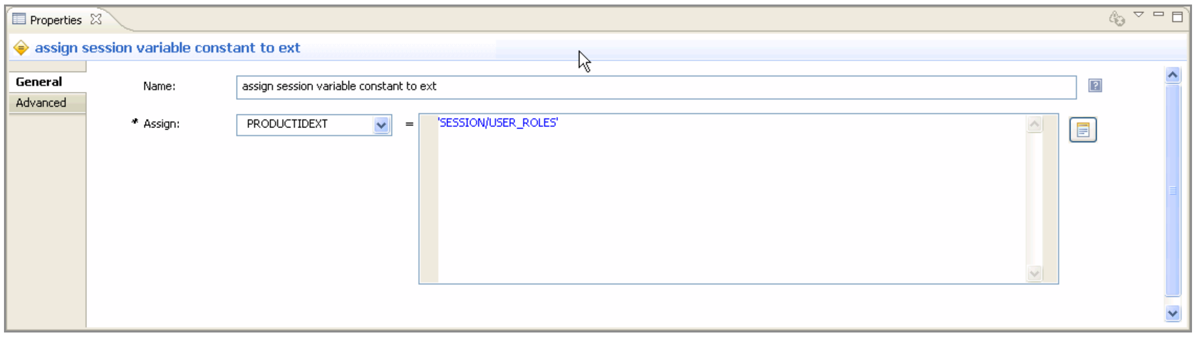
Double click the Cnt_Conditional Assign Action container, the rulebase opens in a new editor.

Assign Action with Conditional General Properties



Expression: !in(SALARY, {80000})

Assign Action with Else condition General Properties



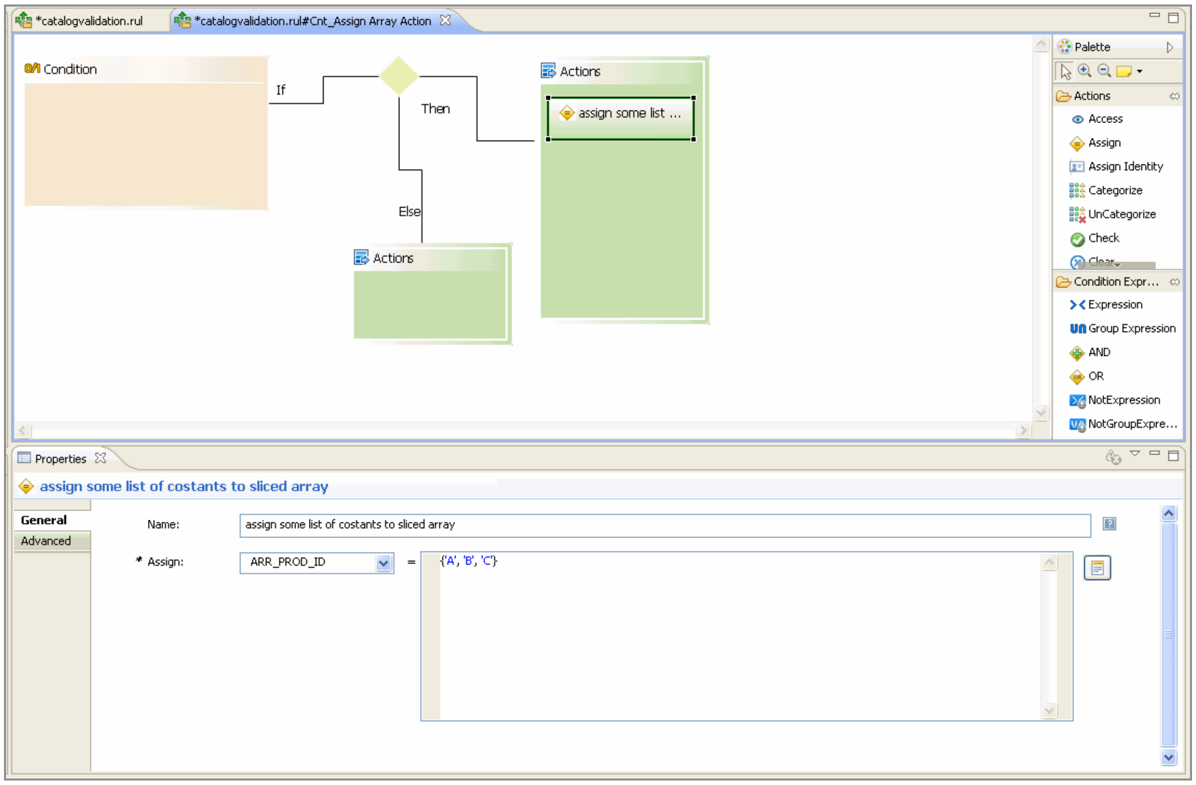
Name: assign session variable constant to ext

Assign: PRODUCTIDEXT = 'SESSION/USER_ROLES'

Assign Action

In this example, a Assign action with arrays is defined.

Assign Action with array with General Properties



assign some list of costants to sliced array

Name: assign some list of costants to sliced array

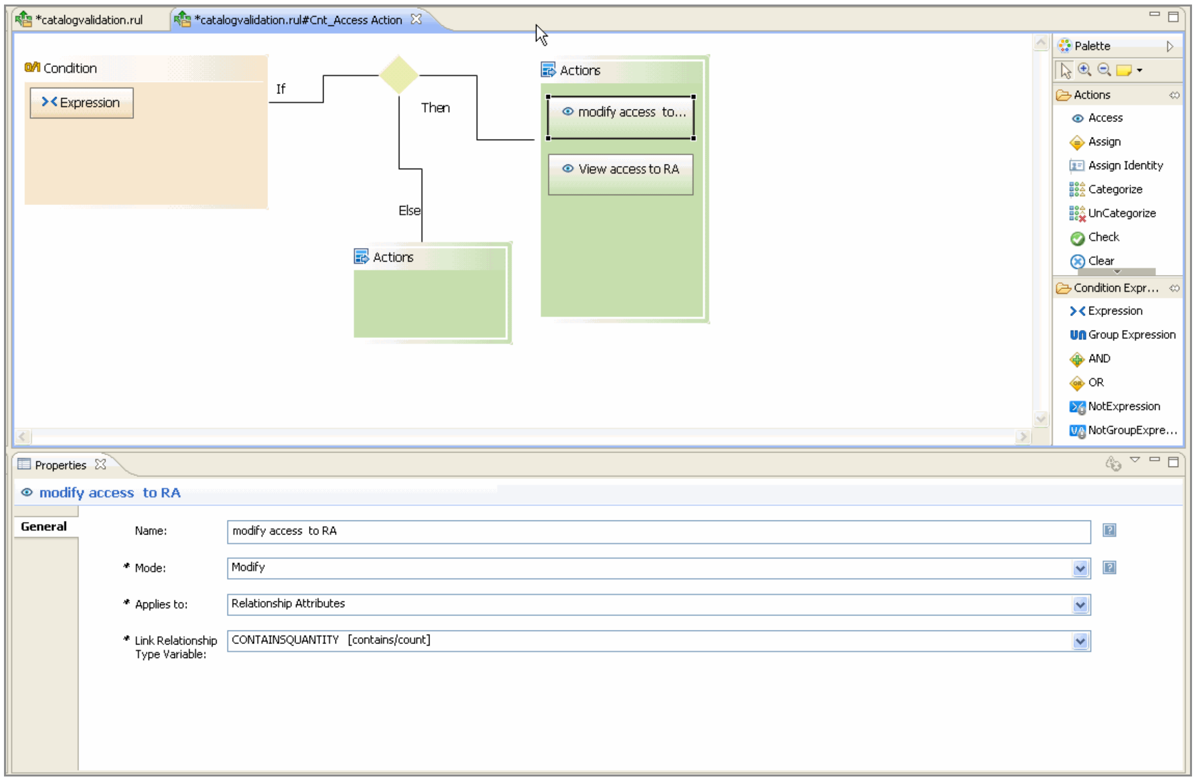
* Assign: ARR_PROD_ID = {'A', 'B', 'C'}

Access Action

In this example a access action is defined.

If a person is a manager, modify access is provided for the contains count and view access is for containby count.

Access Action Modify with General Properties



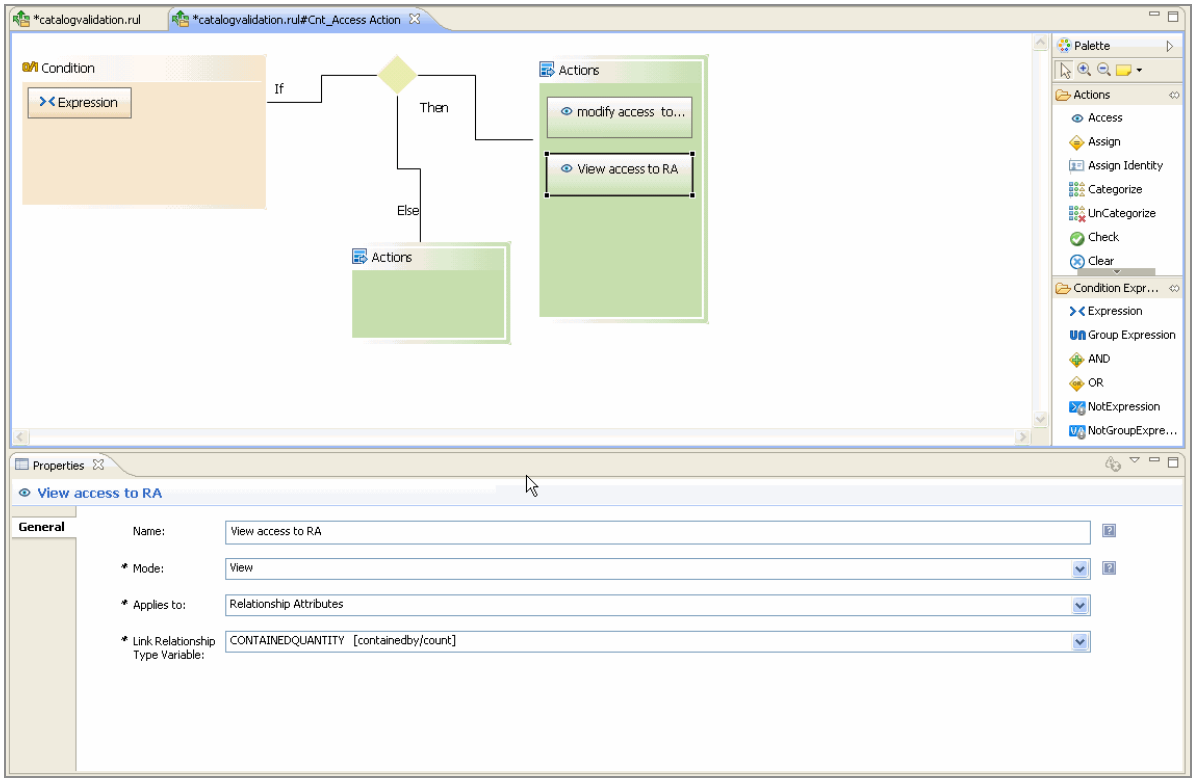
Name: modify access to RA

* Mode: Modify

* Applies to: Relationship Attributes

* Link Relationship Type Variable: CONTAINSQUANTITY [contains/count]

Access Action View with General Properties



View access to RA

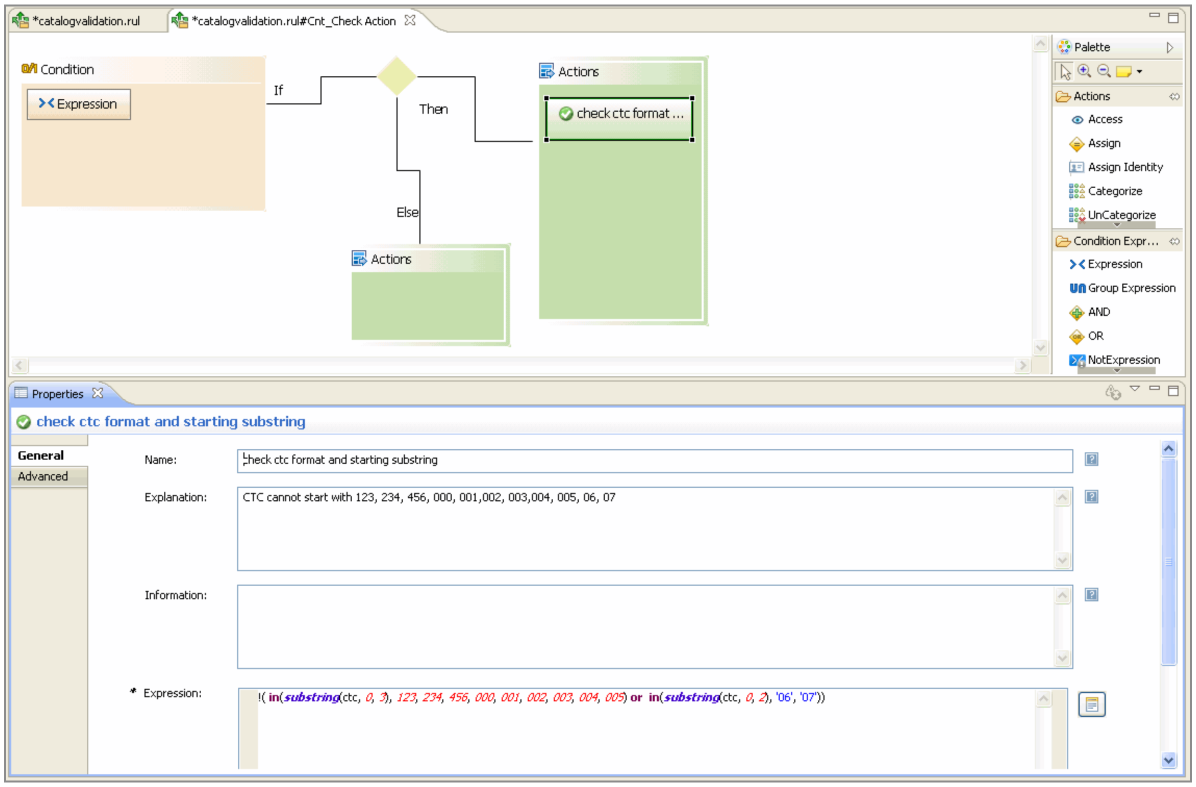
Name:	View access to RA
* Mode:	View
* Applies to:	Relationship Attributes
* Link Relationship Type Variable:	CONTAINEDQUANTITY [containedby/count]

Check Action

In this example, a check action is defined.

A check is done to check the CTC format and the starting substring.

Check Action with General Properties



check ctc format and starting substring

Name: check ctc format and starting substring

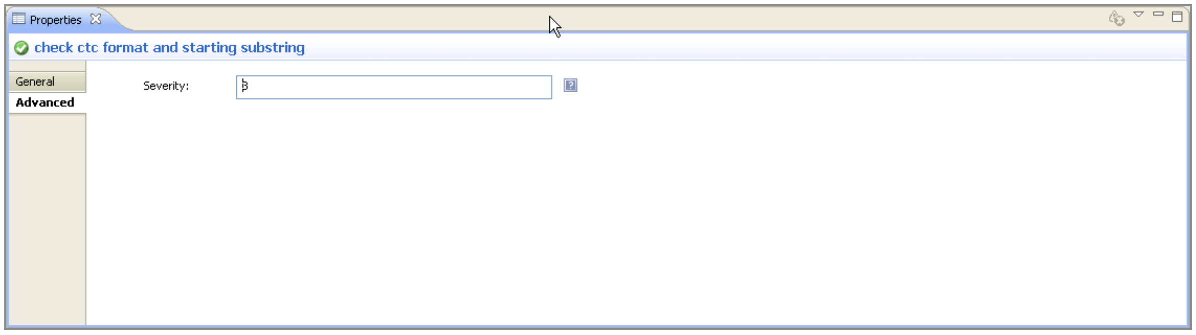
Explanation: CTC cannot start with 123, 234, 456, 000, 001,002, 003,004, 005, 06, 07

Information:

* Expression:

```
!( in(substring(ctc, 0, 3), 123, 234, 456, 000, 001, 002, 003, 004, 005) or in(substring(ctc, 0, 2), '06', '07'))
```

Check Action Rule Advanced Properties

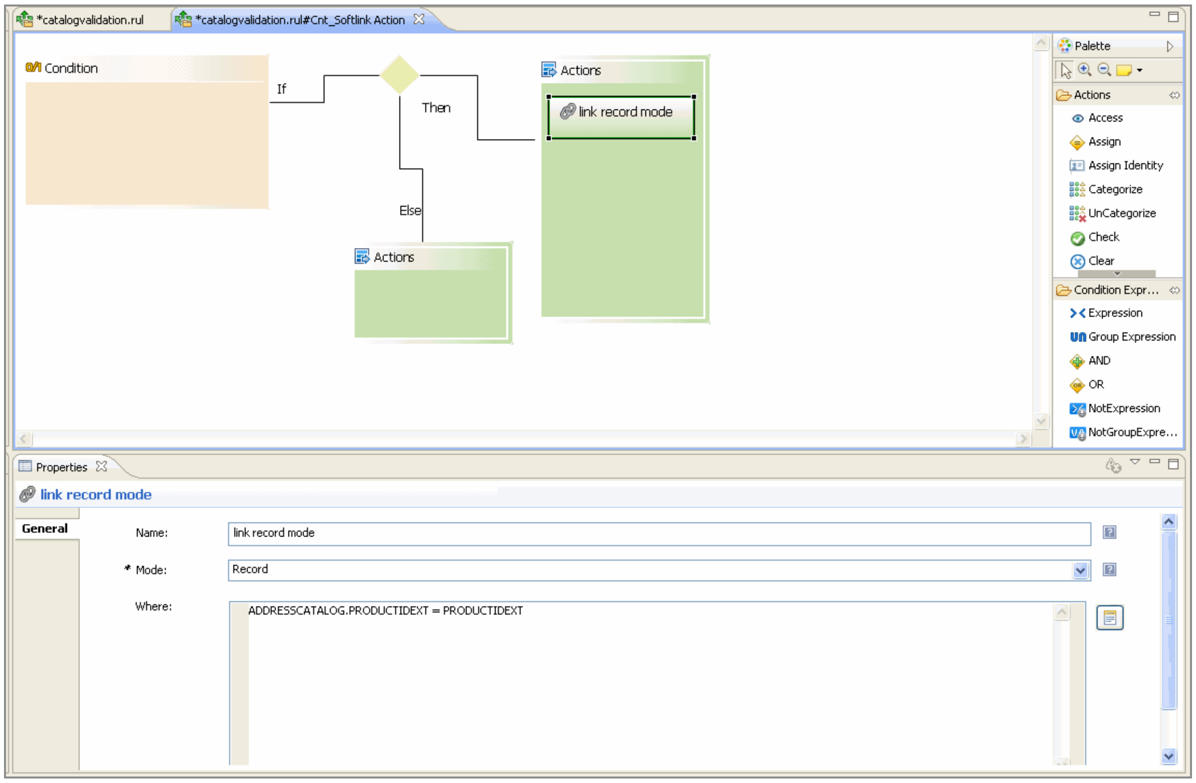


check ctc format and starting substring

Severity: 3

Softlink Action

In this example, a softlink action is defined to return ADDRESS records having the CUSTOMERID similar to the record being processed.



Name: link record mode

* Mode: Record

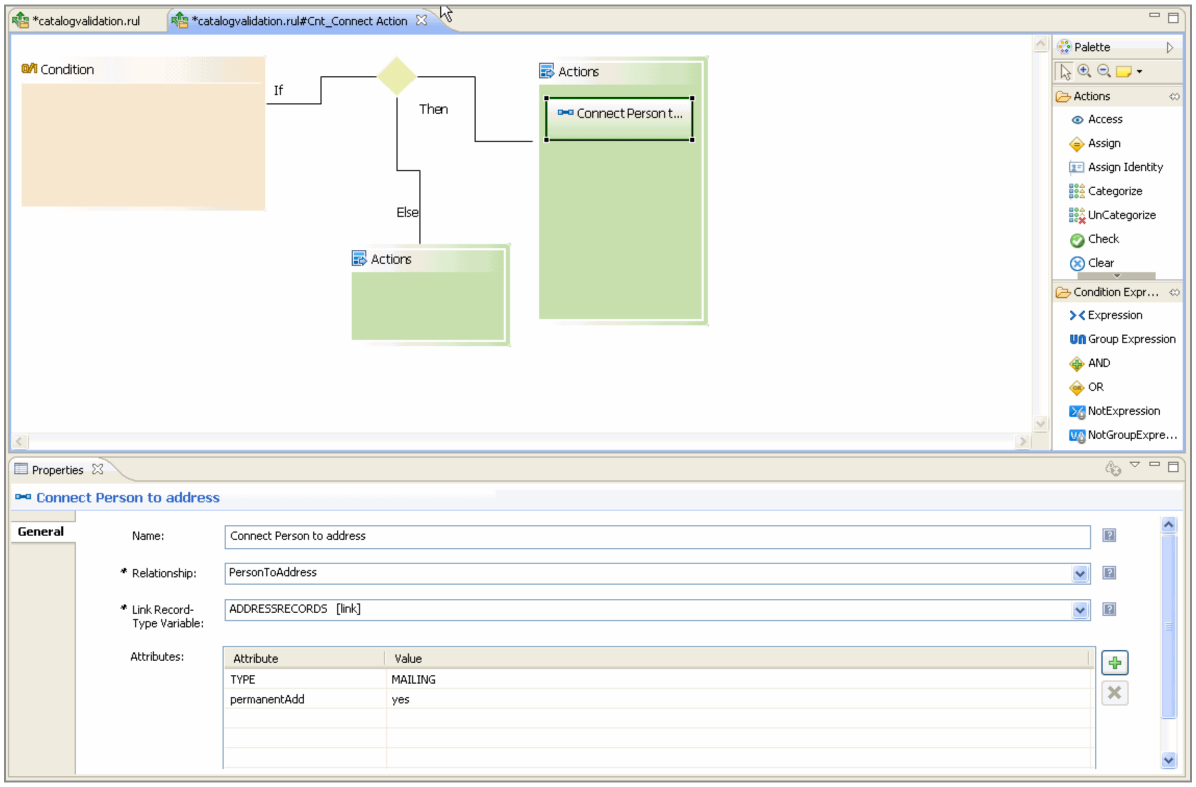
Where: ADDRESSCATALOG.PRODUCTIDEXT = PRODUCTIDEXT

Connect Action

In this example, a connect action is defined.

The address records are connected using the Person to Address relationship.

Connect Action with General Properties



Connect Person to address

Name: Connect Person to address

* Relationship: PersonToAddress

* Link Record-Type Variable: ADDRESSRECORDS [link]

Attributes:

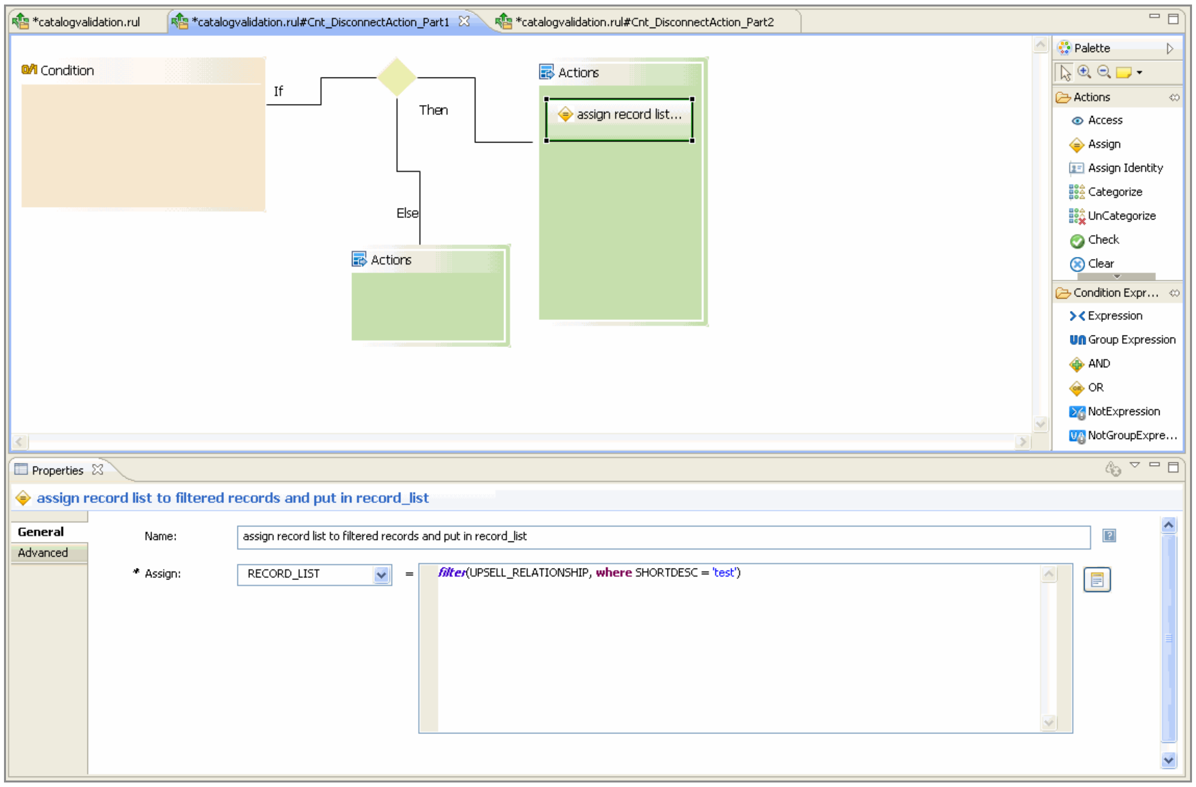
Attribute	Value
TYPE	MAILING
permanentAdd	yes

Disconnect Action

In this example, a disconnect action is defined.

A list of records is obtained to disconnect and is assigned to variable.

Disconnect Action with assigned records with General Properties

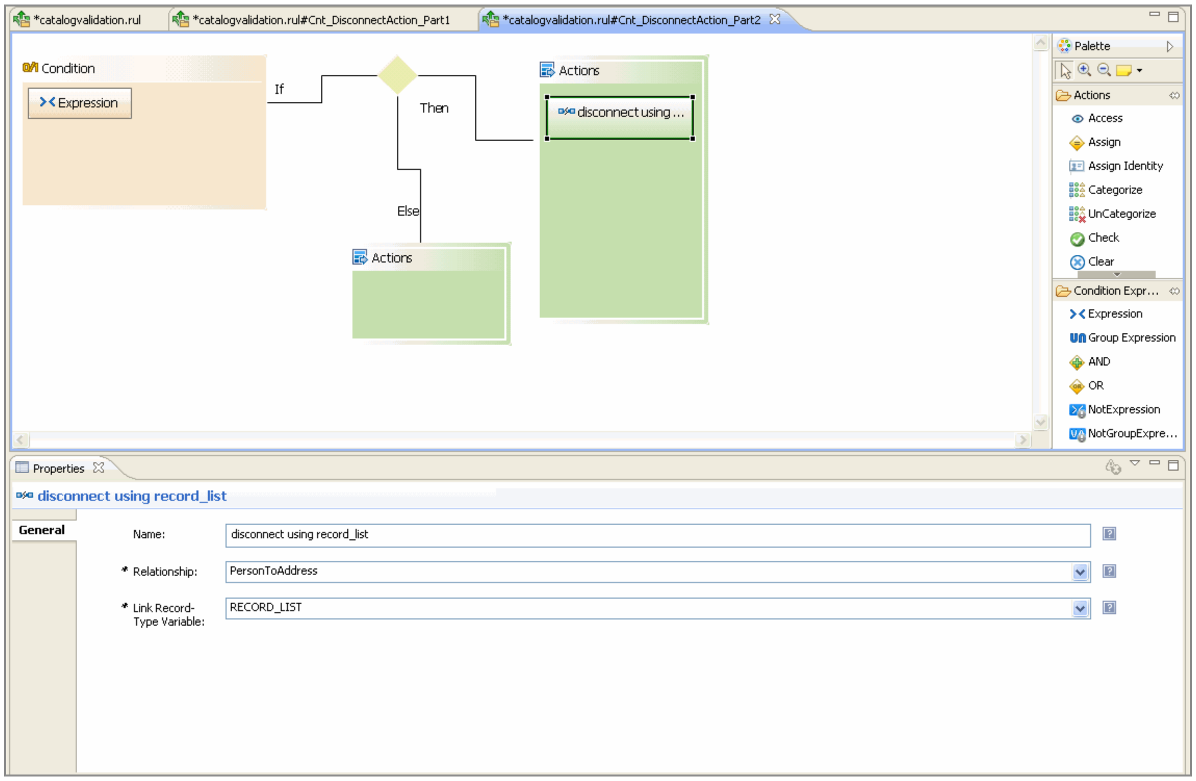


assign record list to filtered records and put in record_list

Name: assign record list to filtered records and put in record_list

* Assign: RECORD_LIST = filter(UPSELL_RELATIONSHIP, where SHORTDESC = 'test')

Disconnect action using record list with General Properties



disconnect using record_list

Name: disconnect using record_list

* Relationship: PersonToAddress

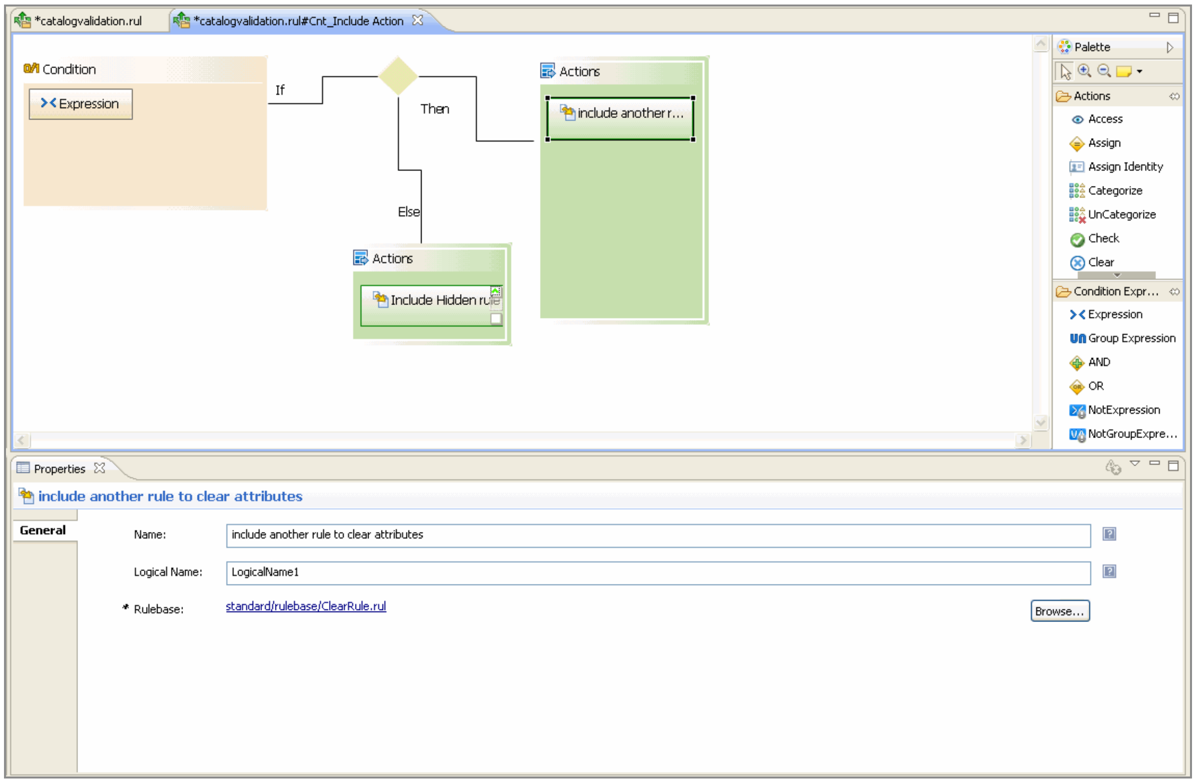
* Link Record-Type Variable: RECORD_LIST

Include Action

In this example, a include action is defined.

If last name is defined then include another rule to clear attributes else include hidden rule.

Include Action rule to clear attributes



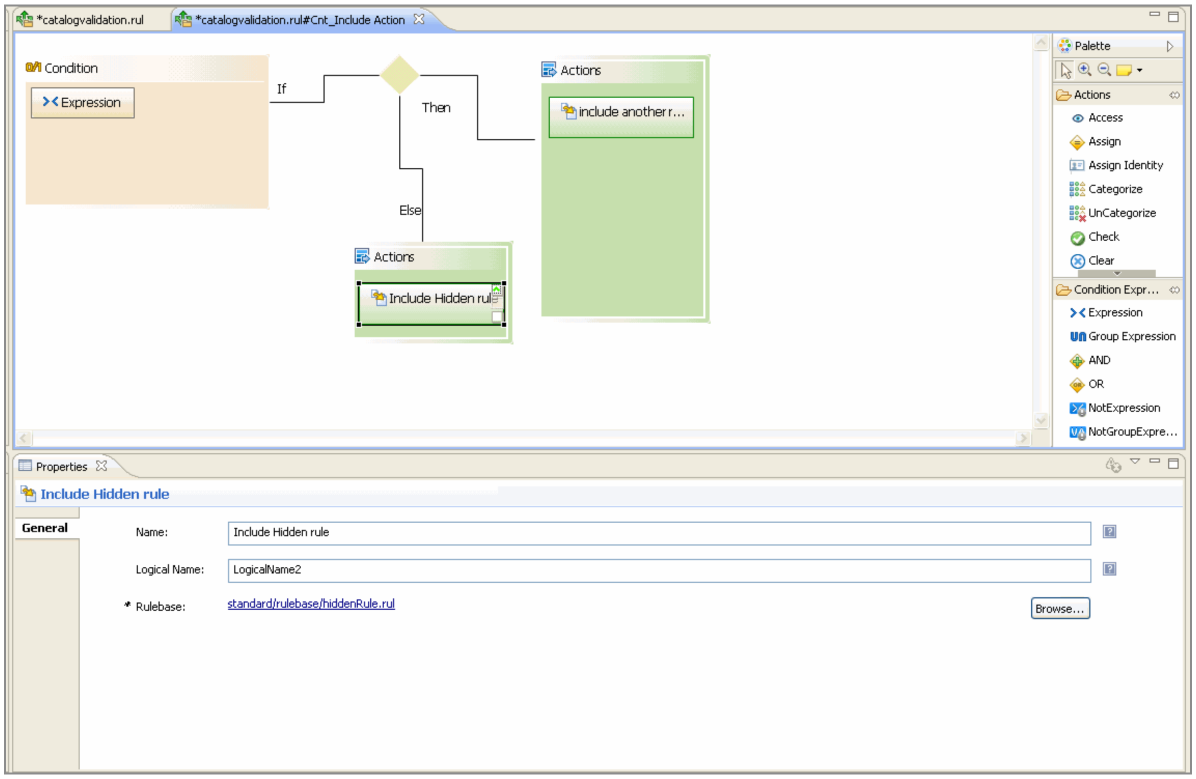
include another rule to clear attributes

Name: include another rule to clear attributes

Logical Name: LogicalName1

* Rulebase: standard/rulebase/ClearRule.rul

Include Action hidden rule



Name: Include Hidden rule

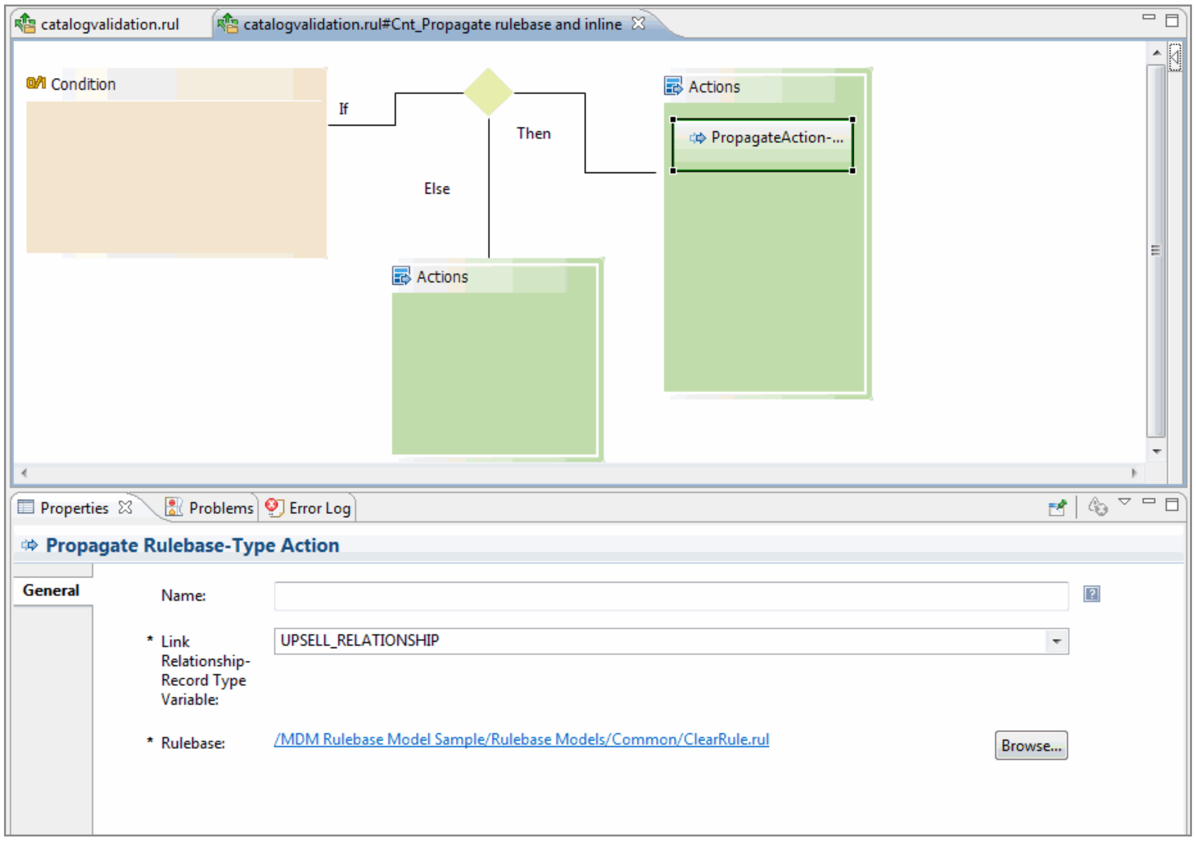
Logical Name: LogicalName2

* Rulebase: standard/rulebase/hiddenRule.rul

Propagate rulebase and inline Action

In this example, a propagate action is defined.

Propagate Action with General Properties



Name:

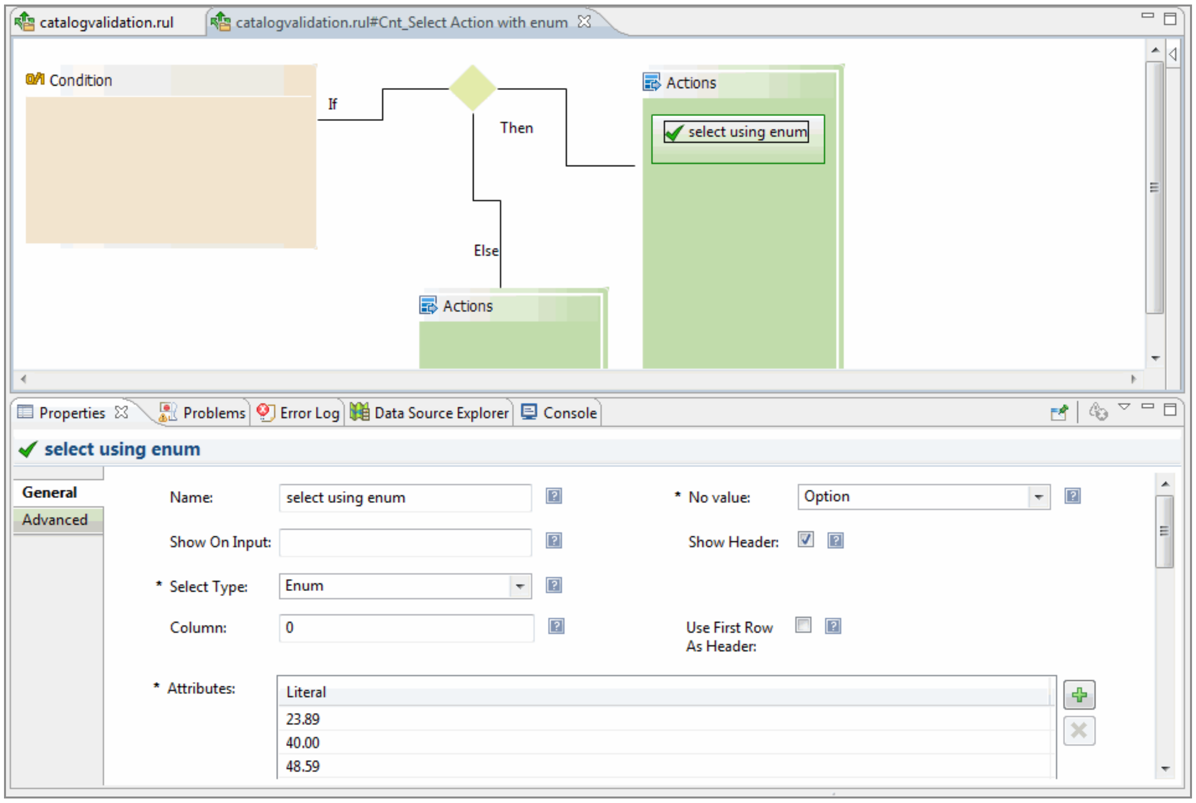
* Link Relationship-Record Type Variable: UPSELL_RELATIONSHIP

* Rulebase: /MDM Rulebase Model Sample/Rulebase Models/Common/ClearRule.rul

Select Action enum

In this example, a select action with enum is defined.

Select Action with enum with General Properties



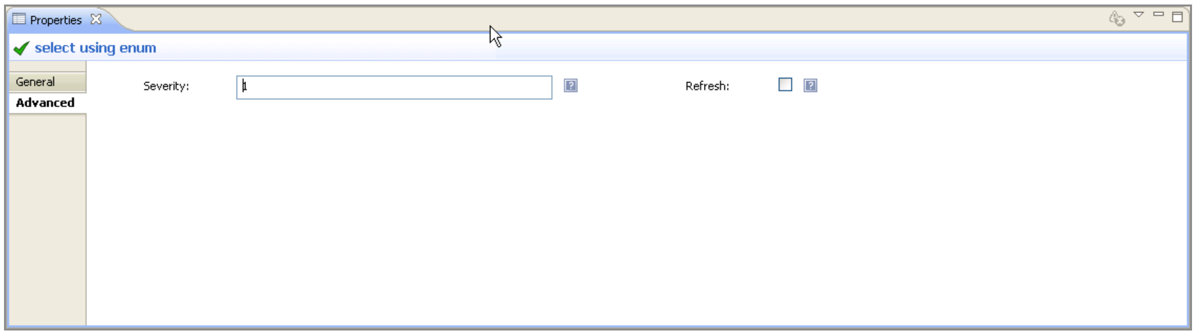
select using enum

Name:	select using enum		* No value:	Option	
Show On Input:			Show Header:	✓	
* Select Type:	Enum				
Column:	0		Use First Row As Header:	☐	

* Attributes:

Literal
23.89
40.00
48.59

Select Action Rule Advanced Properties



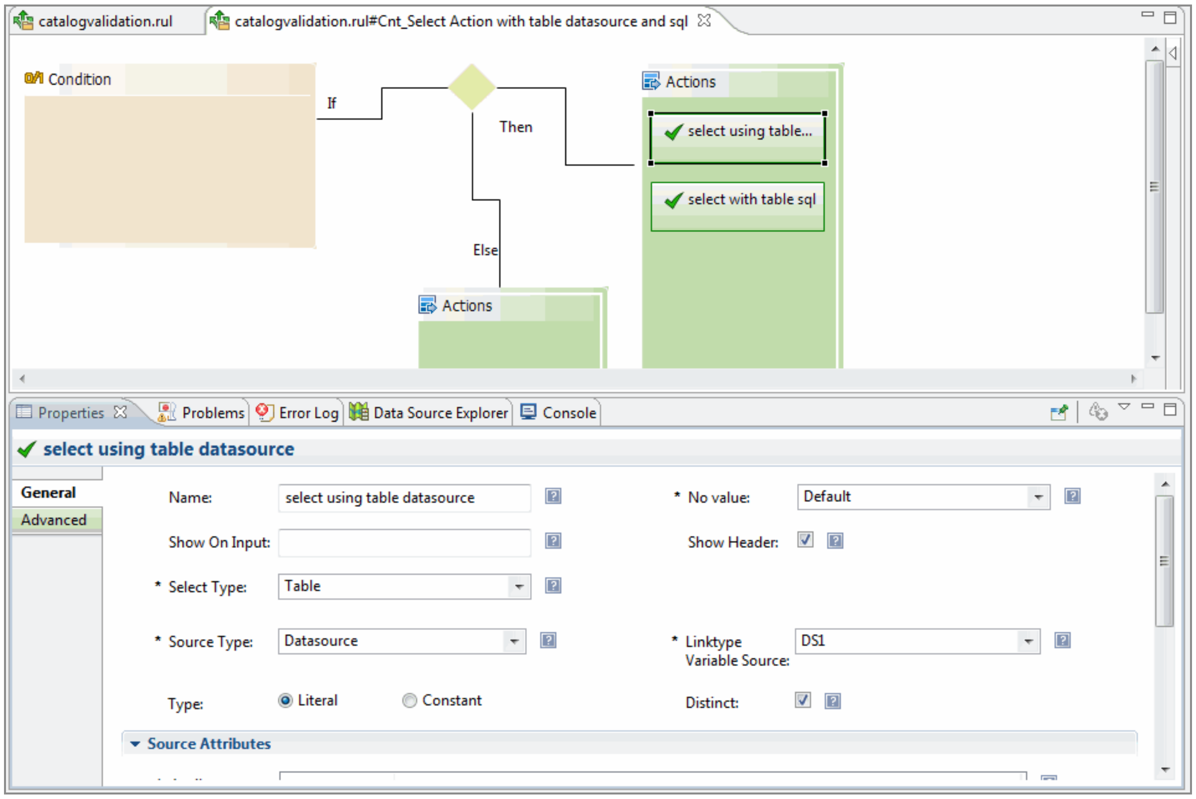
select using enum

Severity:	1	Refresh:	☐

Select Action Tables

In this example, a select action with table datasource and sql is defined.

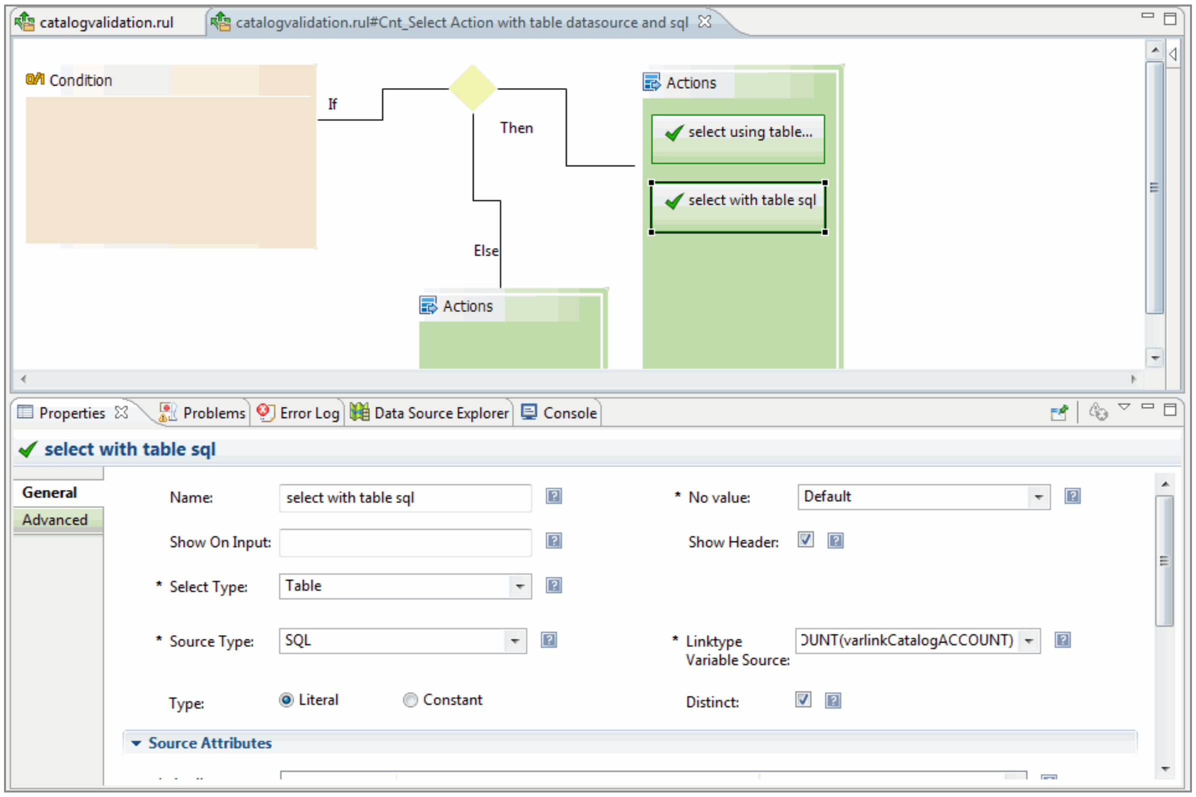
Select Action with table datasource table



select using table datasource

Name:	select using table datasource	* No value:	Default
Show On Input:		Show Header:	✓
* Select Type:	Table		
* Source Type:	Datasource	* Linktype Variable Source:	DS1
Type:	[✓ Literal] Constant	Distinct:	✓

Select Action with SQL table



select with table sql

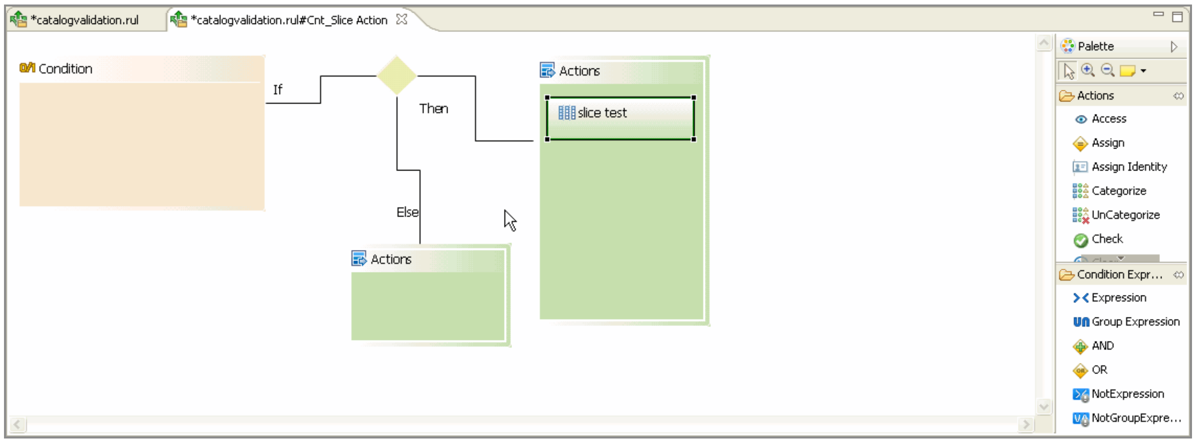
Name:	select with table sql	* No value:	Default
Show On Input:		Show Header:	✓
* Select Type:	Table		
* Source Type:	SQL	* Linktype Variable Source:	OUNT(varlinkCatalogACCOUNT)
Type:	[✓ Literal] Constant	Distinct:	✓

Source Attributes

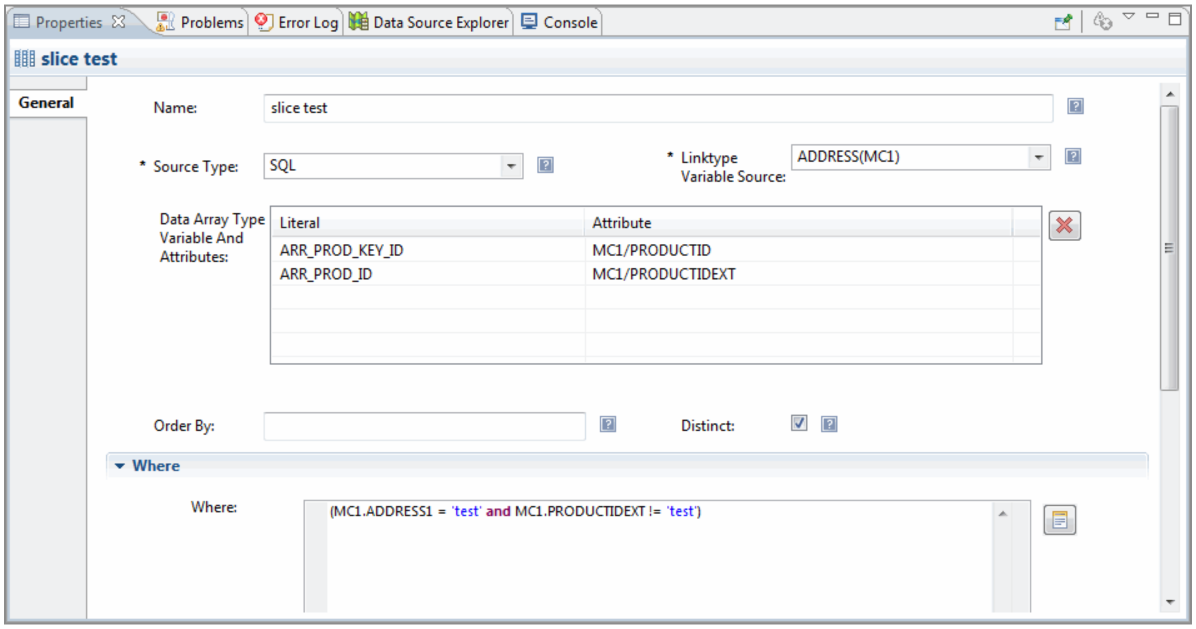
Slice Action

In this example, a slice action is defined.

Slice Action



Slice Action General Properties



slice test

Name: slice test

* Source Type: SQL

* Linktype Variable Source: ADDRESS(MC1)

Data Array Type Variable And Attributes:

Literal	Attribute
ARR_PROD_KEY_ID	MC1/PRODUCTID
ARR_PROD_ID	MC1/PRODUCTIDEXT

Order By:

Distinct: ✓

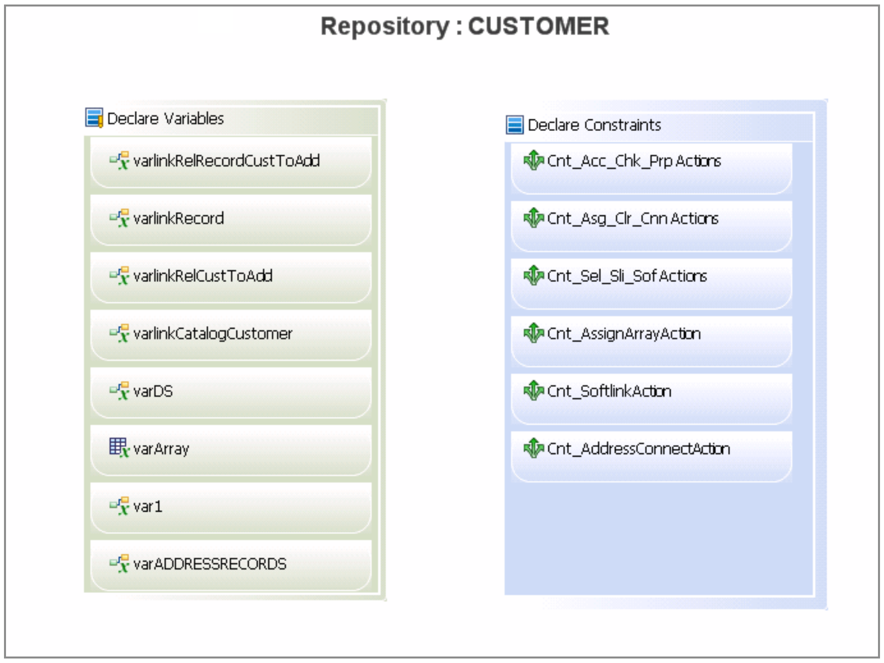
Where: (MC1.ADDRESS1 = 'test' and MC1.PRODUCTIDEXT != 'test')

Sample - 2

The following is a sample rulebase which has all the commonly used action.

The Rulebase type used in this sample is of type validation.

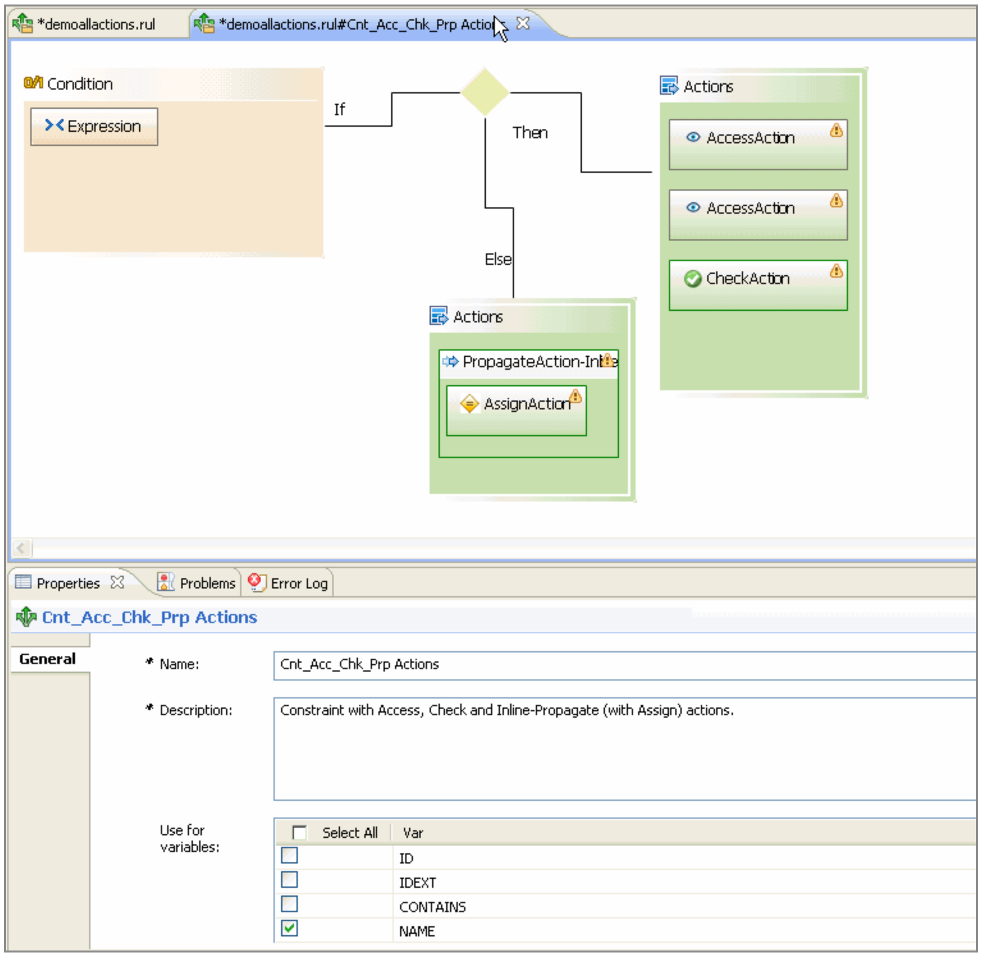
Rulebase Sample 2



Constraint with Access Check and Inline-Propagate (with Assign) actions

In this example, a constraint with access, check and inline-propagate (with assign) actions are described.

Constraint with Action, Check and Inline propagate action



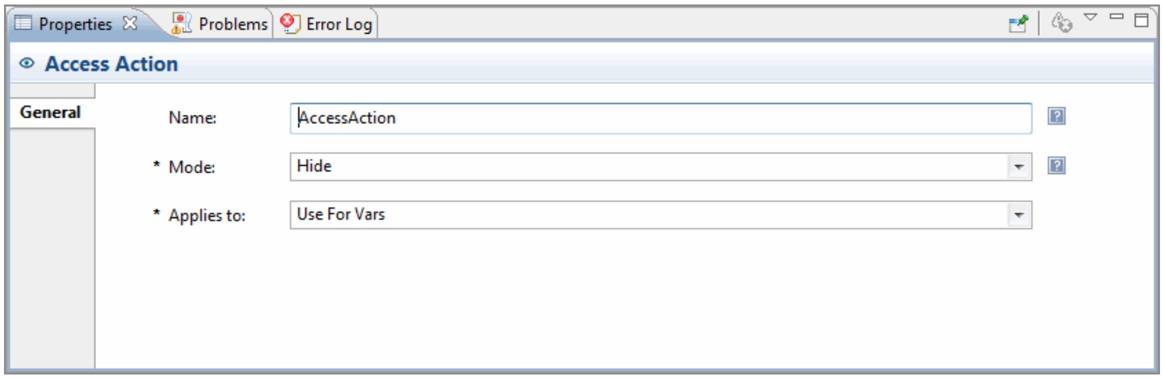
* Name: Cnt_Acc_Chk_Prp Actions

* Description: Constraint with Access, Check and Inline-Propagate (with Assign) actions.

Use for variables:

Select All	Var
☐	ID
☐	IDEXT
☐	CONTAINS
✓	NAME

Access Properties



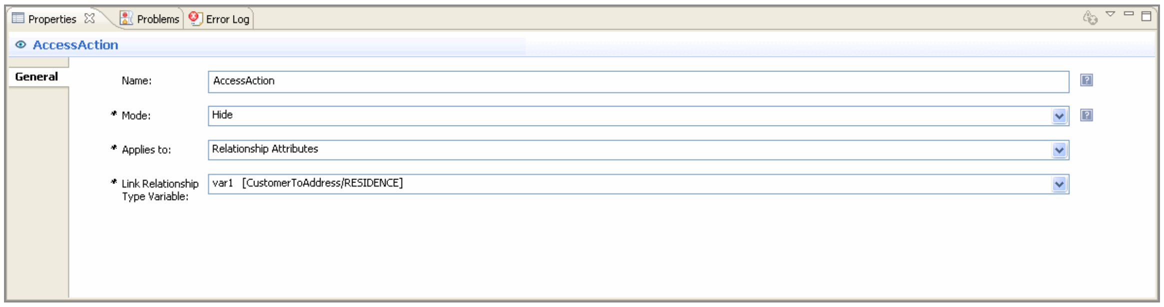
Access Action

Name: AccessAction

* Mode: Hide

* Applies to: Use For Vars

Access Action Properties



AccessAction

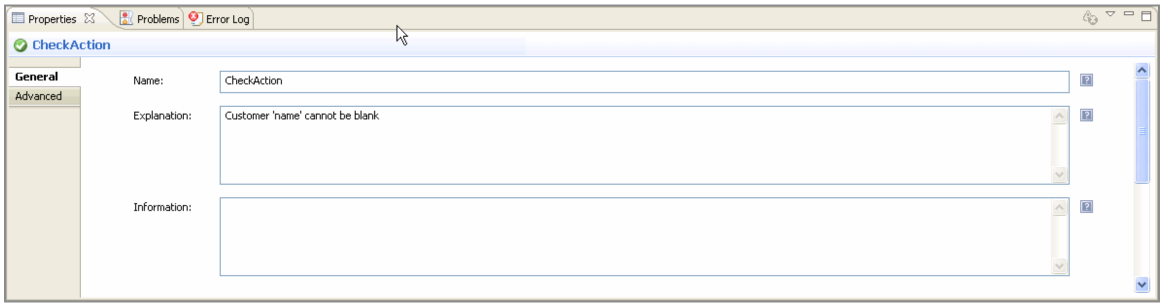
Name: AccessAction

* Mode: Hide

* Applies to: Relationship Attributes

* Link Relationship Type Variable: var1 [CustomerToAddress/RESIDENCE]

Check Action Properties



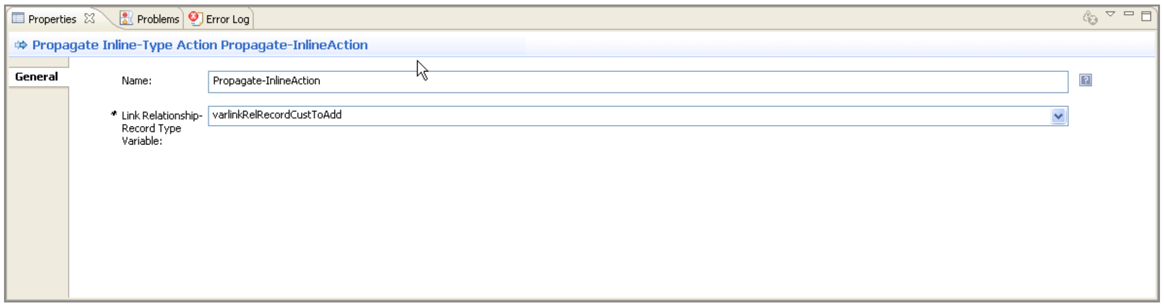
CheckAction

Name: CheckAction

Explanation: Customer 'name' cannot be blank

Information:

Propagate InlineAction Properties

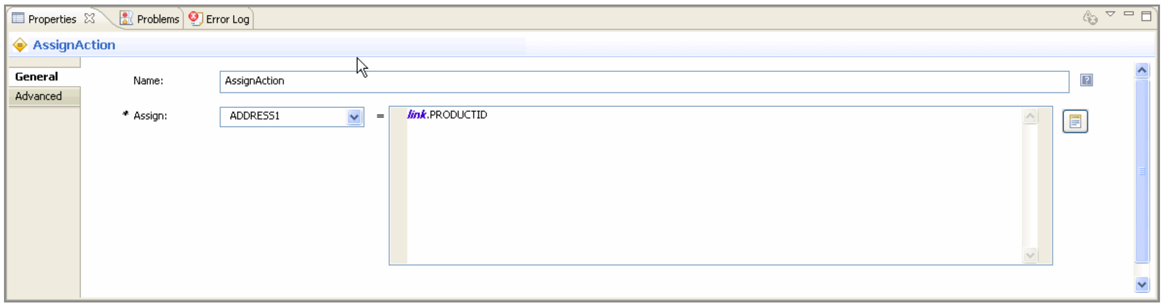


Propagate Inline-Type Action Propagate-InlineAction

Name: Propagate-InlineAction

* Link Relationship-Record Type Variable: varlinkRelRecordCustToAdd

Assign Action Properties



AssignAction

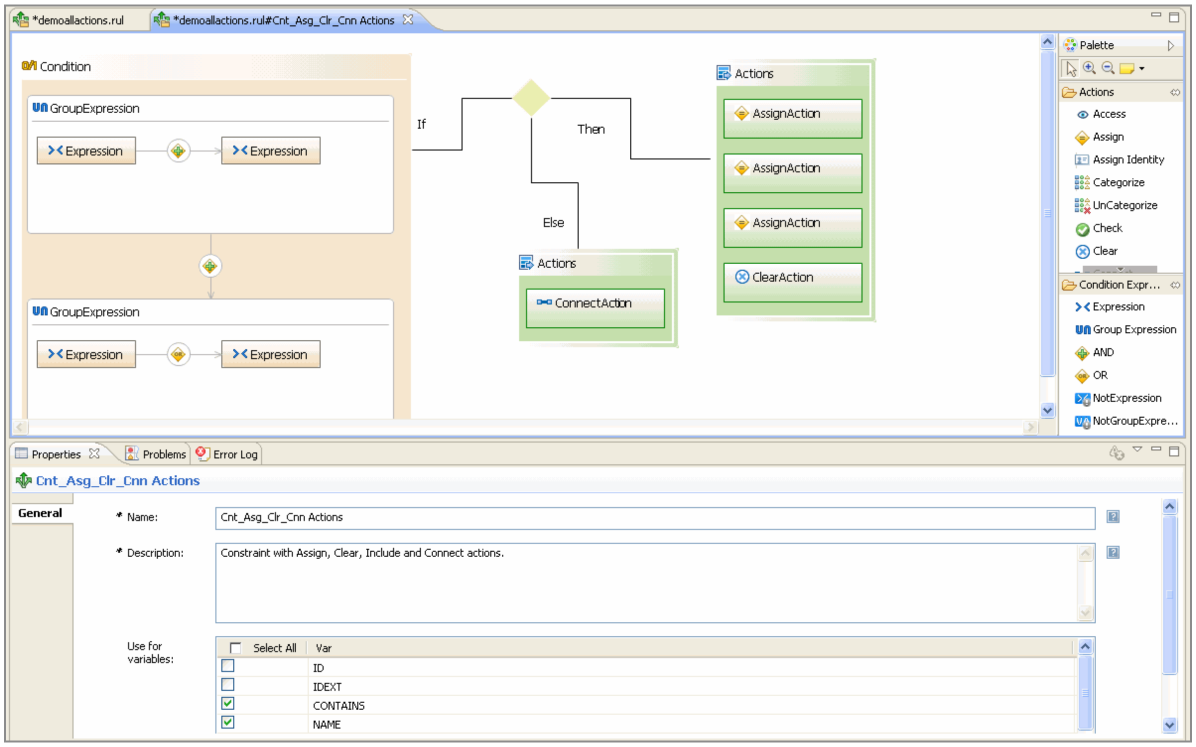
Name: AssignAction

* Assign: ADDRESS1 = *link*.PRODUCTID

Constraint with Assign Clear Include and Connect actions

In this example constraint with assign, clear, include, and connect actions are described.

Constraint with Assign, Clear, Include and Connect actions



Cnt_Asg_Clr_Cnn Actions

General

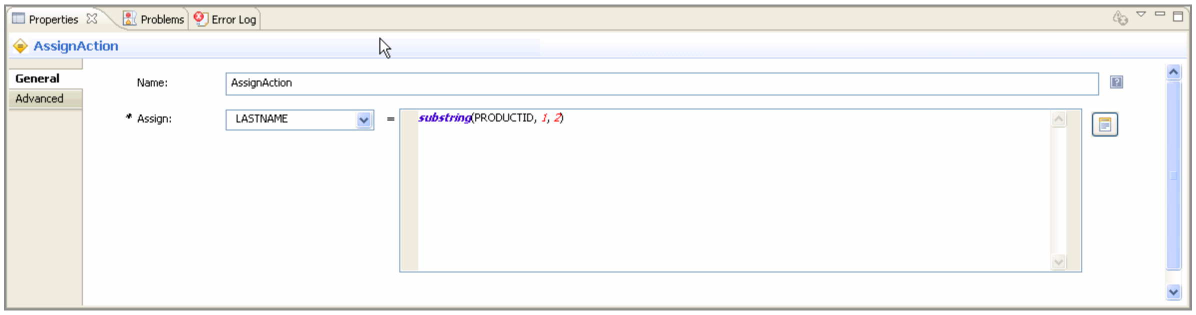
* Name: Cnt_Asg_Clr_Cnn Actions

* Description: Constraint with Assign, Clear, Include and Connect actions.

Use for variables:

Select All	Var
☐	ID
☐	IDEXT
✓	CONTAINS
✓	NAME

Assign Action General Properties

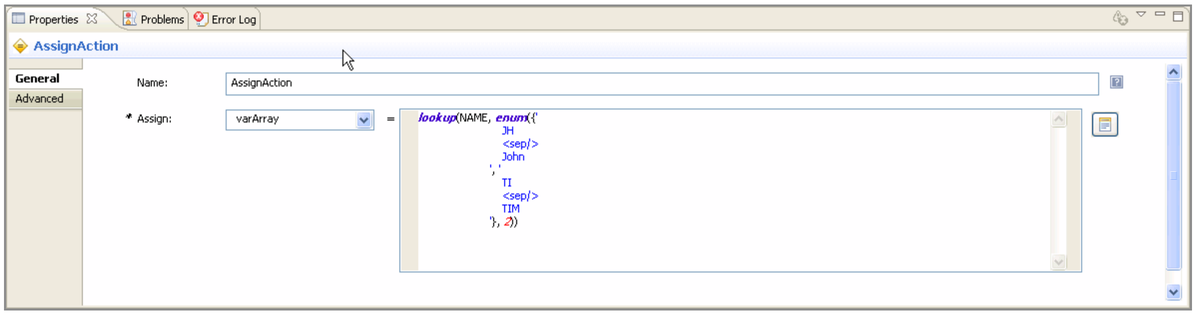


AssignAction

Name: AssignAction

* Assign: LASTNAME = *substring*(PRODUCTID, 1, 2)

Assign Action with Array General Properties



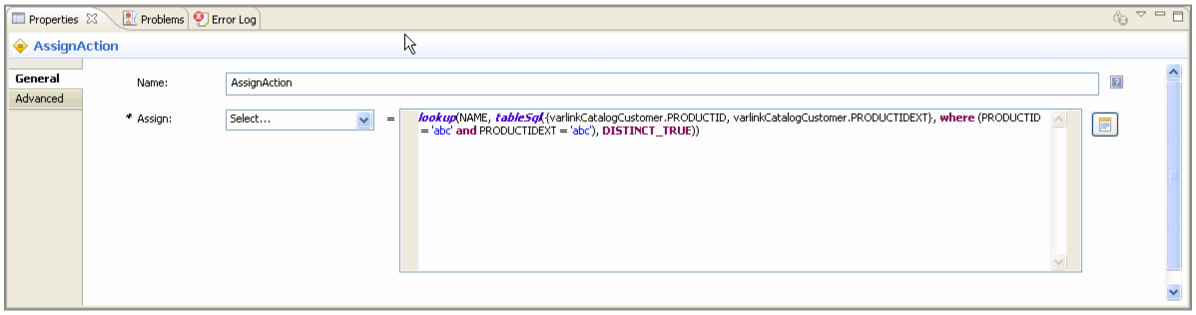
AssignAction

Name: AssignAction

* Assign: varArray =

```
lookup(NAME, enum({'
              JH
              <sep/>
              John
            ', '
              TI
              <sep/>
              TIM
            '}, 2))
```

Assign Action General Properties

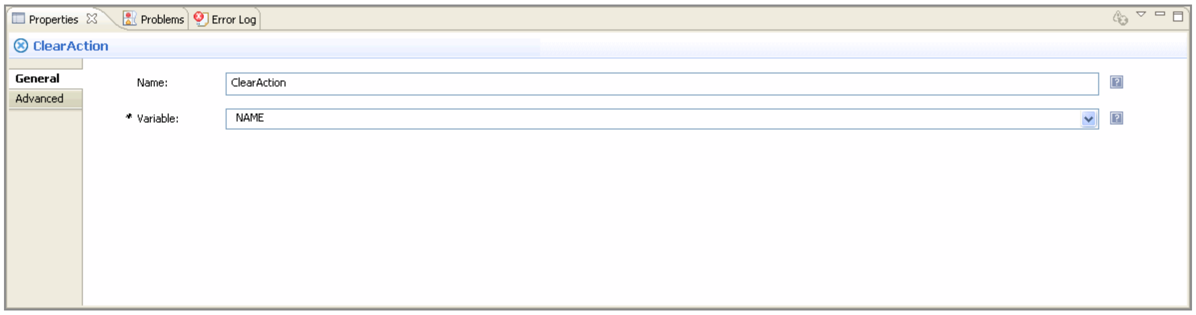


Name: AssignAction

* Assign: Select... =

```
lookup(NAME, tableSql({varlinkCatalogCustomer.PRODUCTID, varlinkCatalogCustomer.PRODUCTIDEXT}, where (PRODUCTID
= 'abc' and PRODUCTIDEXT = 'abc'), DISTINCT_TRUE))
```

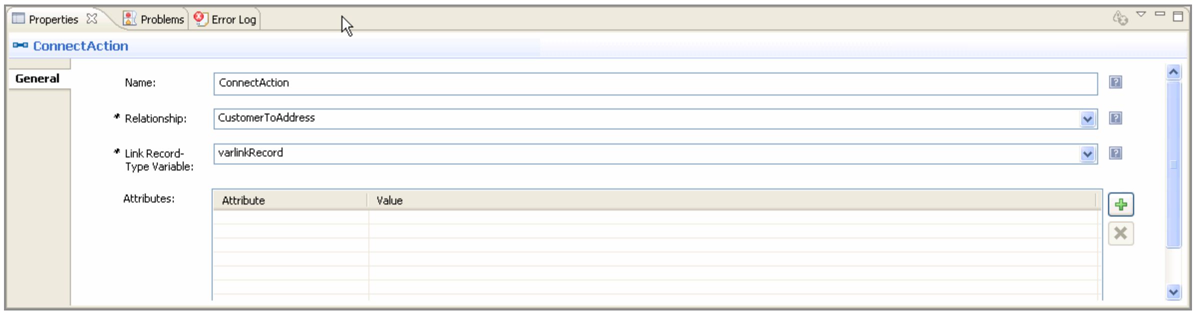
Clear Action General Properties



Name: ClearAction

* Variable: NAME

Connect Action General Properties



Name: ConnectAction

* Relationship: CustomerToAddress

* Link Record-Type Variable: varlinkRecord

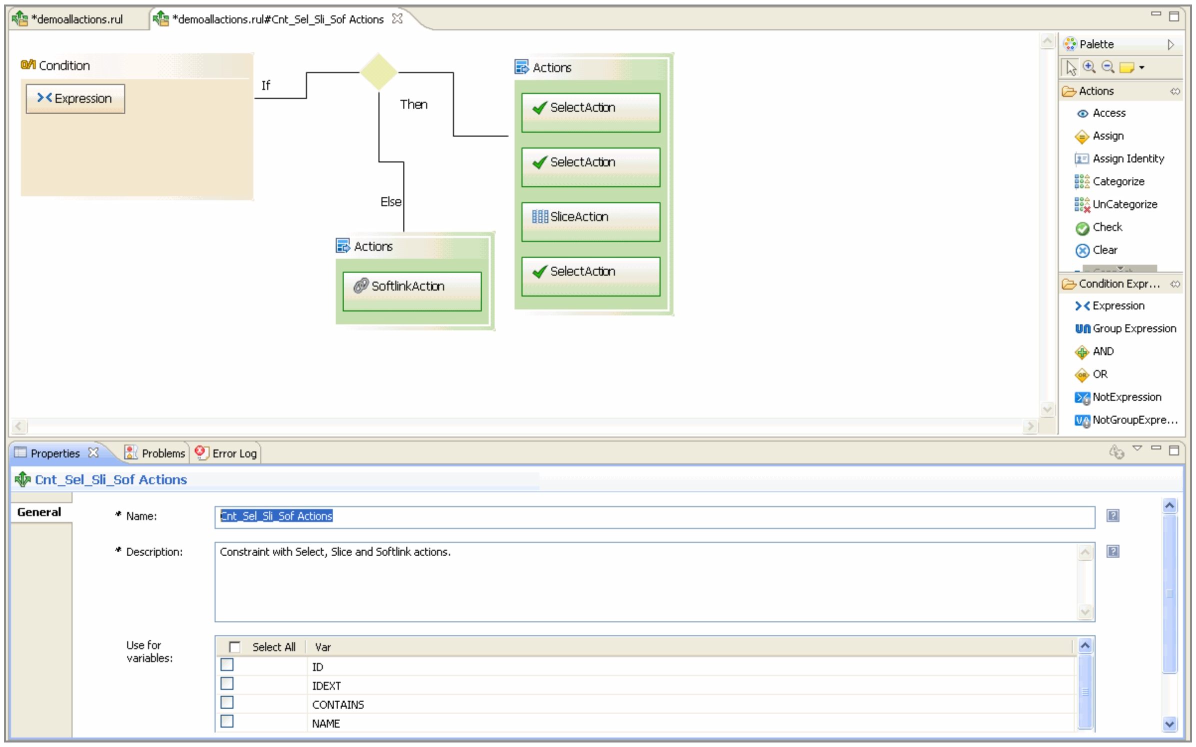
Attributes:

Attribute	Value

Constraint with Select Slice and Softlink actions

In this example constraint with select, slice and softlink actions are described.

Constraint with Select, Slice and Softlink actions



Cnt_Sel_Sli_Sof Actions

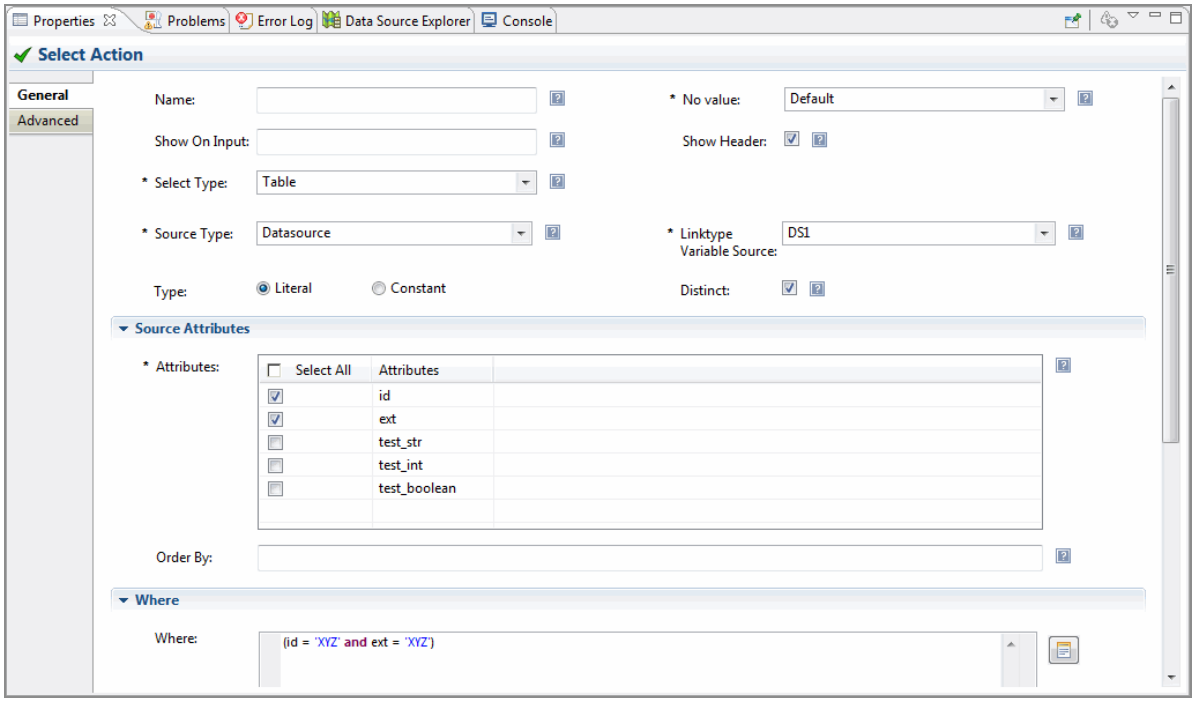
* Name: Cnt_Sel_Sli_Sof Actions

* Description: Constraint with Select, Slice and Softlink actions.

Use for variables:

☐ Select All	Var
☐	ID
☐	IDEXT
☐	CONTAINS
☐	NAME

Select Action General Properties



Select Action

Name:

Show On Input:

* Select Type: Table

* Source Type: Datasource

Type: [✓ Literal] Constant

* No value: Default

Show Header: ✓

* Linktype Variable Source: DS1

Distinct: ✓

Source Attributes

* Attributes:

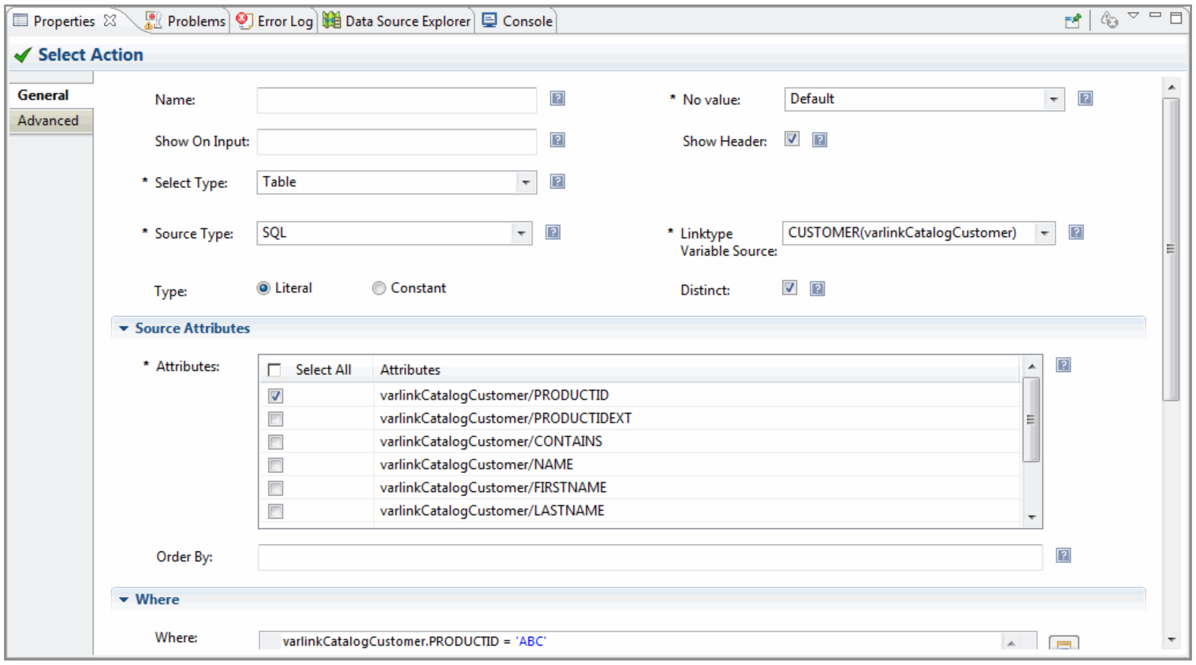
☐ Select All	Attributes
✓	id
✓	ext
☐	test_str
☐	test_int
☐	test_boolean

Order By:

Where

Where: (id = 'XYZ' and ext = 'XYZ')

Select Action General Properties



Select Action

Name:

Show On Input:

* Select Type: Table

* Source Type: SQL

Type: [✓ Literal] Constant

* No value: Default

Show Header: ✓

* Linktype Variable Source: CUSTOMER(varlinkCatalogCustomer)

Distinct: ✓

Source Attributes

* Attributes:

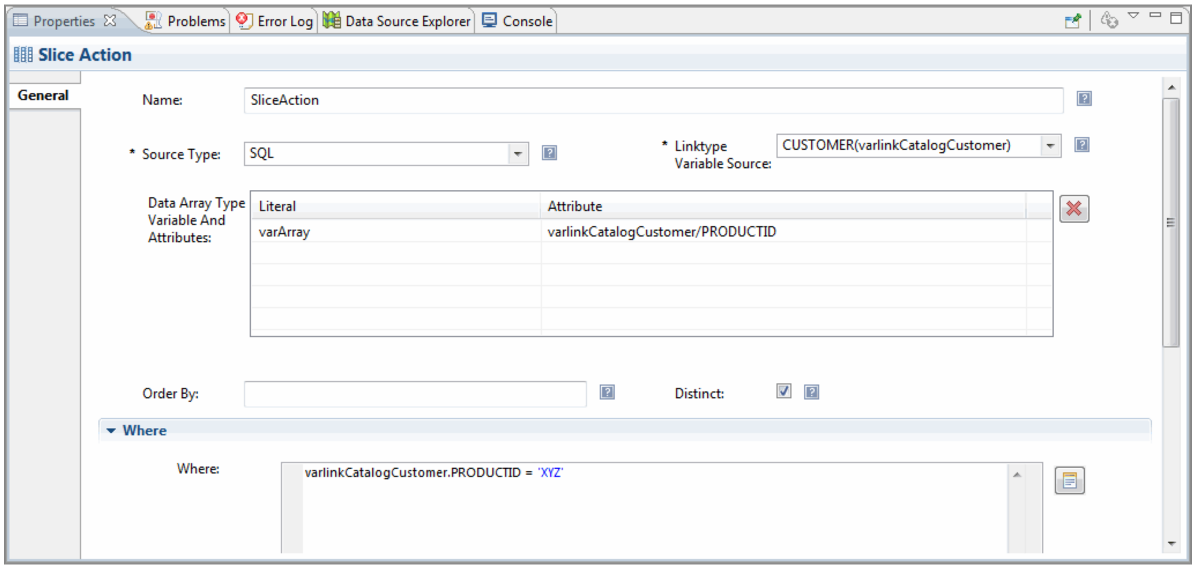
Select All	Attributes
✓	varlinkCatalogCustomer/PRODUCTID
	varlinkCatalogCustomer/PRODUCTIDEXT
	varlinkCatalogCustomer/CONTAINS
	varlinkCatalogCustomer/NAME
	varlinkCatalogCustomer/FIRSTNAME
	varlinkCatalogCustomer/LASTNAME

Order By:

Where

Where: varlinkCatalogCustomer.PRODUCTID = 'ABC'

Slice Action General Properties



Slice Action

Name: SliceAction

* Source Type: SQL

* Linktype Variable Source: CUSTOMER(varlinkCatalogCustomer)

Data Array Type Variable And Attributes:

Literal	Attribute
varArray	varlinkCatalogCustomer/PRODUCTID

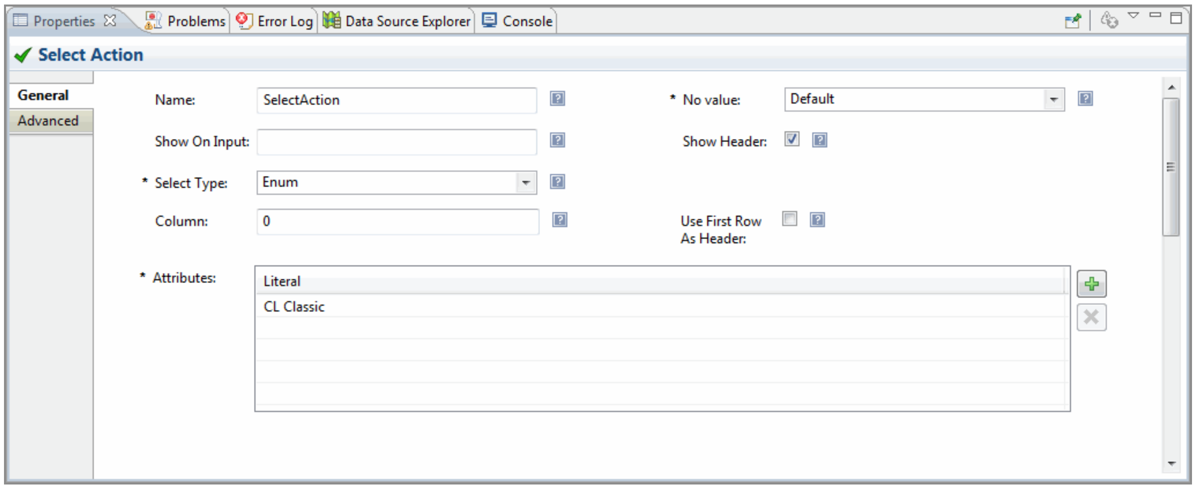
Order By:

Distinct: ✓

Where

Where: varlinkCatalogCustomer.PRODUCTID = 'XYZ'

Select Action General Properties



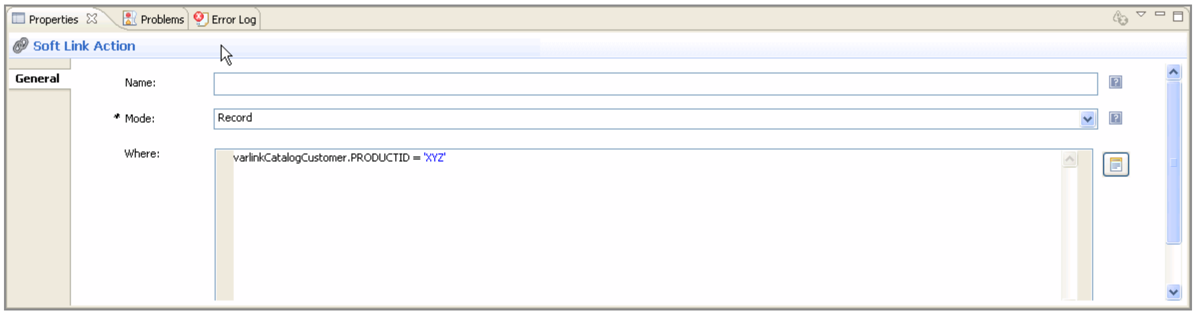
Select Action

Name:	SelectAction	* No value:	Default
Show On Input:		Show Header:	✓
* Select Type:	Enum		
Column:	0	Use First Row As Header:	

* Attributes:

Literal
CL Classic

Softlink Action General Properties



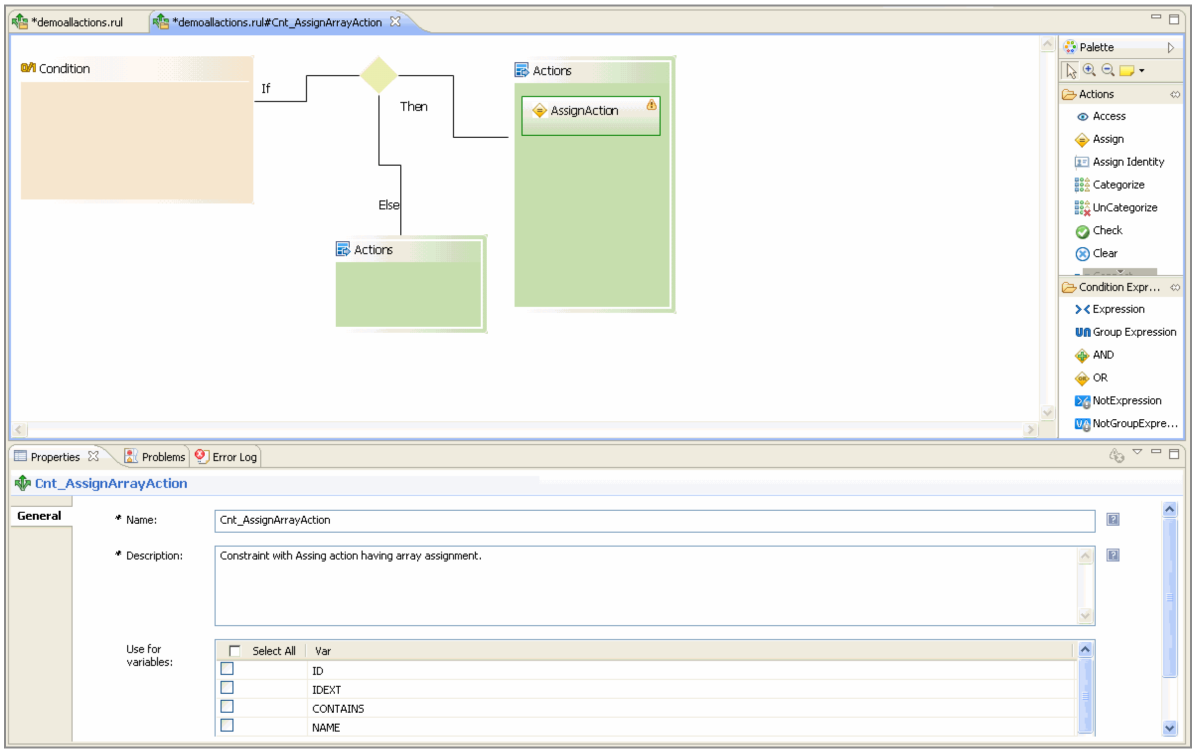
Soft Link Action

Name:	
* Mode:	Record
Where:	varlinkCatalogCustomer.PRODUCTID = 'XYZ'

Constraint with Assign action having array assignment

In this example a constraint with assign action having array assignment is described.

Constraint with assign action having array assignment



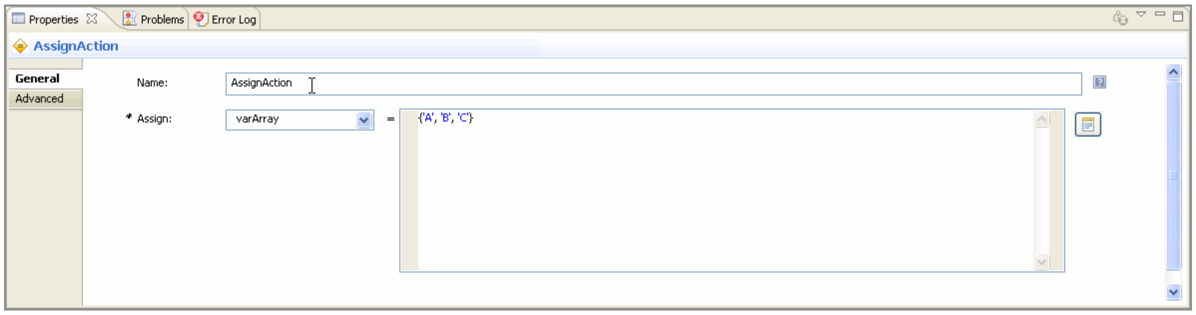
Name: Cnt_AssignArrayAction

Description: Constraint with Assing action having array assignment.

Use for variables:

Select All	Var
☐	ID
☐	IDEXT
☐	CONTAINS
☐	NAME

Assign Action General Properties



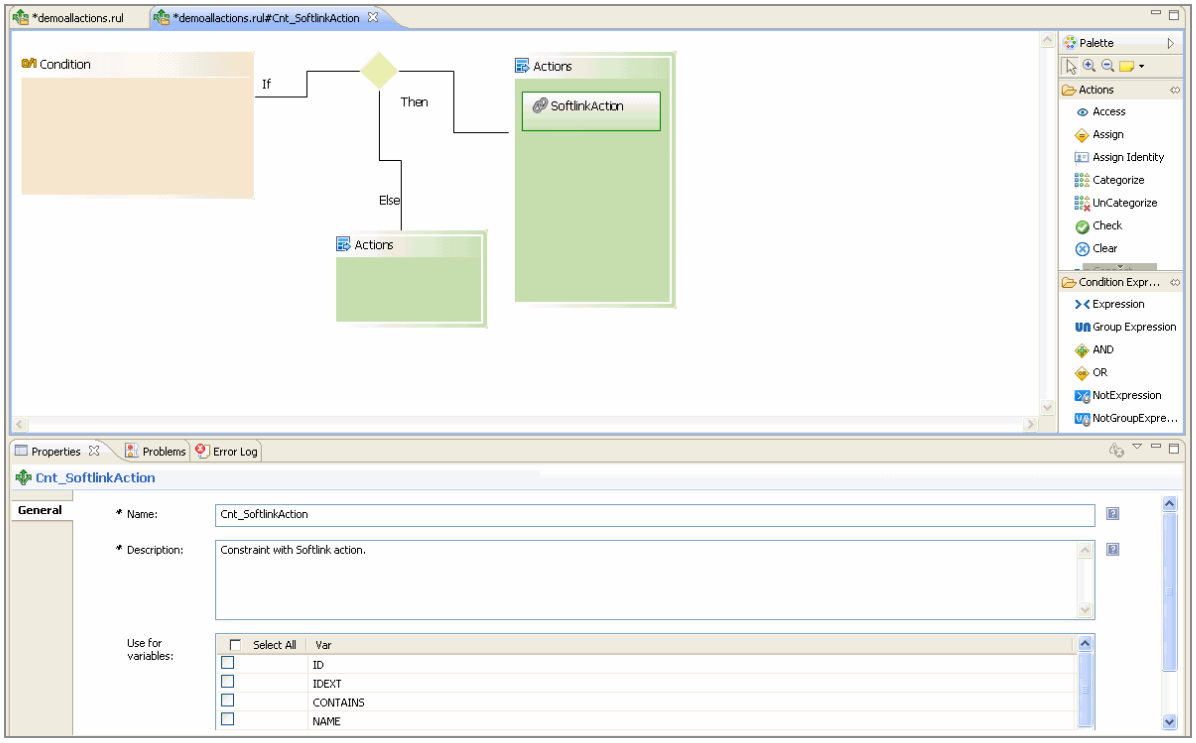
Name: AssignAction

Assign: varArray = {'A', 'B', 'C'}

Constraint with Softlink action

In this example constraint with softlink action is described.

Constraint with Softlink Action



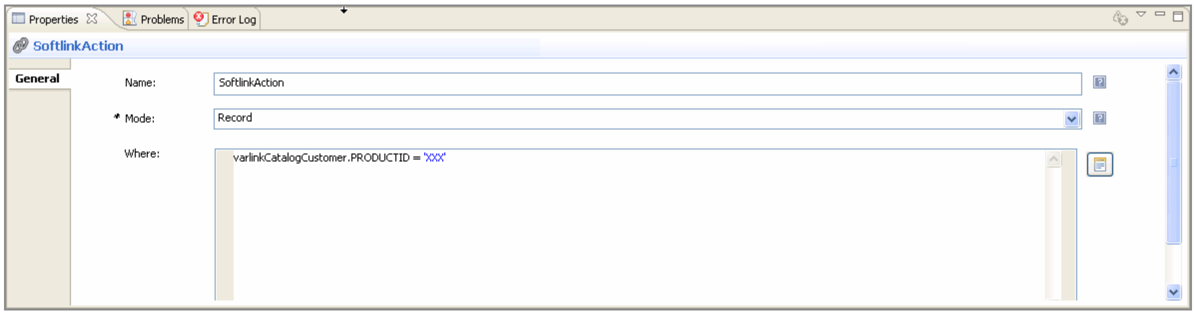
Cnt_SoftlinkAction

* Name:	Cnt_SoftlinkAction
* Description:	Constraint with Softlink action.

Use for variables:

Select All	Var
☐	ID
☐	IDEXT
☐	CONTAINS
☐	NAME

Softlink Action General Properties



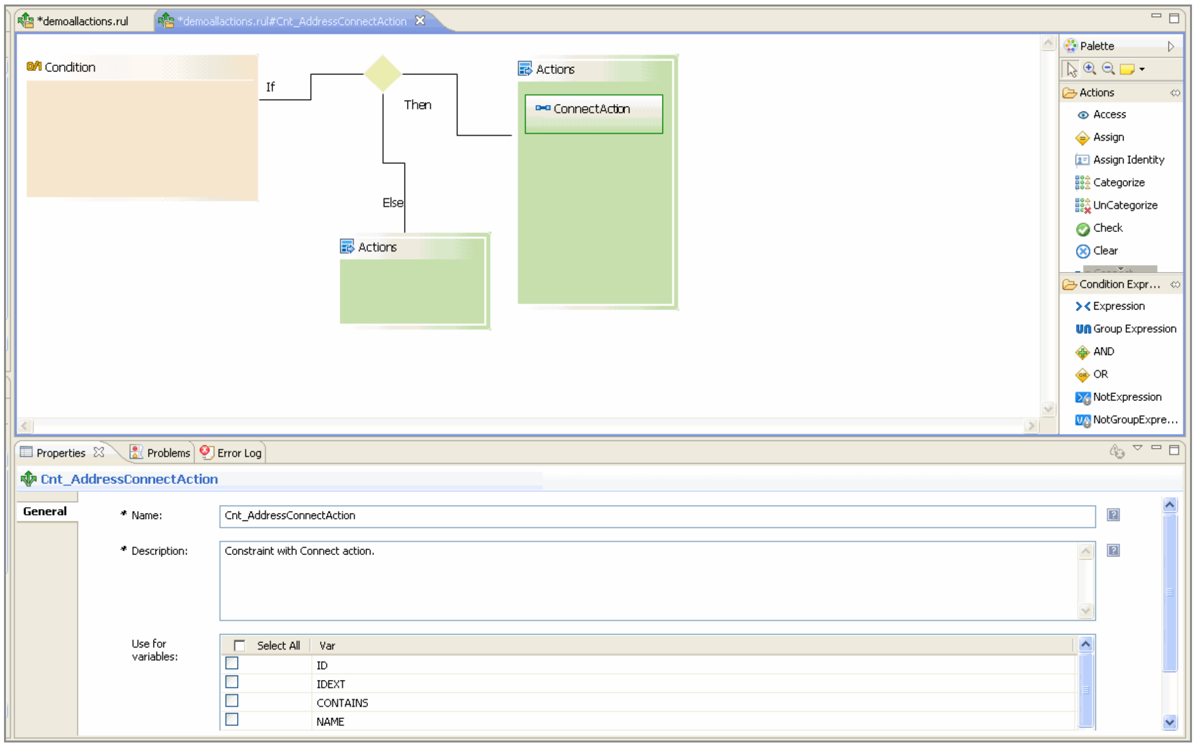
SoftlinkAction

Name:	SoftlinkAction
* Mode:	Record
Where:	varlinkCatalogCustomer.PRODUCTID = 'XXX'

Constraint with Connect action

In this example constraint with connect action is described.

Constraint with Connect action



Cnt_AddressConnectAction

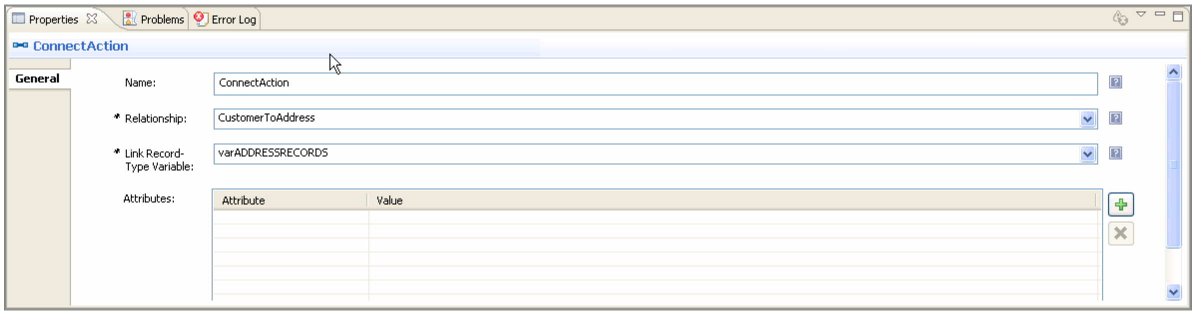
* Name: Cnt_AddressConnectAction

* Description: Constraint with Connect action.

Use for variables:

Select All	Var
☐	ID
☐	IDEXT
☐	CONTAINS
☐	NAME

Connect action General Properties



ConnectAction

Name: ConnectAction

* Relationship: CustomerToAddress

* Link Record-Type Variable: varADDRESSRECORDS

Attributes:

Attribute	Value

Context Variables

This appendix lists the context variables.

Context Variables

The followings special variables can be used in a rulebase:

- SESSION
- WORKITEM
- PREVIOUS_VERSION PREVIOUS_CONFIRMED_VERSION
- RECORD_ACTION
- RECORD_SUB_ACTION
- RECORD_IS_TOPMOST
- RECORD_IS_BOTTOMMOST
- PARENT
- CHILD
- WORKFLOW

SESSION

The following table lists session variables that can be used without explicit declaration.

Variable	Type	Value
`SESSION/DATE`	date	Current date.
`SESSION/USER_ID`	string	User ID of current user.

Variable	Type	Value
SESSION/USER_ROLES	array	Roles this user belongs to.
SESSION/ORGANIZATION_NAME	string	Organization Name.
SESSION/ORGANIZATION_TYPE	string	Organization type.
SESSION/ORGANIZATIONID	number	Organization identification number
SESSION/TIMESTAMP	timestamp	Date and Time
SESSION/ENTERPRISE_NAME	string	Enterprise Name.
SESSION/ENTERPRISE_INTERNAL_NAME	string	Enterprise Internal Name.
SESSION/LANGUAGE	string	User profile's language locale value.
SESSION/COUNTRY	string	User profile's country locale value.
SESSION/LANGSEL	string	Language locale selected from Login page.

The following example shows an access rule which restricts access to the attribute “SENSITIVE_ATTRIBUTE” only to Admin users. Notice that “in” has to be used, because a user can belong to more than 1 role, and USER_ROLES therefore returns an array of values.

```
<constraint>
       <name>HideSensitiveAttribute</name>
       <description>Only Admin Role can see
 Sensitive Attribute</description>
       <condition>
               <in>
                       <const type="string">Admin</const>
                       <var>SESSION/USER_ROLES</var>
               </in>
       </condition>
       <action>
```

```
                <access mode="modify">SENSITIVE_ATTRIBUTE</access>
        </action>
        <action>
                <access mode="hide">SENSITIVE_ATTRIBUTE</access>
        </action>
</constraint>
```

The following example shows usage of SESSION/LANGUAGE, SESSION/COUNTRY and SESSION/LANGSEL.

```
<constraint>
        <name>checkForRegion</name>
        <description>To Check if region is
 defined</description>
        <usefor>
                <var>REGION</var>
        </usefor>
        <condition>
                <defined>
                        <var>CHILD/ZIPCODE</var>
                </defined>
        </condition>
        <action>
                <check>
                        <explanation>REGION type
 should be defined.</explanation>
                        <defined>
                                <var>REGION</var>
                        </defined>
                </check>
        </action>
</constraint>
```

Note: After selecting language and country in User Accounts screen, the TIBCO MDM administrator has to logout and login as the user for whom the language and country selection is done. Otherwise these settings are persisted through the current session.

WORKITEM

Each step in the workflow has dependent criteria, and requires specific variables to be defined.

The following table lists variables, their types, and values.

Variable	Type	Value
`WORKITEM/ACTIVITY_NAME`	string	Name of current activity.
`WORKITEM/SEVERITY`	number	Workitem Severity.
`WORKITEM/STEP_SEVERITY`	number	Workitem Step Severity.
`WORKITEM/DOCTYPE`	string	Document Type that created workitem.
`WORKITEM/DOCSUBTYPE`	string	Document Sub-Type that created workitem.
`WORKITEM/ERRORS`	number	Number of errors in the record bundle.
`WORKITEM/WARNINGS`	number	Number of warnings in the record bundle.
`WORKITEM/REJECTIONS`	number	Number of rejections in the record bundle.
`WORKITEM/TRADING_PARTNER`	string	Trading Partner Name.
`WORKITEM/TRADING_PARTNER_TYPE`	string	Trading Partner Organization Type.
`WORKITEM/MARKETPLACE_NAME`	string	Marketplace name.
`WORKITEM/MASTER_CATALOG_NAME`	string	Name of the repository of the record being processed.
`WORKITEM/MASTER_CATALOG_VERSION`	Number	Repository Version.
`WORKITEM/INTENT`	String	Intent passed in to WorkItem activity.

Variable	Type	Value
WORKITEM/RECORD_COUNT	Number	Total number of records in the bundle.
WORKITEM/SUCCESS_COUNT	Number	Number of records with no errors.
Custom*	String	Any Parameter starting with “Custom” that is passed to Workitem Activity.

PREVIOUS_VERSION PREVIOUS_CONFIRMED_VERSION

You can access previous unconfirmed and confirmed versions of the records with two explicitly defined contexts.

The following table lists these contexts.

Context	Description
PREVIOUS_VERSION	Latest confirmed or unconfirmed version.
PREVIOUS_CONFIRMED_VERSION	Last confirmed version.

The following example shows how to access the weight attribute value of the last confirmed version:

```
<var>PREVIOUS_CONFIRMED_VERSION/WEIGHT</var>
```

In TIBCO MDM 8.2, you can also use this context variable to check record modification status.

```
<constraint>
    <name>TestChanged</name>
```

```
      <description>test changed</description>
      <condition>
         <changed><const type="string">PREVIOUS_CONFIRMED_VERSION</const>
   </changed>
      </condition>
      <action>
      <assign>
         <var>notes</var>
         <const type="number">Record has changed</const>
      </assign>
      </action>
  </constraint>
```

CONTEXT_RELATIONSHIP NAME

While adding a new record and executing the “newrecord” rulebase, the CONTEXT_ RELATIONSHIP variable contains the name of a relationship if the record is added by selecting the “add” action and then the relationship name on an existing record.

For example, if you are keeping a Vendor repository which has a relationship for multiple addresses, then you can automatically set the “RECORD_TYPE” to address if the user is adding a new address relationship.

```
<constraint>
        <name>Vendor_Record_Type</name>
        <condition>
                <undefined>
                        <var>CONTEXT_RELATIONSHIP/NAME</var>
                </undefined>
        </condition>
        <action>
                <assign>
                <var>RECORD_TYPE</var>
                <const type="string">VENDOR</const>
                </assign>
        </action>
</constraint>
<constraint>
        <name>Address_Record_Type</name>
        <condition>
                <eq>
                        <var>CONTEXT_RELATIONSHIP/NAME</var>
                        <const type="string">ADDRESS_REL</const>
                </eq>
        </condition>
        <action>
                <assign>
                        <var>RECORD_TYPE</var>
                        <const type="string">ADDRESS</const>
                </assign>
        </action>
</constraint>
```

The following is another example of assigning the RECORD_TYPE based on the Context Relationship.

```
<constraint>
        <name>PRODUCT</name>
        <condition>
                <undefined>
                        <var>CONTEXT_RELATIONSHIP/NAME</var>
                </undefined>
        </condition>
        <action>
                <assign>
                        <var>RECORD_TYPE</var>
                        <const type="string">PRODUCT</const>
                </assign>
        </action>
</constraint>
<constraint>
        <name>SKU</name>
        <condition>
                <eq>
                        <var>CONTEXT_RELATIONSHIP/NAME</var>
                        <const type="string">PRODUCT_TO_SKU_REL</const>
                </eq>
        </condition>
        <action>
                <assign>
                        <var>RECORD_TYPE</var>
                        <const type="string">SKU</const>
                </assign>
        </action>
</constraint>
```

RECORD_ACTION

Possible values:

RECORD_ACTION	Escaped Version
ADD	New record is being added.
EDIT	Existing record is being edited.
COPY	New Record is being copied from another record.
VIEW	Record is being viewed.
MASS_UPDATE	Executes a constraint only when the mass update is in process, that is, the same validation file can be used for record edit and mass update. Refer to mass_update.
RECORD_SEARCH	The record search is being executed. If a constraint is defined with this action, the same catalog validation file can be used for record search screen if the property com.tibco.cim.recordsearch.rulesfile is configured with catalog validation file name.

The following example restricts view access to the BASE_UNIT, ITEM_BRAND_NAME, and BRAND_OWNER_ID attributes, if the record is being viewed or edited:

```
<constraint>
       <name>ViewOnlyGeneralAttributesForViewAndChange</name>
       <description>View only attributes for
 View and Change</description>
       <condition>
       <or>
               <eq>
                       <var>RECORD_ACTION</var>
                       <const type="string">EDIT</const>
               </eq>
               <eq>
                       <var>RECORD_ACTION</var>
                       <const type="string">VIEW</const>
               </eq>
```

```
        </or>
        </condition>
        <action>
                <access mode="view">BASE_UNIT</access>
                <access mode="view">ITEM_BRAND_NAME</access>
                <access mode="view">BRAND_OWNER_ID</access>
        </action>
</constraint>
```

mass_update

This value allows you to execute a constraint only when the mass update is in process, that is, the same validation file can be used for record edit and mass update.

The RecordAction filter can be used for mass update as shown in the following example:

```
<constraint>
        <name>Check-int</name>
        <description>Check-int</description>
        <usefor>
                <var>TEST_INT</var>
        </usefor>
        <condition>
                <eq>
                <var>RECORD_ACTION</var>
                <const type="string">MASS_UPDATE</const>
                </eq>
        </condition>
        <action>
                <check>
                <explanation>TEST_INT should only be 1</explanation>
                <and>
                        <defined>
                                <var>TEST_INT</var>
                        </defined>
                                <eq>
                                <var/>
                                <const type="number">1</const>
                                </eq>
                </and>
                </check>
        </action>
</constraint>
```

record_search

The following is an example of a constraint to be used when RECORD_ACTION=RECORD_SEARCH is used:

```
<constraint>
       <name>TestSQLIN</name>
       <description>Test Get with SQL query
 using IN</description>
       <usefor>
           <var>Installments</var>
       </usefor>
       <condition>
         <eq>
           <var>RECORD_ACTION</var>
           <const type="string">RECORD_SEARCH</const>
    </eq>
       </condition>
         <action>
           <select>
       <table source="datasource">
            <literal>CUSTDS1</literal>
            <literal>INSTALLMENT</literal>
             <where>
                <sql>
                  <in>
                       <literal>INSTALLMENT</literal>
                       <const type="number">34000</const>
                       <const type="number">23000</const>
                       </in>
                     </sql>
              </where>
             </table>
           </select>
</action>
</constraint>
```

RECORD_SUB_ACTION

Currently, the only possible value that can be assigned to this variable is RESTORE.

When a record is restored, for all UI based rulebase validations for restored record, RECORD_SUB_ACTION is set to RESTORE.

Note: RECORD_SUB_ACTION is bound to RESTORE only for UI based validations and not during workflow processing.

RECORD_IS_TOPMOST

This variable allows you to check whether the record is the topmost in the hierarchy.

Example

```
<constraint>
<name>Test for parent</name>
<description>To Check if this is the parent</description>
<usefor>
<var>SHORTDESC</var>
</usefor>
<condition>
<and>
<eq>
<var>RECORD_TYPE</var>
<const type="string">CUST</const>
</eq>
<var>RECORD_IS_TOPMOST</var>
</and>
</condition>
<action>
<assign>
<var>SHORTDESC</var>
<const type="string">Parent Record</const>
</assign>
</action>
</constraint>
```

RECORD_IS_BOTTOMMOST

This variable allows to check whether the record is bottommost in the hierarchy.

Example

```
<constraint>
<name>Test for child/leaf node</name>
<description>To Check if this is the leaf node in a hierarchy of
records</description>
<usefor>
<var>SHORTDESC</var>
</usefor>
<condition>
<and>
<eq>
<var>RECORD_TYPE</var>
<const type="string">CUST</const>
</eq>
<var>RECORD_IS_BOTTOMMOST</var>
</and>
</condition>
<action>
<assign>
<var>SHORTDESC</var>
<const type="string">Child Record</const>
</assign>
</action>
</constraint>
```

PARENT

This variable allows to access attribute values of a parent record during relationship catalog rulebase execution.

Note: This context variable is only available in relationship catalog rulebase execution. The parent/child record is always determined in context of forward relationship.

Example

```
<constraint>
       <name>checkForAssetType</name>
       <description>To Check if asset type is defined</description>
```

```
        <usefor>
                <var>ASSET_TYPE</var>
        </usefor>
        <condition>
                <defined>
                        <var>PARENT/HAS_ASSETS</var>
                </defined>
        </condition>
        <action>
                <check>
                <explanation>Asset type should be defined.</explanation>
                <defined>
                        <var>ASSET_TYPE</var>
                </defined>
                </check>
        </action>
</constraint>
```

CHILD

This variable allows to access attribute values of child record during relationship catalog rulebase execution.

Note: This context variable is only available in relationship catalog rulebase execution. The parent/child record is always determined in context of forward relationship.

Example

```
<constraint>
  <name>checkForRegion</name>
  <description>To Check if region is defined</description>
  <usefor>
    <var>REGION</var>
  </usefor>
  <condition>
    <defined>
      <var>CHILD/ZIPCODE</var>
    </defined>
  </condition>
  <action>
<check>
      <explanation>REGION type should be defined.</explanation>
  <defined>
      <var>REGION</var>
  </defined>
</check>
 </action>
</constraint>
```

WORKFLOW

The following table lists session variables that can be used to access values of workflow.

Variable	Type	Value
ACTIVITY_NAME	String	Activity Name.

Tips and Tricks

This appendix lists some tips and tricks that you may find useful while working with the Rulebase Designer.

Tips

- While refreshing Rulebase Data View, make sure that the respective rulebase diagram is open and set as the active diagram.
- Tooltips on the Rulebase Data view contain some examples for rulebase functions. You can select an example and drag and drop it directly to the Expression Editor.
- While defining an expression in the condition compartment, make sure that the expression flow of connecting operators (“and”,”or”) is from left to right. The operators are provided with an arrow head on the right side.
- To arrange constraints and actions, select the constraint or action and drop it to the appropriate location in the respective container. To rearrange, select Main Canvas and press Ctrl + SHIFT + F.
- To resize a container, press **Auto Size** on the toolbar.
- Cut/Copy/Paste is allowed in variables, constraints and actions. Press Ctrl+C for copying and Ctrl+V for pasting.
- Save a diagram immediately after modification.
- When you modify an expression, always click **Save** in the Expression Editor followed by a diagram **Save** immediately.
- For Undo in the Expression Editor, press Ctrl + z.
- For Redo in the Expression Editor, Ctrl + y.
- In the Expression Editor, press Ctrl + Space to invoke the “Content Assist”.
- When there is a problem in an expression, you can refer to the actual problem by navigating through the Problems View.

- In the variable reference section, click on a variable to find its usage. A reference list is displayed. Double clicking on a reference in this list, navigates to the action in which this variable is used.

Classification Functions

This appendix lists the classification functions.

getClassificationScheme

Description	Parameters	Returns	Example

Retrieves the Classification Scheme with the name "classificationSchemeName" in repository "repositoryName"	repositoryName: Specify the repository name.
	
	classificationSchemeName: Specify the classifcation scheme name.
		the object of type classification scheme.	`<op func="getClassificationScheme"> <const type="string">mycatalogName</const> (any expression that evaluates to string)`
			`<const type="string">myschemeName</const> (any expression that evaluates to string) </op>`

isRecordCategorizedUnderScheme

Description	Parameters	Returns	Example
Checks whether present record is classified or categorized under the mentioned classification scheme	`classificationScheme`: specify the classification scheme object.	true or false.	`<op func="isRecordCategorizedUnderScheme"> <var>LINK_CSCHEME</var> or any expression that evaluates to classification scheme </op>`

getClassificationCodeByCode

Description	Parameters	Returns	Example
Retrieves the classification code object using provided parent code object and the code string	• `parentCode`: parent classification code object (link type classificationcode).	classification code object	`<op func="getClassificationCodeByCode"> <var>LINK_CCODE</var> or any expression that evaluates to classification code <const type="string">code</const> <!-- the code to retrieve -->             </op>`

Description	Parameters	Returns	Example
	• `Child classificationCode`: child classification code string.		

getClassificationCodeByName

Description	Parameters	Returns	Example
Retrieves the classification code object using provided parent code object and the code NAME string	• `parentCode`: parent classification code object (link type classificationcode). • `Child classificationCodeName`: child classification code NAME string.	classification code object	`<op func="getClassificationCodeByName"> <var>LINK_CCODE</var> or any expression that evaluates to classification code <cons type="string">codename</const> <!-- the code NAME to retrieve --> </op>`

getClassificationCodeForCodesInPath

Description	Parameters	Returns	Example
Constructs a classification code object from given path of codes in the scheme mentioned	• `classificationScheme`: the classification scheme (link type classification). • `treepathOfCodes`: the full treepath of codes from root code.	classification code object	`<op func="getClassificationCodeForCodesInPath"> <var>LINK_CSCHEME</var> or any expression that evaluates to classification scheme<consttype="string">rootcode/firstLevel/secondLevel/mycode</const> </op>`

getClassificationCodeForCodeNamesInPath

Description	Parameters	Returns	Example
Constructs a classification code object for given path of code names and in the scheme mentioned.	• `classificationScheme`: the classification scheme (link type classification). • `treepathOfCodeNames`: the full treepath of code names from root code.	classification code object	`<op func="getClassificationCodeForCodeNamesInPath"> <var>LINK_CSCHEME</var> or any expression that evaluates to classification scheme <const type="string">rootcode/firstLevelCodeName/secondLevelCodeName/mycodeName</const> </op>`

isRecordCategorizedUnderCodesPath

Description	Parameters	Returns	Example
Checks whether current record is categorized under the mentioned code path.	• `classificationScheme`: the classification scheme link object • `treepathOfCodes`:the full treepath of codes from root code	true or false	`<op func="isRecordCategorizedUnderCodesPath"><var>classificationScheme</var> <const type="string">rootcode/firstLevel/secondLevel/mycode</const> </op>`

isRecordCategorizedUnderCodeNamesPath

Description	Parameters	Returns	Example
Checks whether current record is categorized under the mentioned code name path.	• classificationScheme: the classification scheme link object • treepathOfCodeNames:the full treepath of code names from root code.	true or false	`<op func="isRecordCategorizedUnderCodeNamesPath"> <var>classificationScheme</var><const type="string">rootcode/firstLevelCodeName/secondLevelCodeName/mycodeCodeName</const> </op>`

isRecordCategorizedUnderMultipleCodePaths

Description	Parameters	Returns	Example

Checks whether current record is categorized under all provided array of code paths	classificationScheme: the classification scheme link object
	
	arrayOfTreepathOfCodes:an array of full treepath of codes from the root code
		true or false	`<op func="isRecordCategorizedUnderMultipleCodePaths"`
			`<var>classificationScheme</var>`
			`<cons`
			`type="string"rootcode/firstLevel/secondLevel/mycodeONE/</const>`
			`<const`
			`type="string"rootcode/firstLevel/secondLevel/mycodeTWO/</const>`
			`<const`
			`type="string"rootcode/firstLevel/secondLevel/mycode_N/</const>`
			`</op>`

isRecordCategorizedUnderMultipleCodeNamePaths

Description	Parameters	Returns	Example

Checks whether current record is categorized under all provided array of code names paths.	classificationScheme: the classification scheme link object
	
	arrayOfTreepathOfCodeNames
		true or false	`<op func="isRecordCategorizedUnderMultipleCodeNamesPaths">`
			`<var>classificationScheme</var>`
			`<const`
			`type="string">rootcode/firstLevel/secondLevel/mycodeNameONE</const>`
			`<const`
			`type="string">rootcode/firstLevel/secondLevel/mycodeNameTWO</const>`
			`<const`
			`type="string">rootcode/firstLevel/secondLevel/mycodeName_N</const>`
			`</op>`

getClassificationCodePathsForRecord

Description	Parameters	Returns	Example

Returns the array of all code paths under which current record is classified/
 	
categorized.
	None	array of all the treepaths of codes.
		
		["rootcode/cat1/../code1", "rootcode/cat2/../code2", "rootcode/cat3/../code3"]
			`<op`
			`func="getClassificationCodePathsForRecord">`
			`</op>`

getClassificationCodeNamePathsForRecord

Description	Parameters	Returns	Example
Returns the array of all code NAME paths under which current record is classified/categoriezed.	None	array of all the treepaths of code names ["rootcode/cat1/../codeNAME1", "rootcode/cat2/../codeNAME2", "rootcode/cat3/../codeNAME3"]	`<op func="getClassificationCodeNamePathsForRecord"></op>`

getClassificationCodesForRecord

Description	Parameters	Returns	Example
Returns the array of all classification code strings under which current record is classified/categorized.	None	array of code strings. ["code1", "code2", "code3", ...]	`<op func="getClassificationCodesForRecord"></op>`

getClassificationCodeNamesForRecord

Description	Parameters	Returns	Example
Returns the array of all classification code NAME strings under which current record is classified/categorized.	None	array of code names ["codeNAME1", "codeNAME2", "codeNAME3", ...]	`<op func="getClassificationCodeNamesForRecord"></op>`

getClassificationCodeLevel

Description	Parameters	Returns	Example
Retrieves the level for provided classification code object.	classificationCode:classification code object	level for this code e.g. 1,2,3 etc	`<op func="getClassificationCodeLevel"> <var>LINK_C_CODE</var> or any expression that evaluates to classification code </op>`

isSubCategoryOfCode

Description	Parameters	Returns	Example
Checks whether provided classification code is sub-category of mentioned parent code in the given classification scheme.	classificationScheme: classification scheme object parentCode: parent code string childCode: child code string	true or false	`<op func="isSubCategoryOfCode"> <var>LINK_CSCHEME</var> or any expression that evaluates to classification scheme <const type="string">parentCode</const> <const type="string">childCode</const> </op>`

isSubCategoryOfCodeName

Description	Parameters	Returns	Example
Checks whether provided classification code name is sub-category of mentioned parent code name in the given classification scheme.	classificationScheme:classification scheme object parentCodeName: parent code name string childCodeName: child code name string	true or false	`<op func="isSubCategoryOfCodeName"> <var>LINK_CSCHEME</var> or any expression that evaluates to classification scheme <const type="string">parentCodeName</const> <const type="string">childCodeName</const> </op>`

stringTreepathOfCodeToClassificationCode

Description	Parameters	Returns	Example
Constructs classification code objects for the each treepath provided in the array of code paths.	repositoryName: name of the repository classificationSchemeName: name of the classification scheme arrayOfTreepathOfCodes:an array of full treepath of codes from the root code	array of classification code objects for provided treepaths	`<op func="stringTreepathOfCodeToClassificationCode"> <const type="string">repositoryName</const> <const type="string">schemeName</const> <const type="string">rootcode/cat1/..../code1</const> <const type="string">rootcode/cat2/..../code2</const> <const type="string">rootcode/cat3/..../code3</const> </op>`

stringTreepathOfCodeNamesToClassificationCode

Description	Parameters	Returns	Example
Constructs classification code objects for the each treepath provided in the array of code name paths in parameter	repositoryName: name of the repository classificationSchemeName: name of the classification scheme arrayOfTreepathOfCodeNames: an array of full treepath of code names from the root code.	array of classification code objects for provided treepaths	`<op func="stringTreepathOfCodeNamesToClassificationCode"> <const type="string">repositoryName</const> <const type="string">schemeName</const> <const type="string">rootcode/cat1/..../codeName1</const> <const type="string">rootcode/cat2/..../codeName2</const> <const type="string">rootcode/cat3/..../codeName3</const> </op>`

isRecordCategorized

Description	Parameters	Returns	Example
Checks whether current record is categorized under provided code under mentioned scheme.	ClassificationSchemeName: name of the classification scheme ClassificationCode: Classification code link object	true or false	`<op func="isRecordCategorized"> <var>LINK_C_SCHEME</var> <var>LINK_C_CODE</var> </op>`

isRecordCategorizedUnderAll

Description	Parameters	Returns	Example
Checks whether current record is categorized under all the code provided in list under mentioned scheme.	ClassificationSchemeName: name of the classification scheme list of classifiication code link objects: An output of other classification custom function. flag: true or false	true or false If the flag is set true, then this method returns true ONLY IF the record is categorized under ALL codes in the given list. If the flag is set false, then this method returns false ONLY IF the record is NOT categorized under ANY codes in the given list of codes.	`<op func="isRecordCategorizedUnderAll"> <var>LINK_C_SCHEME</var> <op func="stringTreepathOfCodeNamesToClassificationCode"> <const type="string">repositoryName</const> <const type="string">schemeName</const> <const type="string">rootcode/cat1/..../codeName1</const> <const type="string">rootcode/cat2/..../codeName2</const> <const type="string">rootcode/cat3/..../codeName3</const> </op> <const type="boolean">true</const> </op>`

TIBCO Documentation and Support Services

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

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the Product Documentation website, mainly in HTML and PDF formats.

The Product Documentation website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The documentation for this product is available on the TIBCO® MDM Studio Documentation page.

How to Contact Support for TIBCO Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our product Support website.
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the product Support website. If you do not have a username, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

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

Legal and Third-Party Notices