

TIBCO® MDM Studio

Rulebase Designer User's Guide

Software Release 5.0
December 2015

Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

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

This document contains confidential information that is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO and Two-Second Advantage TIB, TIBCO Adapter, Predictive Business, Information Bus, TIBCO BusinessConnect, TIBCO ActiveMatrix BusinessWorks, TIBCO Enterprise Message Service, TIBCO MDM, TIBCO MDM Studio, TIBCO MDM Studio Process Designer, TIBCO MDM Studio Rulebase Designer, TIBCO MDM Studio Repository Designer are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Enterprise Java Beans (EJB), Java Platform Enterprise Edition (Java EE), Java 2 Platform Enterprise Edition (J2EE), and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle Corporation in the U.S. and other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

THIS SOFTWARE MAY BE AVAILABLE ON MULTIPLE OPERATING SYSTEMS. HOWEVER, NOT ALL OPERATING SYSTEM PLATFORMS FOR A SPECIFIC SOFTWARE VERSION ARE RELEASED AT THE SAME TIME. SEE THE README FILE FOR THE AVAILABILITY OF THIS SOFTWARE VERSION ON A SPECIFIC OPERATING SYSTEM PLATFORM.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

Copyright © 2007-2015 TIBCO Software Inc. ALL RIGHTS RESERVED.

TIBCO Software Inc. Confidential Information

Contents

Figures	11
TIBCO Documentation and Support Services	12
Overview	13
Rulebase Designer Overview	13
What you can do with the Rulebase Designer	13
Establish a Rulebase File	14
New Record File	14
Validation File	15
Search Control Rules File	15
Global Property Settings for Rules	15
LogFlag	15
Attribute Names Checking	15
Date Formats allowed in Rulebase	16
Getting Started	17
Starting MDM Rulebase Designer	17
Welcome Screen	17
Accessing Samples	17
Accessing TIBCO MDM Studio Help	18
Modeling Perspective	18
Project Explorer	19
Diagram Editor	19
Properties Tab	19
Rulebase Data View	20
Problems Tab	20
Palette	20
Main Palette	20
Expression Editor Palette	21
HTML Tooltips	22
Quick Search	23
Rulebase Navigation	24
Create a New Rulebase	25
Pre-Requisites for Creating a Rulebase	25
Creating a new Project to hold your Rulebase Model	25
Defining or Importing Repository Data	27
Creating a Rulebase Model	28
Rulebase Properties	30
Actions Allowed for Different Types of Rulebases	32
Adding Rulebase To a Folder	33

Importing Users and Roles	34
Variables	38
Types of Variables	38
User Defined Variables	38
Data-type variables	38
Link-type variables	38
Implicit Context Variables	38
Session Variables	39
Workitem Variables	40
Synch History variables	41
System Variables	44
Other Variables	45
Workflow Variables	46
Attribute History Variable	47
Attribute Quality Variables	48
Precedence Result Variable	48
Declaring Variables	49
Through the Palette	50
Through the Project Explorer	50
Editing Variables	50
Variable Properties	51
Properties for Link type variables	51
Properties for Data type variables	51
Constraints	52
Adding Constraints	52
Editing Constraints	52
Constraint Properties	52
Conditions	53
Working with Decision Tables	55
Create a Decision Table	55
Creating a Decision Table	55
Operators	57
Data Type - Operator Listings	58
Custom Operator	58
Decision Table Editor	59
Filtering and Sorting Rows	60
Cell Validation	61
Cell Skipping	62
Skipping a Cell	62
Decision Table Export	63

Exporting a Decision Table	63
Direct Deploying the Decision Table	66
Expressions	68
Creating Expressions	68
Expression Editor	68
Content Assist	69
Syntax Errors	69
Templates	69
Restrictions on SQL Expressions	70
Actions	71
Logic Implemented for Action	71
Access Action	71
Access Action Properties	72
Access Action Validation	72
ApplyPrecedence Action Properties	73
General tab properties	73
ApplyPrecedence Action Validation	74
Apply Precedence Action	74
ApplyPrecedence Action Properties	74
General tab properties	74
ApplyPrecedence Action Validation	75
Assign Action	75
Assign Action Properties	75
Advanced tab properties	76
Assign Action Validation	76
Assign Identity Action	76
Assign Identity Action Properties	76
Advanced tab properties	77
Assign Identity Action Validation	77
Categorize Action	77
Categorize Action Properties	77
Categorize Action Validation	78
Check Action	78
Check Action Properties	78
General tab properties	78
Advanced Tab Properties	79
Check Action Validation	79
Clear Action	79
Clear Action Properties	79
General tab properties	79

Advanced tab properties	79
Clear Action Validation	80
Connect Action	80
Connect Action Properties	80
Connect Action Validation	80
Disconnect Action	80
Disconnect Action Properties	81
Disconnect Action Validation	81
Include Action	81
Include Action Properties	81
Include Action Validation	81
Include Rulebase	82
Including a Rulebase in a Current Rulebase File	82
Propagate Inline Action	85
Propagate-Inline Action Properties	85
Propagate Rulebase	86
Propagate-Rulebase Action Properties	86
Propagate-Rulebase Validation	86
Select Action	86
Select Action Properties	87
Slice Action	89
Slice Action Properties	89
Softlink Action	90
Softlink Action Properties	90
UnCategorize Action	91
UnCategorize Action Properties	91
UnCategorize Action Validation	91
Severity Priority Refresh and Level	92
Severity	92
Priority	92
Refresh	92
Level	92
Rulebase Data View	93
Rulebase Data View	93
Navigating through the Rulebase Data View	93
Components	93
Domain Objects	94
Operators	95
Math Operators	95
Minus	95

Div	96
Mult	96
Percent	96
Plus	96
Relational Operators	96
Not Equal to	97
Scalar Matching - Not equal to	97
Less Than	98
Less than equal to	98
Equal to	98
Scalar Matching - Equal to	98
Greater Than	99
Greater Than Equal To	99
Functions	99
Comparison Functions	100
changed	100
defined	100
in	101
like	101
match	101
undefined	102
Math Functions	102
round	103
String Functions	103
concat	104
length	104
lpad	104
rpad	105
substring	105
trim	105
uppercase	106
Other Functions	106
checkdigit	106
count	106
distinct	106
duplicate	107
enum	107
filter	108
invokeJavaAPI	109
lookup	109

max	110
min	110
toMultivalue	110
nvl	110
sequence	111
strip	111
synchstatus	111
syncOperationAttribute	112
tableDatasource	112
tableSql	113
toDate	113
validate_checkdigit	113
Classification Function	113
Built-in Functions	114
Custom Functions	114
Creating a Custom Function	114
Input HashMap	115
Output HashMap	115
Custom Rulebase Class Example	116
Variables	116
Templates	116
Deployment	118
Deployment Overview	118
Creating a MDM Deployment Server	118
Deploying TIBCO MDM Studio where SSL is Enabled	121
Direct Deploying of Rulebases	121
Undeploying Rulebases	123
Import and Export Rulebases	124
Importing Rulebases	124
Exporting Rulebases	125
Rulebase Examples	127
Sample - 1	127
Assign Action Constant	127
Assign Action Conditional	128
Assign Action	129
Access Action	130
Check Action	132
Softlink Action	133
Connect Action	134
Disconnect Action	135

Include Action	137
Propagate rulebase and inline Action	139
Select Action enum	140
Select Action Tables	141
Slice Action	143
Sample - 2	143
Constraint with Access Check and Inline-Propagate (with Assign) actions	144
Constraint with Assign Clear Include and Connect actions	146
Constraint with Select Slice and Softlink actions	147
Constraint with Assign action having array assignment	148
Constraint with Softlink action	149
Constraint with Connect action	151
Context Variables	152
Context Variables	152
SESSION	152
WORKITEM	153
PREVIOUS_VERSION PREVIOUS_CONFIRMED_VERSION	154
CONTEXT_RELATIONSHIP_NAME	155
RECORD_ACTION	156
mass_update	157
record_search	157
RECORD_SUB_ACTION	158
RECORD_IS_TOPMOST	158
RECORD_IS_BOTTOMMOST	158
PARENT	159
CHILD	159
WORKFLOW	160
Tips and Tricks	161
Tips	161
Classification Functions	162
getClassificationScheme	162
isRecordCategorizedUnderScheme	162
getClassificationCodeByCode	163
getClassificationCodeByName	163
getClassificationCodeForCodesInPath	164
getClassificationCodeForCodeNamesInPath	164
isRecordCategorizedUnderCodesPath	165
isRecordCategorizedUnderCodeNamesPath	165
isRecordCategorizedUnderMultipleCodePaths	166
isRecordCategorizedUnderMultipleCodeNamePaths	167

getClassificationCodePathsForRecord	167
getClassificationCodeNamePathsForRecord	168
getClassificationCodesForRecord	168
getClassificationCodeNamesForRecord	168
getClassificationCodeLevel	169
isSubCategoryOfCode	169
isSubCategoryOfCodeName	169
stringTreepathOfCodeToClassificationCode	170
stringTreepathOfCodeNamesToClassificationCode	170
isRecordCategorized	171
isRecordCategorizedUnderAll	171

Figures

Rulebase Sample	127
Assign Action with General Properties	128
Assign Action with Conditional General Properties	129
Assign Action with array with General Properties	130
Access Action Modify with General Properties	131
Check Action with General Properties	133
Connect Action with General Properties	135
Disconnect Action with assigned records with General Properties	136
Include Action rule to clear attributes	138
Propagate Action with General Properties	140
Select Action with enum with General Properties	141
Select Action with table datasource table	142
Slice Action	143
Rulebase Sample 2	143
Constraint with Action, Check and Inline propagate action	145
Access Properties	145
Access Action Properties	145
Check Action Properties	146
Propagate InlineAction Properties	146
Assign Action Properties	146
Constraint with Assign, Clear, Include and Connect actions	147
Constraint with Select, Slice and Softlink actions	148
Constraint with assign action having array assignment	149
Constraint with Softlink Action	150
Constraint with Connect action	151

TIBCO Documentation and Support Services

Documentation for this and other TIBCO products is available on the TIBCO Documentation site:

<https://docs.tibco.com>

Documentation on the TIBCO Documentation site is updated more frequently than any documentation that might be included with the product. To ensure that you are accessing the latest available help topics, please visit <https://docs.tibco.com>.

Product-Specific Documentation

Documentation for TIBCO products is not bundled with the software. Instead, it is available on the TIBCO Documentation site. To directly access documentation for this product, double-click the following file:

TIBCO_HOME/release_notes/TIB_bstudio-mdm_5.0.0_docinfo.html

where *TIBCO_HOME* is the top-level directory in which TIBCO products are installed. On Windows, the default *TIBCO_HOME* is *C:\tibco*. On UNIX systems, the default *TIBCO_HOME* is */opt/tibco*.

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

- TIBCO MDM Studio Release Notes
- TIBCO MDM Studio Installation Guide
- TIBCO MDM Studio Process Designer Tutorial
- TIBCO MDM Studio Process Designer User's Guide
- TIBCO MDM Studio Repository Designer Tutorial
- TIBCO MDM Studio Repository Designer User's Guide
- TIBCO MDM Studio Rulebase Designer Tutorial
- TIBCO MDM Studio Rulebase Designer User's Guide
- TIBCO MDM Studio UI Builder Tutorial
- TIBCO MDM Studio UI Builder User's Guide

How to Contact TIBCO Support

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

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

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

How to Join TIBCOCommunity

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

<https://www.tibcommunity.com>

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 detailed information on the Rulebase files, refer to the section [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 needs to be applied. The action describes what the rule actually does and controls which attributes the rule is applicable to. For detailed information, refer to the section [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 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 MDM using network deployment on the 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



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's 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
```



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.



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
<code>rbb1*.xml</code>	First pass of rulebase. Computes propagations.
<code>rbb2*.xml</code>	Propagations.
<code>rbb3*.xml</code>	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
- ddmm/yyyy
- yyyy-mm-dd
- yyyy/mm/dd
- dd-mm-yyyy
- dd/mm/yyyy
- dd-mm-yy

Getting Started

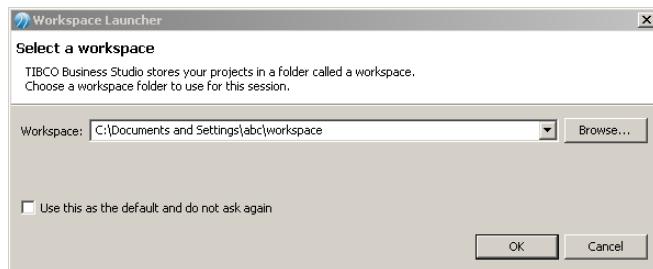
This chapter explains how to start the Rulebase Designer, what you will see at startup, information on accessing samples, tutorials, and help, and elements in the perspective.

Starting MDM Rulebase Designer

Procedure

1. After the installation completes, start the 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 opens up and you are ready to start using it.

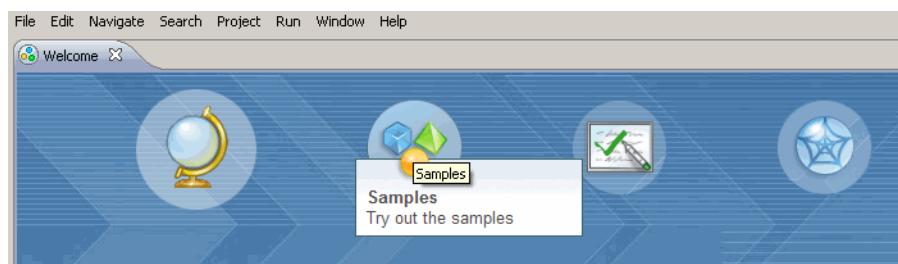
Welcome Screen

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.



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 MDM standard processes, process modeling tutorials, repository data models, rulebase model, custom import project, MDM Model templates, and Process java transitions.

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

Follow these steps to install the **Samples**.

Procedure

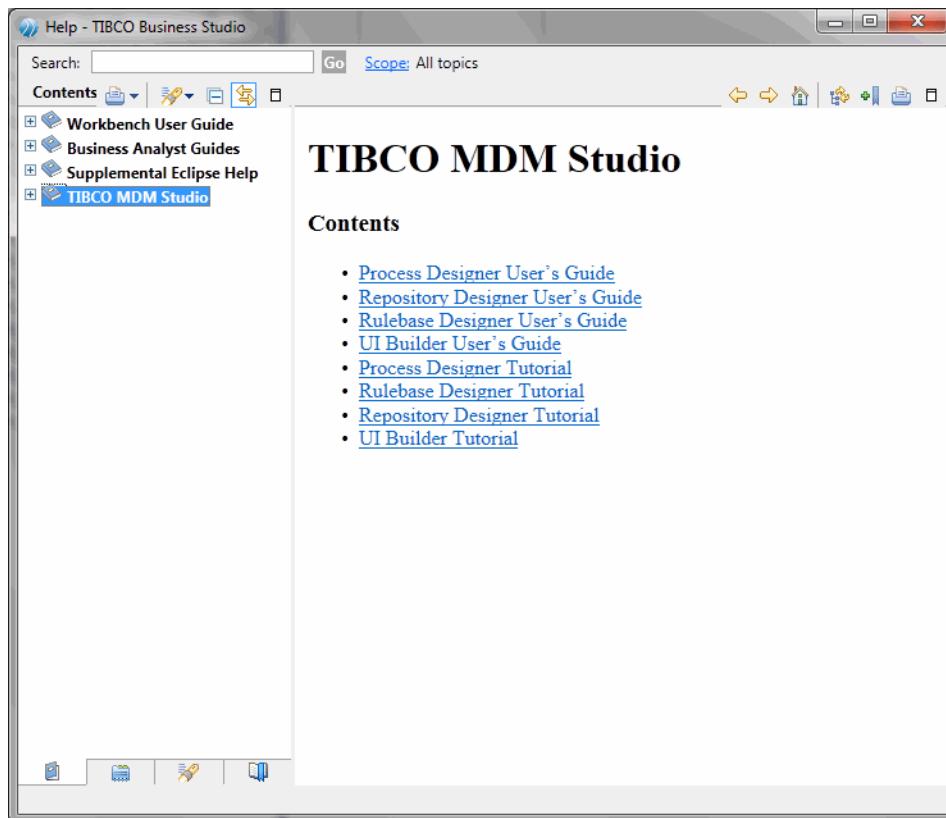
1. On the **File** menu, click **Import**. The import wizard is displayed.
2. From the **General** folder, select **Existing Studio Projects into Workspace**.
3. Click **Next**. The import wizard for selecting the directory path is displayed.
4. Click **Select archive file** option. Click **Browse** and select the sample project zip archives from `\<TIBCOHome>\studio-mdm\5.0\samples` folder.
5. Click **Finish**. The select project opens in the workspace.

Accessing TIBCO MDM Studio Help

Help can be accessed by

Click **Help > Help Contents**.

Expand the **TIBCO MDM Studio** in the **Help** window.



Modeling Perspective

The Modeling 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.

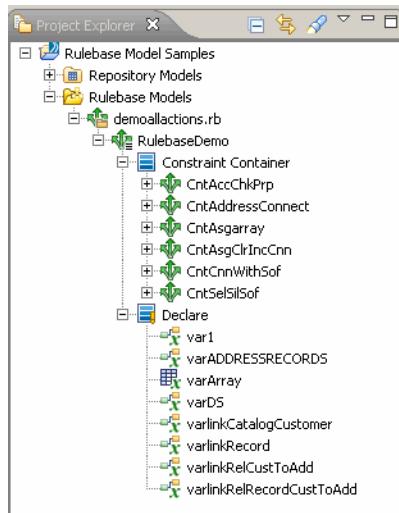


Diagram Editor

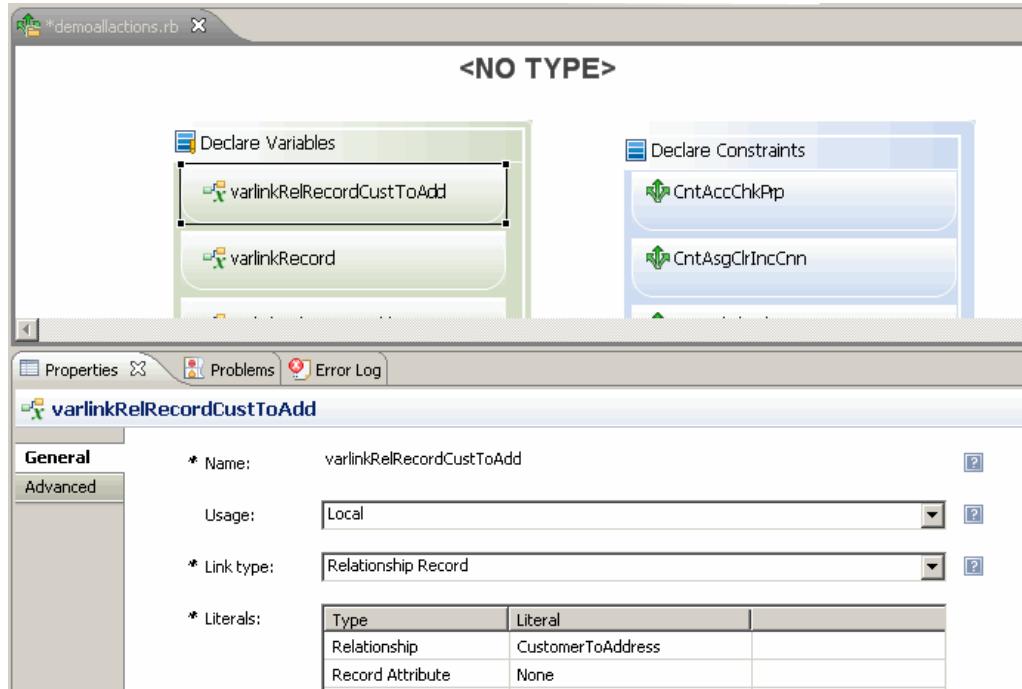
Enables graphical creation of rulebases. Consists of two sections:

- The diagram editing canvas (to design and edit a rulebase diagram).
- The [Palette](#) (for selection of artifacts such as variables and constraints).

Properties Tab

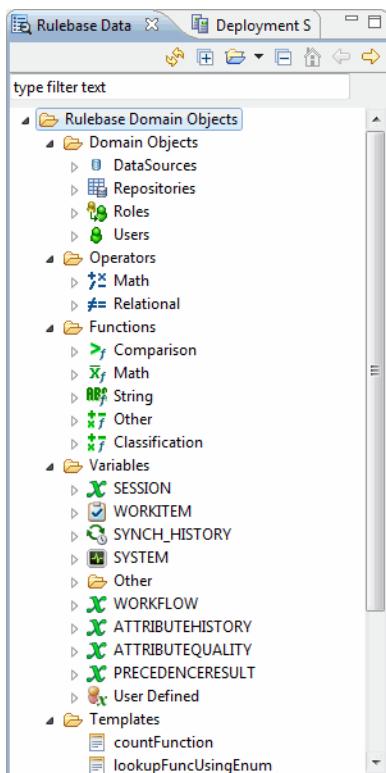
This view allows for editing of properties of rulebase components in the diagram.

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



Rulebase Data View

This view lists 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.

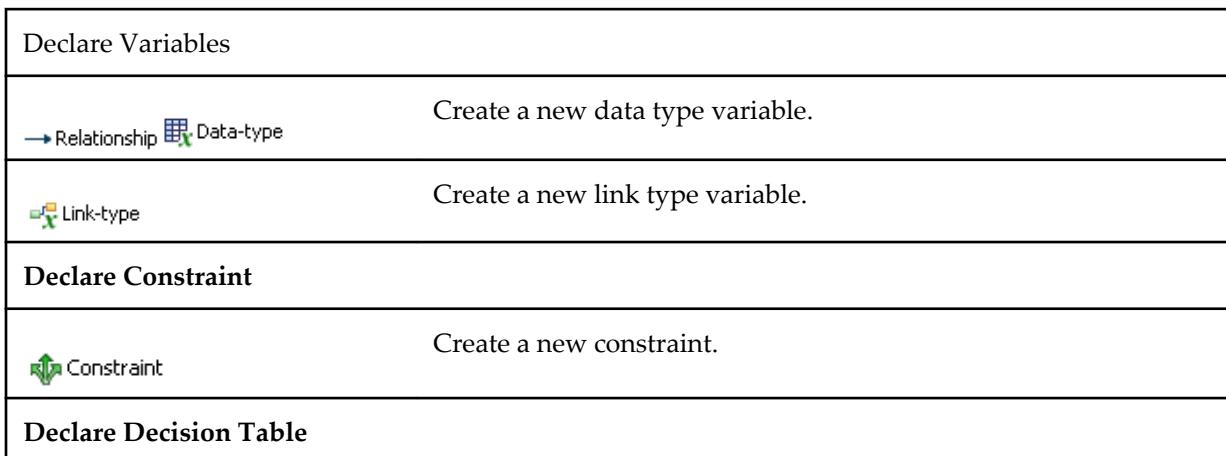
Palette

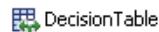
The Palette (to the right of the screen) contains different artifacts to help build a rulebase.

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.





Create a new Decision Table.

Expression Editor Palette

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

Actions	
	Create a new Access action.
	Create a new ApplyPrecedence action.
	Create a new Assign action.
	Create a new AssignIdentity action.
	Create a new Categorize action.
	Create a new Check action.
	Create a new Clear action.
	Create a new Connect action.
	Create a new Disconnect action.
	Create a new Include action.
	Create a new Propagate Inline action.
	Create a new Propagate Rulebase action.
	Create a new Select action.
	Create a new Slice action.
	Create a new Softlink action.
	Create a new UnCatgorize action.
Condition Expression	
	Create new Expression.

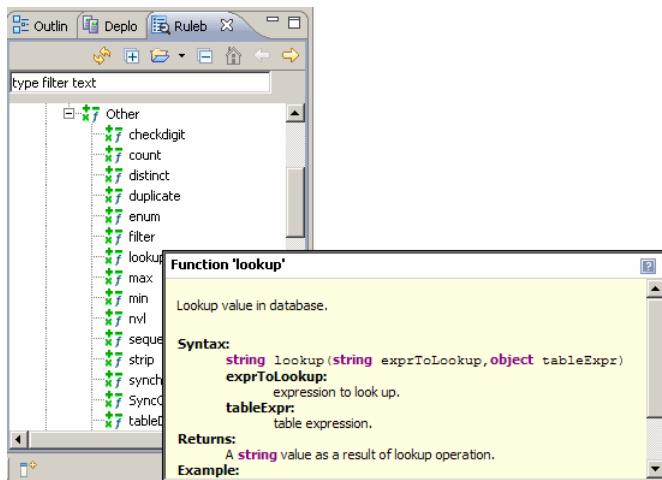
&& Group Expression	Create new Group Expression.
AND	Create new AND Operator.
OR	Create new OR Operator.
NotExpression	<p>Create new Not Expression.</p> <p>The Not Expression returns negation of an expression. If an expression value returns "true" then that expression written inside the "NotExpression" returns "false" as a result value. These kind of expressions are used when you need negation of computational expressions.</p>
NotGroupExpres...	<p>Create new Not Group Expression.</p> <p>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.</p>

HTML Tooltips

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

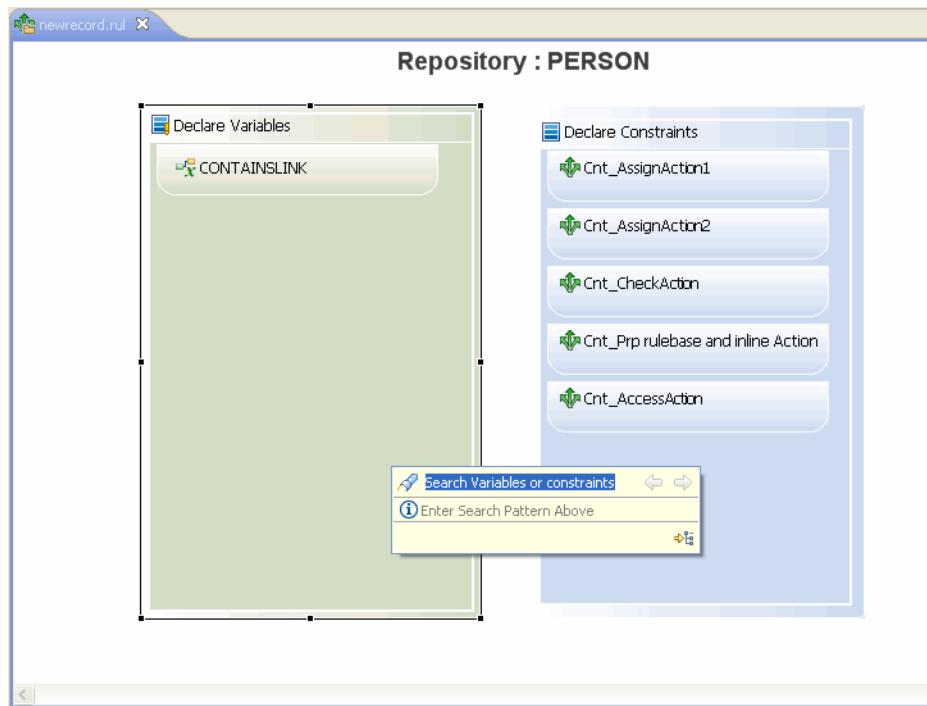
For example, in the **Rulebase Data View**, hover over functions to get a detailed description of that function including syntax and an example.

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



Quick Search

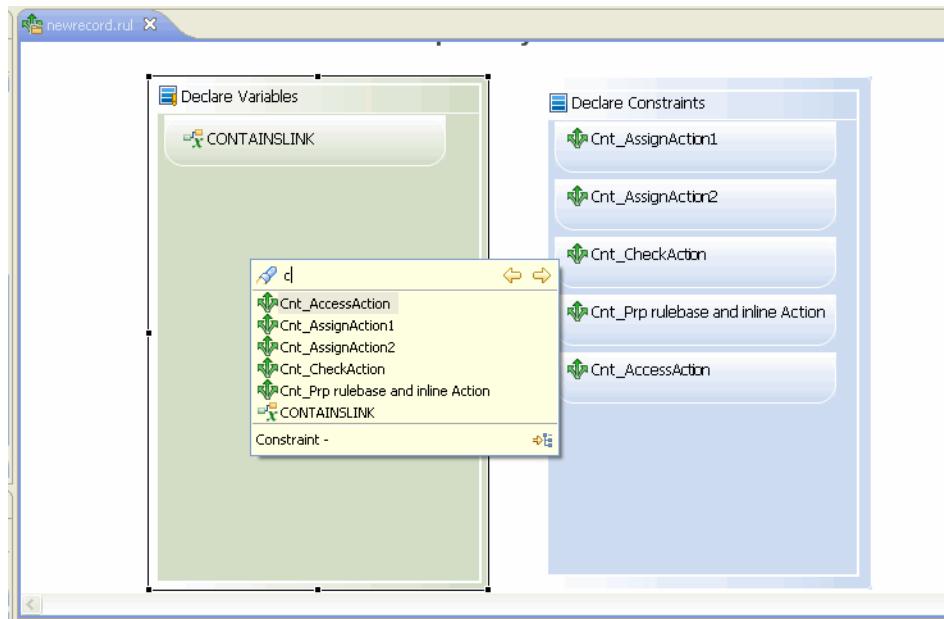
Using the type ahead search dialog you can search for the variables declared in the Declare section and the constraints declared in the Constraints section.



Press Ctrl + F or click



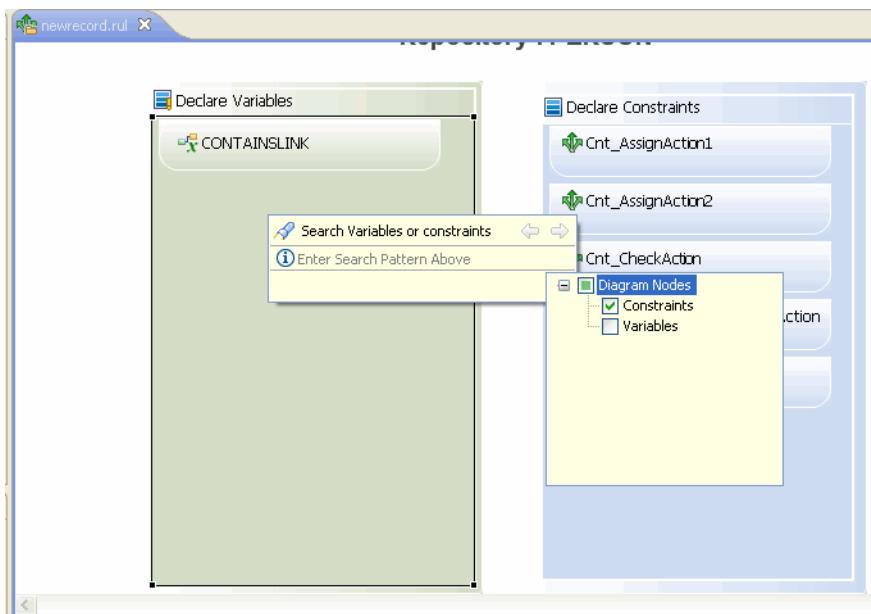
in the toolbar the type ahead search dialog opens. Enter the search criteria in the Search Variable or constraints field. The matching search result for the search criteria entered is shown.



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. Click **Select Search Categories**



. It displays the Search Categories.



Select the **Diagram Node** checkbox if you want to include both Variables and Constraints in the search result. select the **Constraints** checkbox if you want to include on constraints in search result. Similarly select the **Variables** checkbox if you want to include only Variables in the search results.

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.

Pre-Requisites for Creating a Rulebase

- **Step 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.

- **Step 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.



If you intend to create a Rulebase of type other than Initialization, Validation, or Search, you do not need to associate a repository.

- **Step 3 - [Creating a Rulebase Model](#)**

Finally, create a rulebase model.

- **Optional Steps - [Importing Users and Roles](#)**

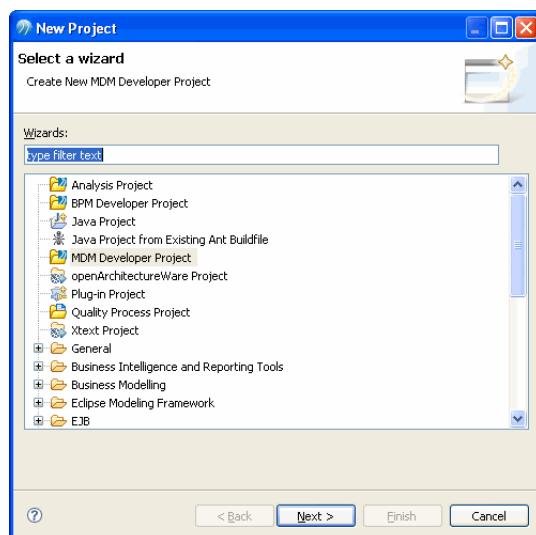
Optionally, import users and roles.

Creating a new Project to hold your Rulebase Model

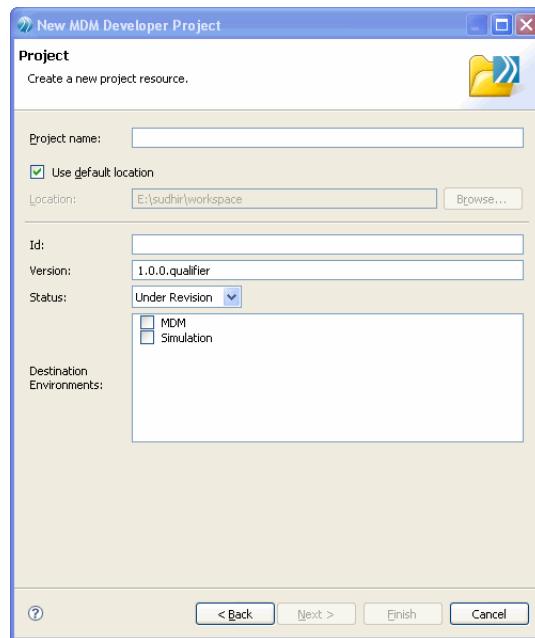
To create a rulebase model, you first need to create a Project to hold your model.

Procedure

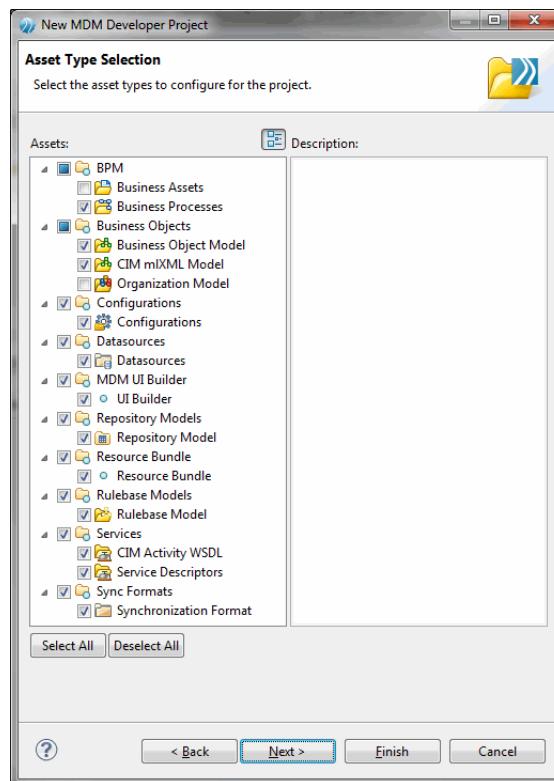
1. Go to **File > New > Project**. The Create New MDM Developer Project wizard is displayed.



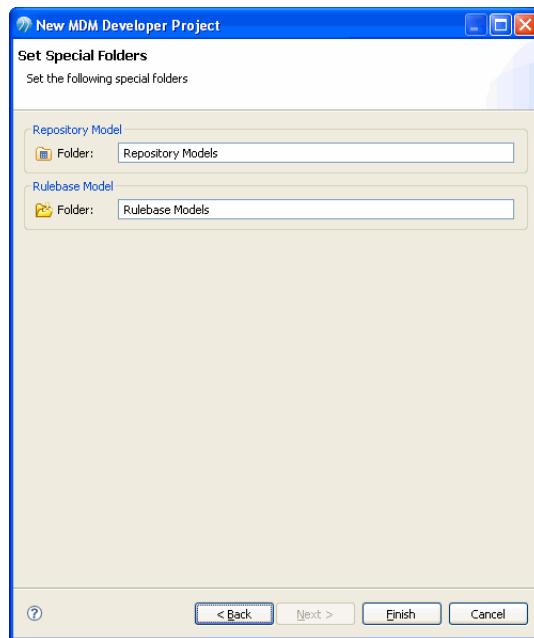
2. Select **MDM Developer Project** and click **Next**.
3. Provide a name for the Project. Clear the **Use default location** checkbox if you want to provide a different location for the project (by default, the current workspace). Select Destination Environment as **MDM**. Click **Next**



4. The Asset Type Selection dialog is displayed - select the **Repository Models** and **Rulebase Models** folders and click **Next**.



5. The folder for the **Repository Models** and **Rulebase Models** is displayed. Click **Finish**. See [Rulebase Properties](#) for more details.



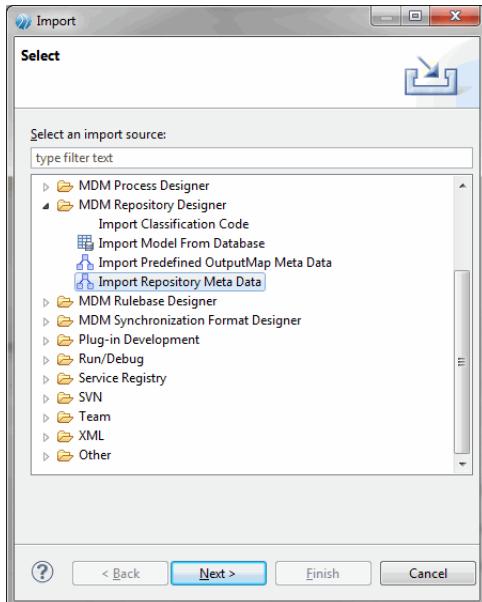
Defining or Importing Repository Data



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

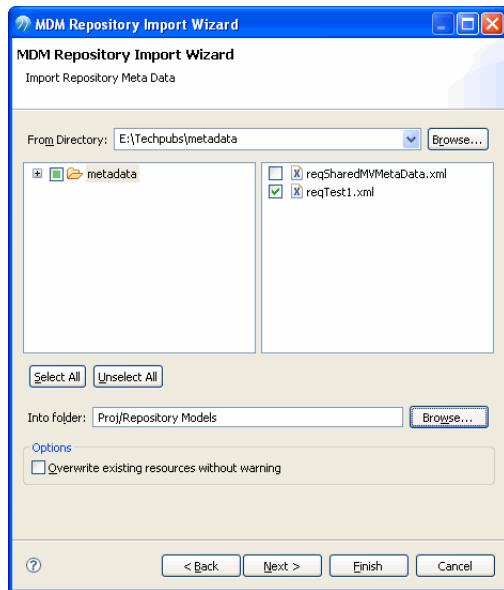
Procedure

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 meta data (.xml format). Click **Finish**.

Result



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

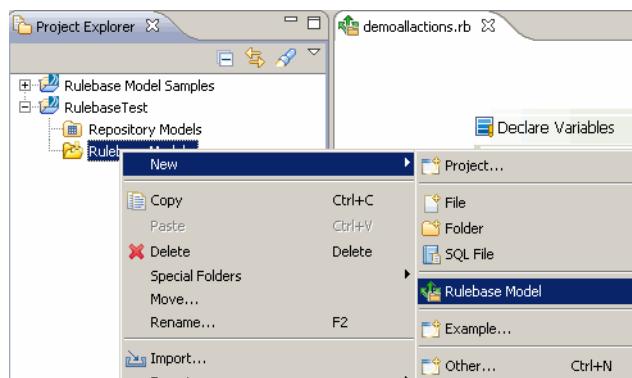
Creating a Rulebase Model



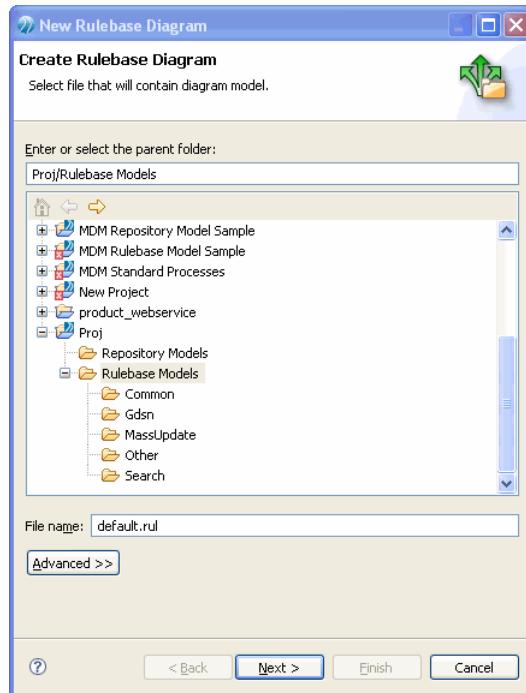
Ensure that you have defined or imported repository metadata before attempting to create a rulebase model.

Procedure

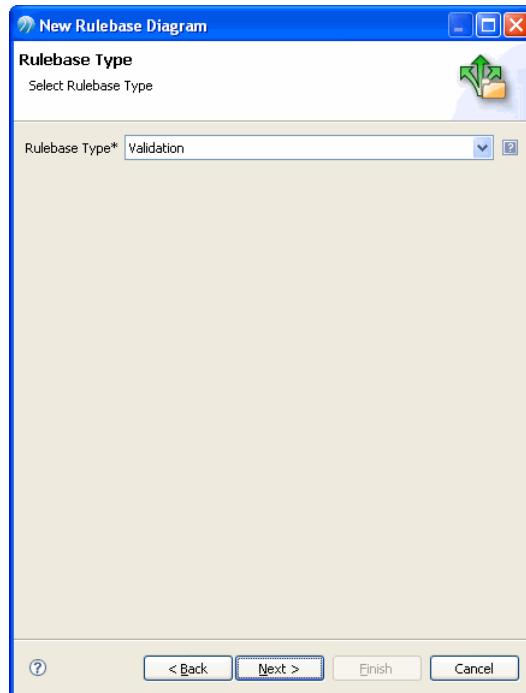
1. Right click the Rulebase Models folder in the Project Explorer and select New Rulebase Model.



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.



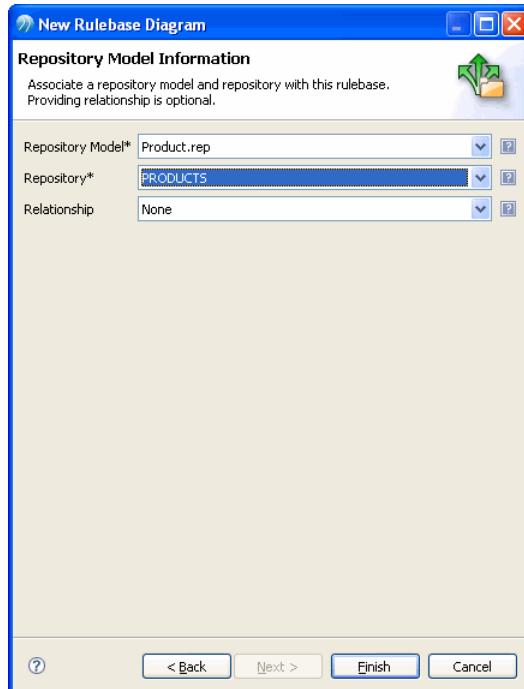
4. Click **Next**.
5. Associate repository data with your rulebase.
 - Select the Repository Model .rep file. This is mandatory field.
 - Select the Repository to associate. This is mandatory field.
 - Select the Relationship to associate.



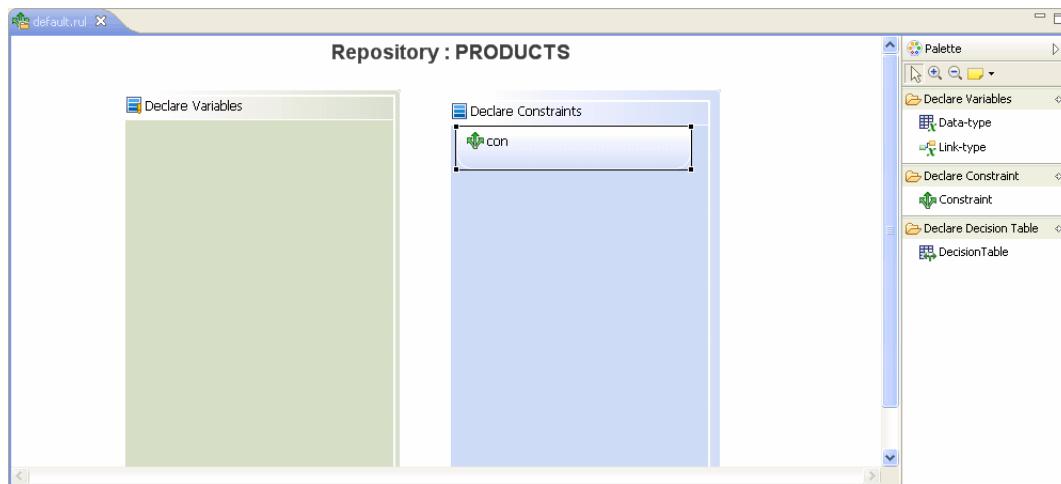
To associate a relationship, select the relationship's source (and not target) repository from the **Repository** drop-down; the **Relationship** drop-down will then get populated with available relationships that you can select to associate.

6. Click **Finish**.

Result



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](#).

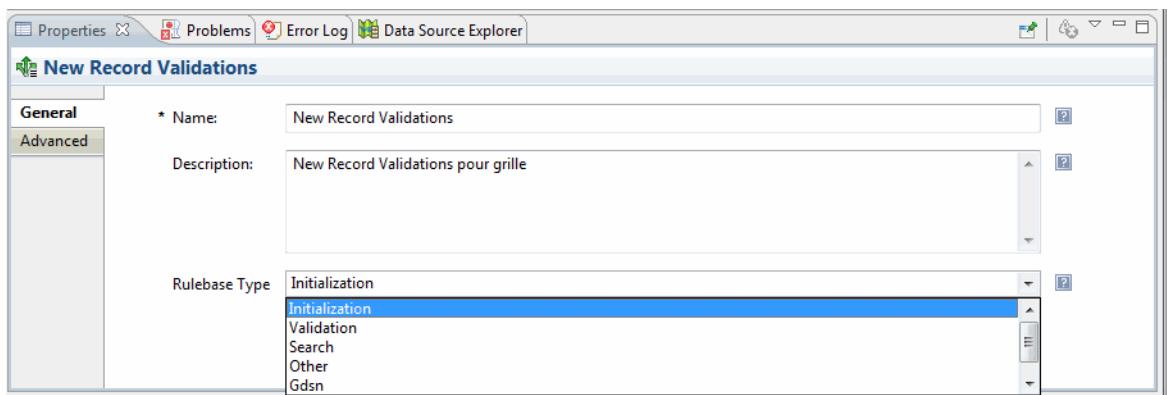


Rulebase Properties

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

General Properties

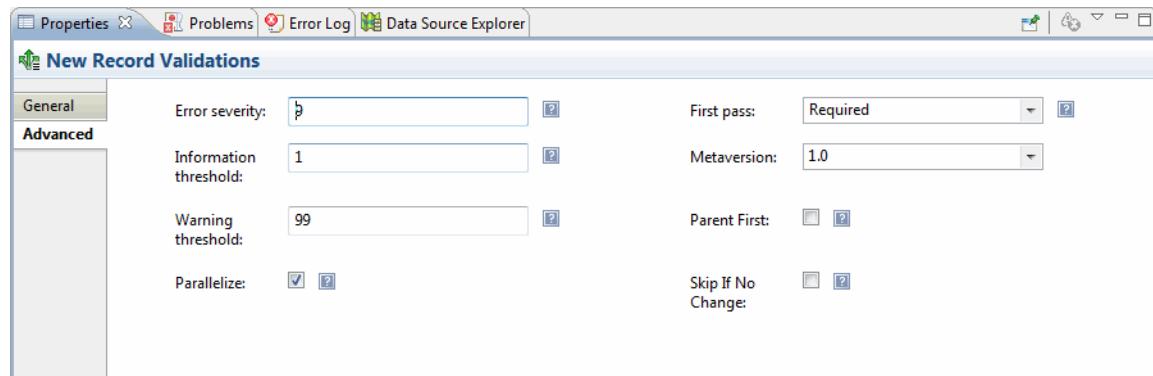
This displays the Rulebase Name, Description, and Type.



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).



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	<filename_gdsn>.xml
MassUpdate	Assign Access Select Include Clear	Advanced Mass Update	<filename>.xml

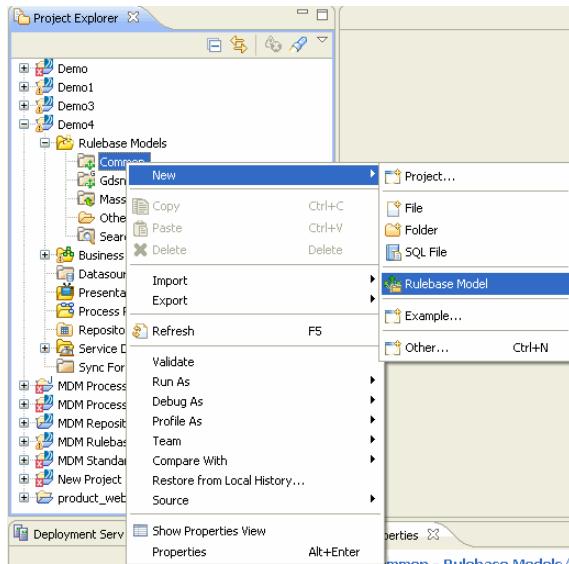
- The Initialization, Validation, Search, and Other type Rulebases are saved in \common_dir \- The GDSN and Mass update type Rulebase are saved in \common_dir\<Enterprise name>\rulebase\
- The Relationship type Rulebase is saved in \common_dir\<Enterprise name>\catalog\master\<Relationship table ID>\

Adding Rulebase To a Folder

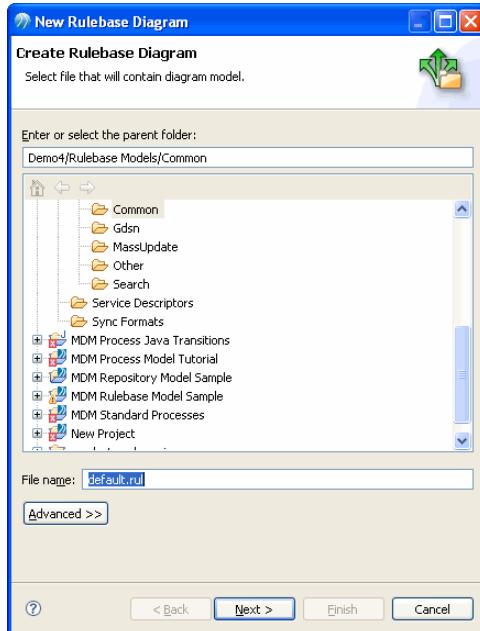
To add a new rulebase to a particular folder.

Procedure

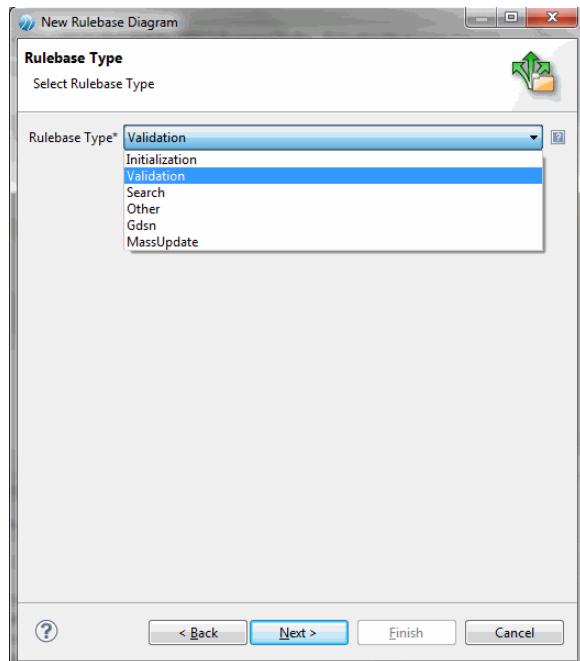
1. Select the folder in which you want to add the rulebase. Right-click on the folder (For example, Common) and select **New** and click **Rulebase Model**.



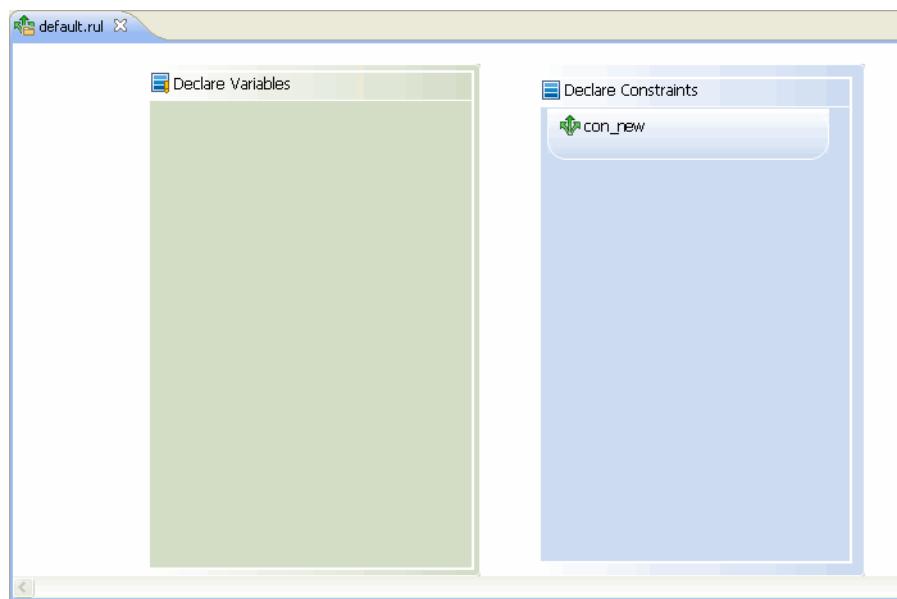
2. The Create Rulebase Diagram dialog is displayed.



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.

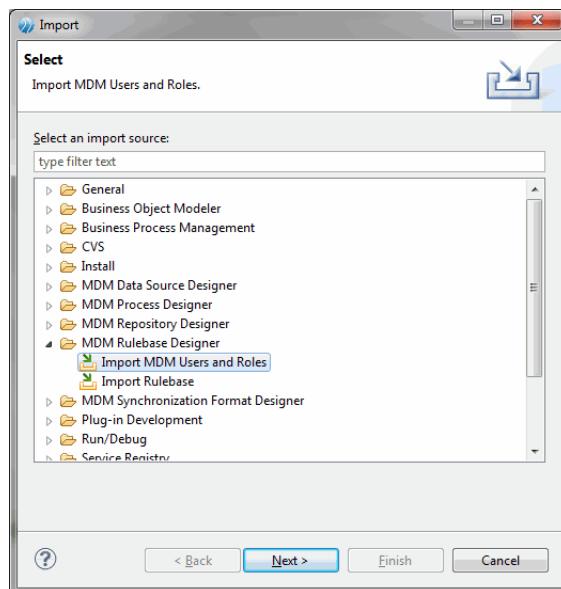


Importing Users and Roles

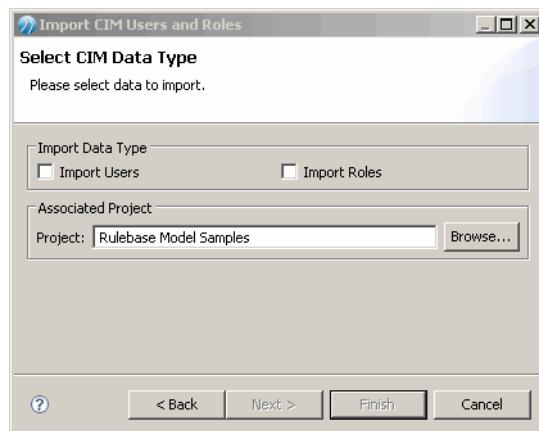
Importing of Users and Roles is an online activity.

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**.

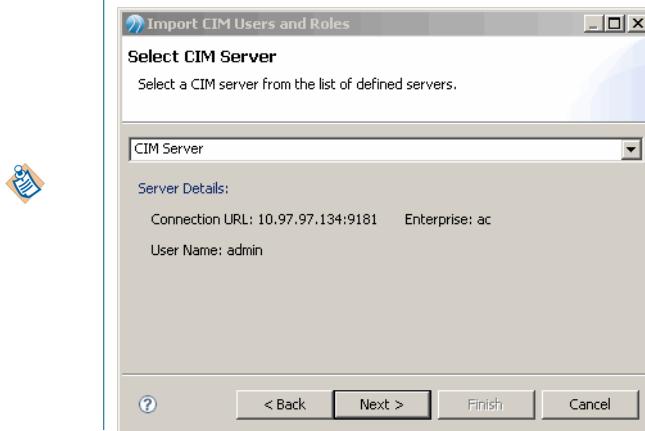


2. Import users, roles or both by selecting the related checkboxes. Browse to select the associated Project. Click **Next**.



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

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



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**.

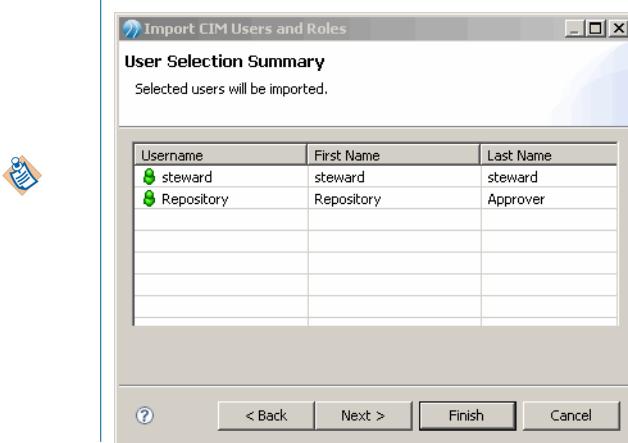


5. If you left the previous screen blank, all applicable users will be displayed here. If you provided some filter criteria in the previous screen, users that match that criteria will be displayed. Select the checkboxes of users to import and click **Next**.



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

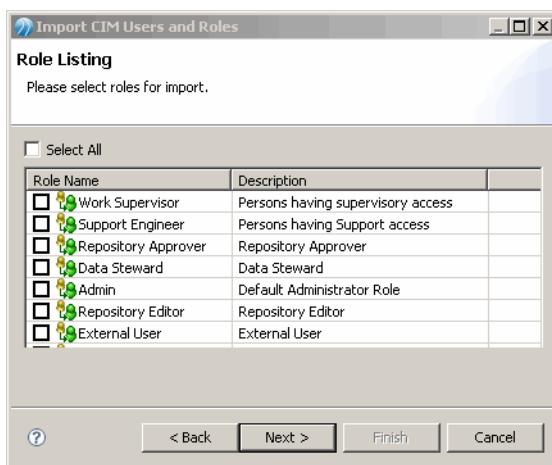
If you selected to import both roles and users but do not want to import roles at this point, click **Finish** instead of **Next**.



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

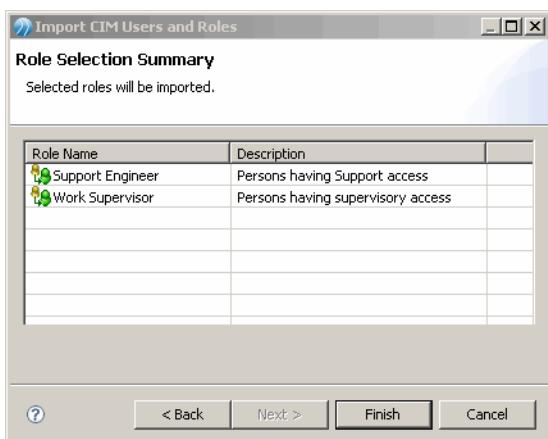


8. A list of applicable roles are displayed for selection; select the appropriate checkboxes and click **Next**.



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

Result



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

Variables

(Short Description to be added).

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).

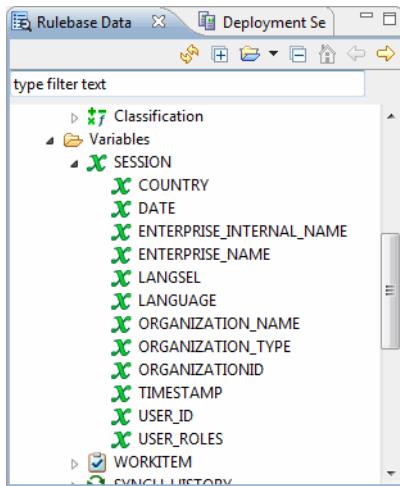
Most of these variables are available in specific contexts, for example, workitem variables will be available if your rulebase is executing in a workflow.

- [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

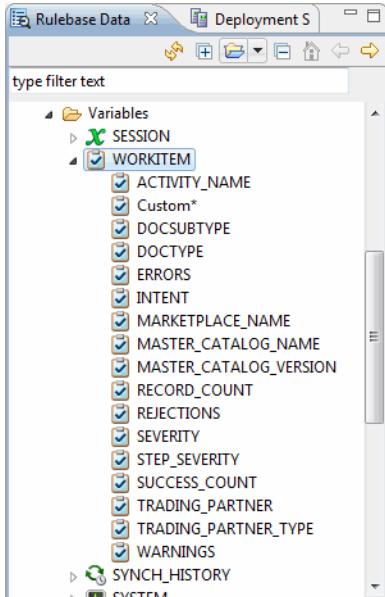
Variable	Type	Value
COUNTRY	String	Users Profile's Country locale value.
DATE	Date	Current date.
ENTERPRISE_INTERNAL_NAME	String	Enterprise Internal Name.
ENTERPRISE_NAME	String	Enterprise Name.
LANGSEL	String	Language locale selected from login page.
LANGUAGE	String	Users profile's Language locale value.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

Variable	Type	Value
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

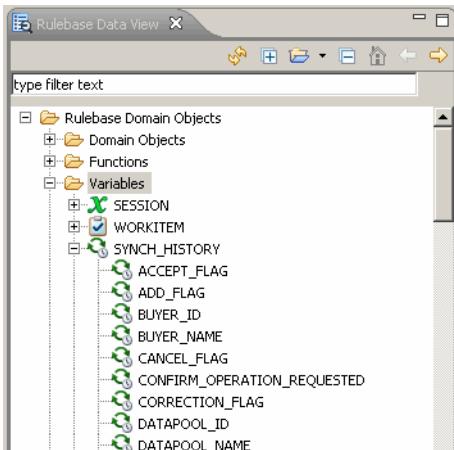
Variable	Type	Value
ACTIVITY_NAME	String	Current activity name.
SEVERITY	Number	Workitem severity.
STEP_SEVERITY	String	Step severity.
DOCTYPE	String	Document Type that created workitem.
DOCSUBTYPE	String	Document Sub-Type that created workitem.
ERRORS	Number	Number of errors in record bundle.
WARNINGS	Number	Number of warnings in record bundle.
REJECTIONS	Number	Number of rejections in record bundle.
TRADING_PARTNER	String	Trading Partner Name.

Variable	Type	Value
TRADING_PARTNER_TYP E	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_VERS ION	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

Variable	Description
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
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
IS_DISCONTINUE_REQUESTED	Is_Discontinue_Requested

Variable	Description
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
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

Variable	Description
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

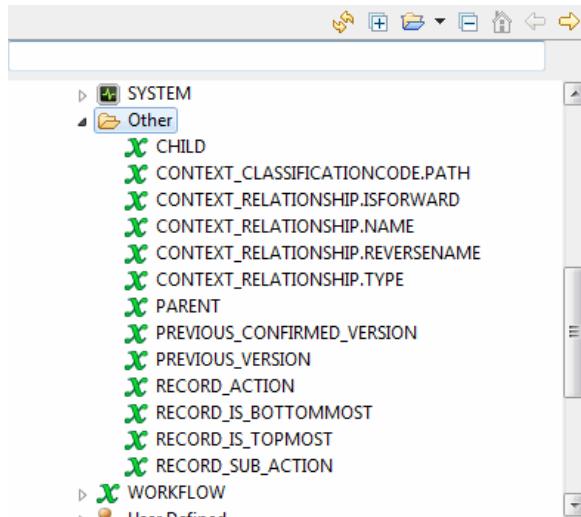
System Variables

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.

Variable	Description
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



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

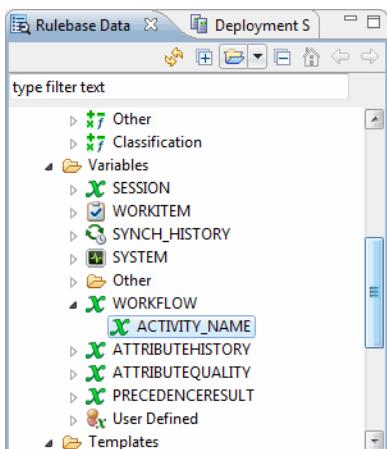
This variable allows you to check whether the record is the topmost in the hierarchy.

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.

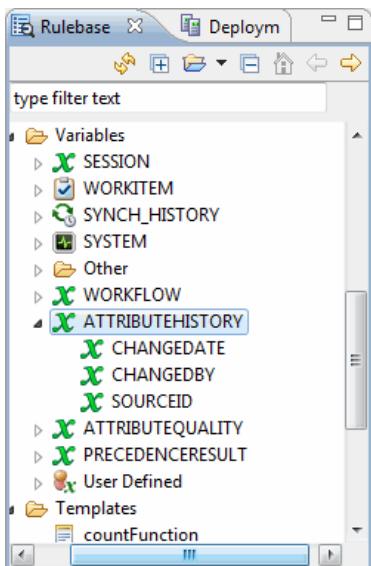


Workflow Variable

Variable	Type	Value
ACTIVITY_NAME	String	Activity Name.

Attribute History Variable

Attributes history variable is used to access the attribute history.

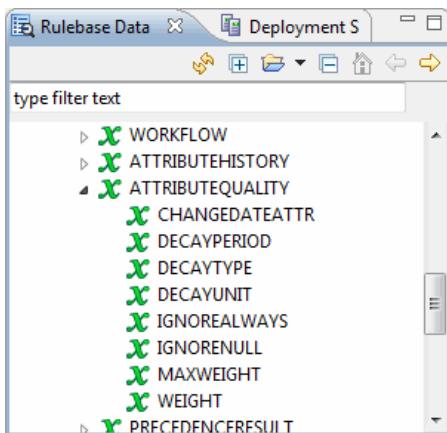


Attribute History Variable

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.

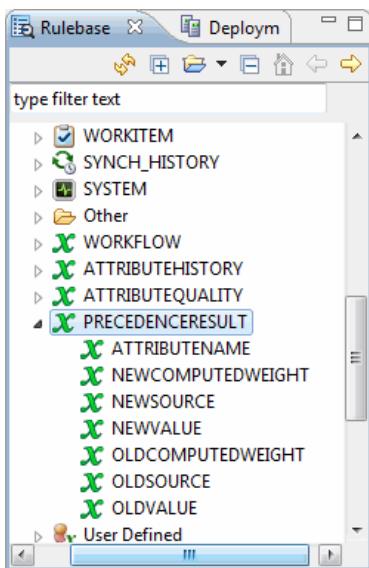


Attribute Quality Variable

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.



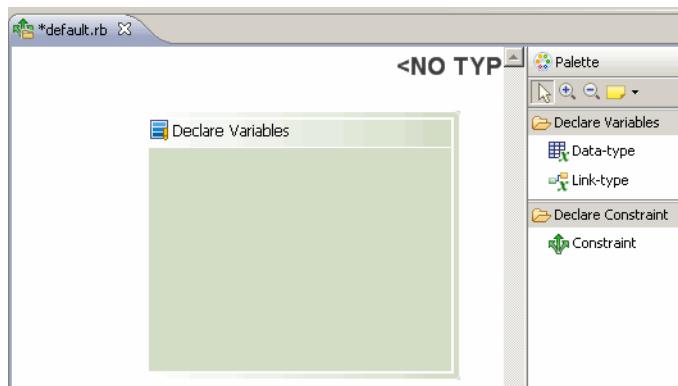
Precedence Result Variable

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.
OLDDVALUE	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.

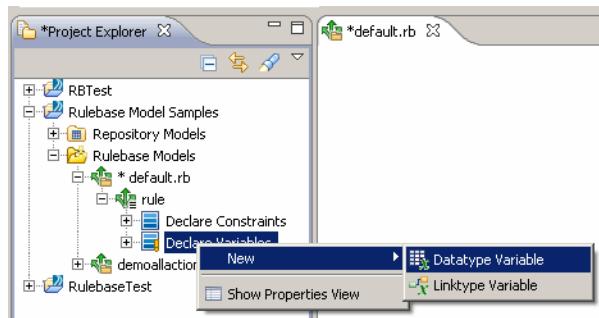


Through the Palette

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.

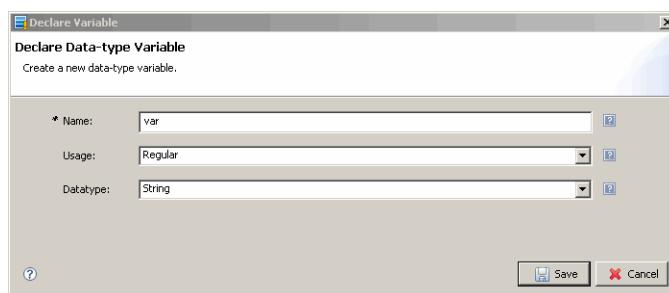
Through the Project Explorer

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](#)).



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**.

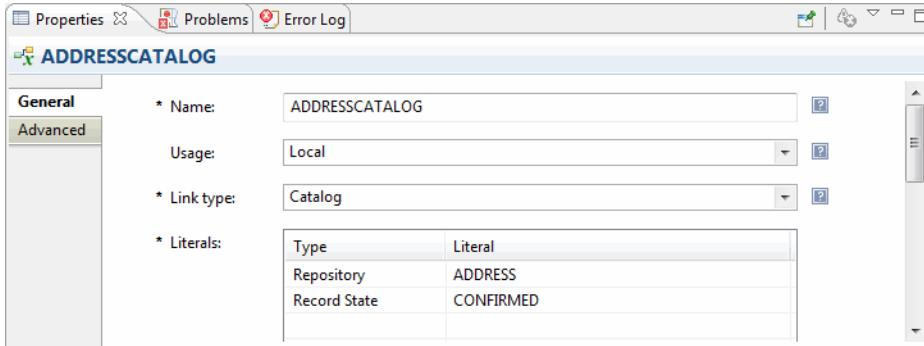
- **Reorder variables:** To reorder a variable, select a variable and drag to reorder its position within the variable declaration compartment.
- **Delete variables:** To delete a variable, select it and press **Delete** on the keyboard. Or select a variable, right click it and select **Delete from Model**.

Variable Properties

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

Properties for Link type variables

The following details are displayed in the **General** tab 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.
- **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.

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 ,etc)
- **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

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.



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

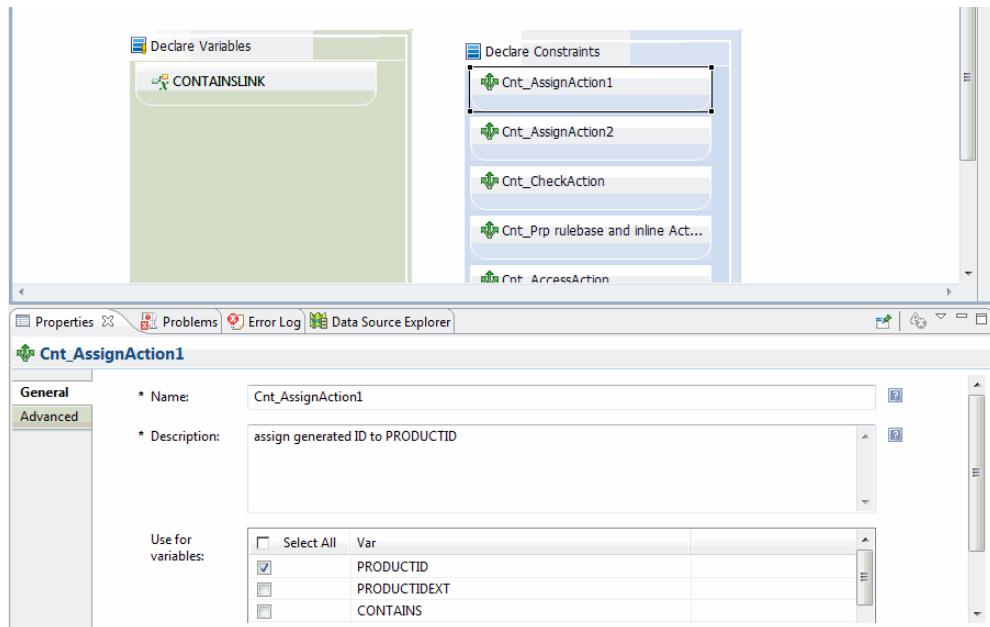
- **Delete constraints:** To delete a constraint, select it and press **Delete** on the keyboard. Or select a constraint, right click it and select **Delete from Model**.

Constraint Properties

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

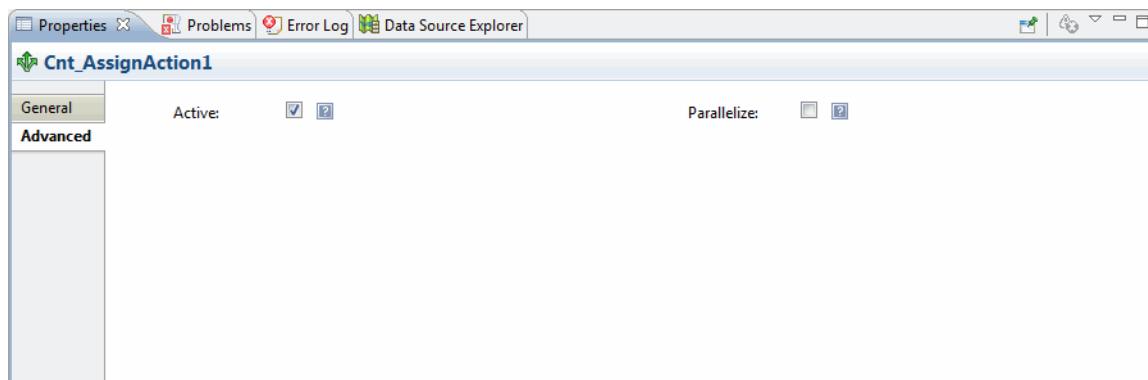
The following properties are displayed for constraints in the **Properties** Pane, General tab:

- **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.



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

- **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.



Double click a constraint in the main rulebase diagram to open up the Constraint diagram which lists Conditions and Actions. For details see, [aExpressions](#) 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:

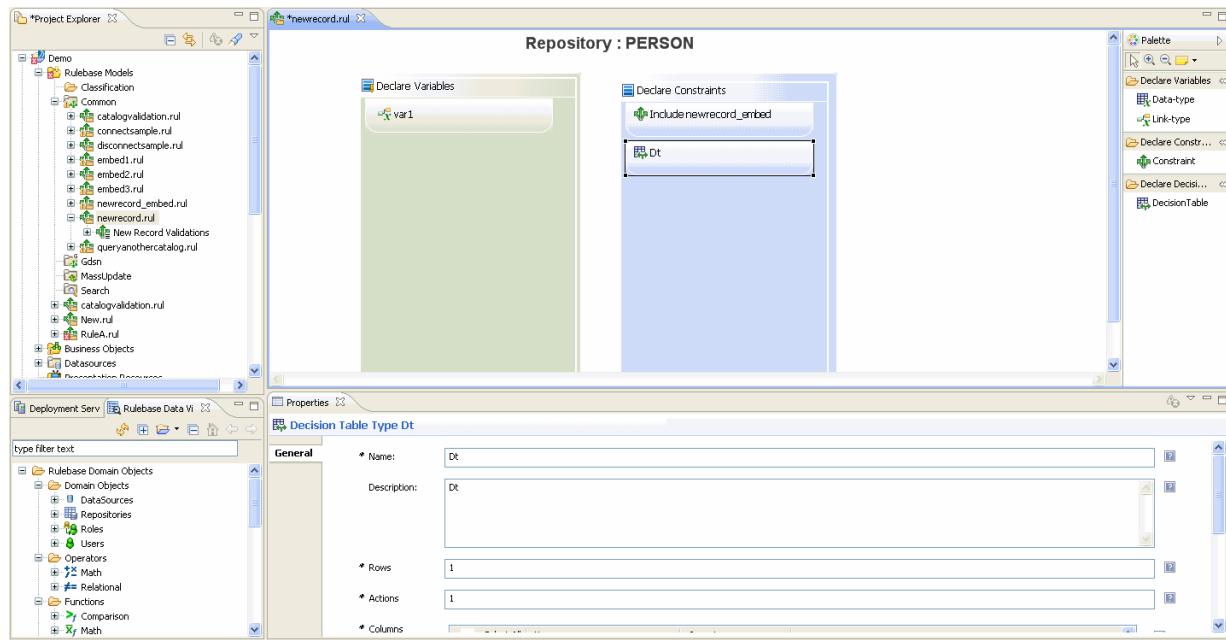
- **Name:** While creating the decision table a default name Dt, Dt1, Dt2, and so on is automatically populated. When a decision table is created for the first time the first instance in the constraint container has the name as Dt, the second instance has Dt1, the third instance has Dt2 and so on. The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

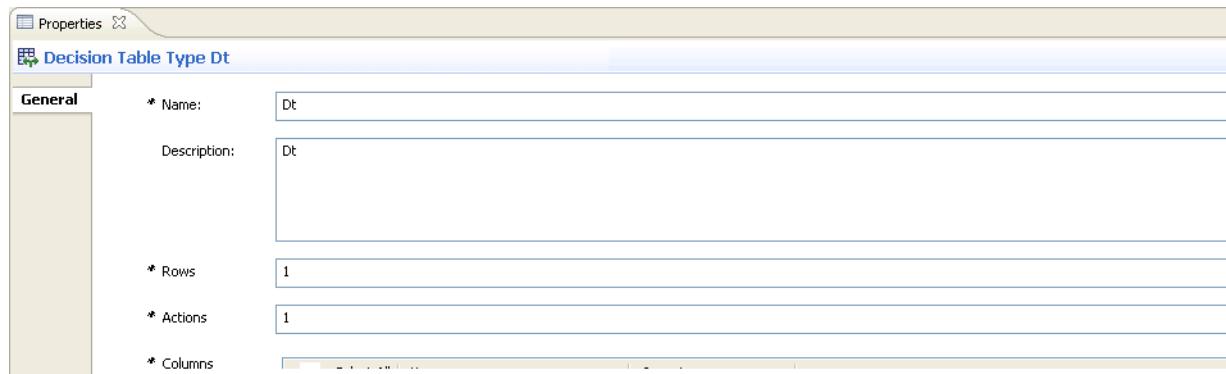
Procedure

1. Select the  **DecisionTable**

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



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



4. 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.
5. 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.
6. 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.
7. 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.
8. 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 divide 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.



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](#).

9. 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.



10. 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,

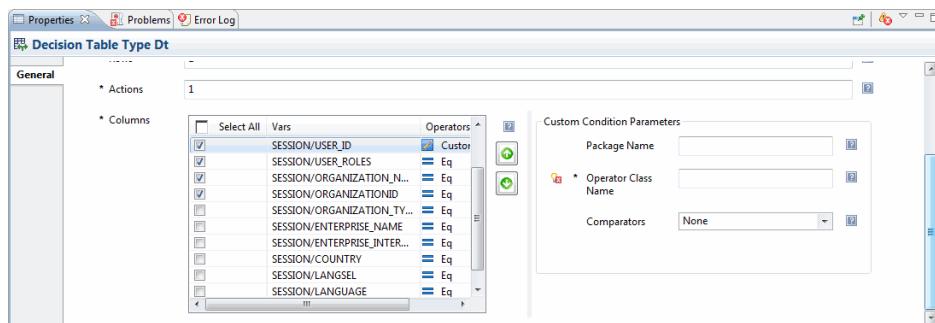
For multi-value attribute, following operators are supported

- 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.



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 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
1	John	age = 36	DATEOFBIRTH = 03-09-1975	MARRIED = <input checked="" type="checkbox"/> Action1 <input type="checkbox"/> Age

Actions

Insert Row Above
Insert Row Below
Duplicate
Move Row Above
Move Row Below
Delete Row

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:

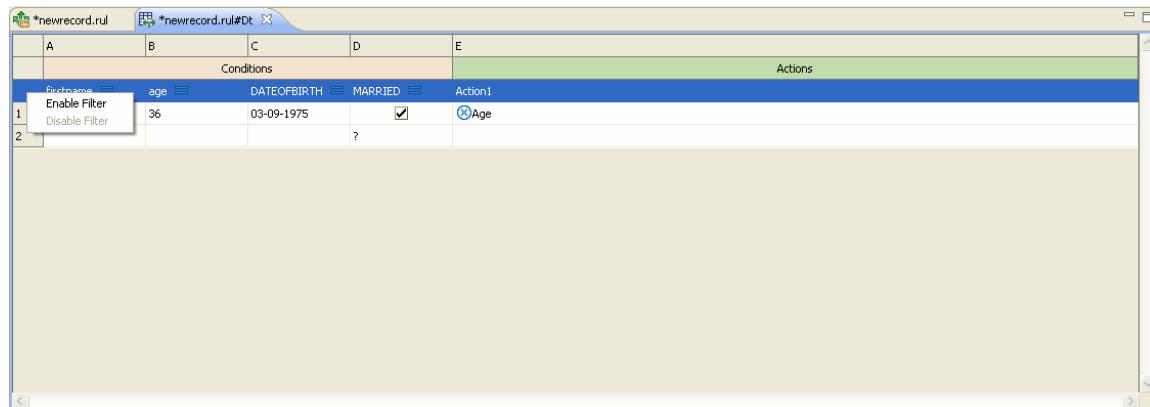
- **Insert Row Above:** Insert a row above the selected row.
- **Insert Row Below:** Insert a row below the selected row.
- **Duplicate:** Insert a duplicate row.
- **Move Row Above:** Move the selected row above the previous row. This is disabled for the first row.
- **Move Row Below:** Move the selected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

Similarly, you can filter and sort the rows.

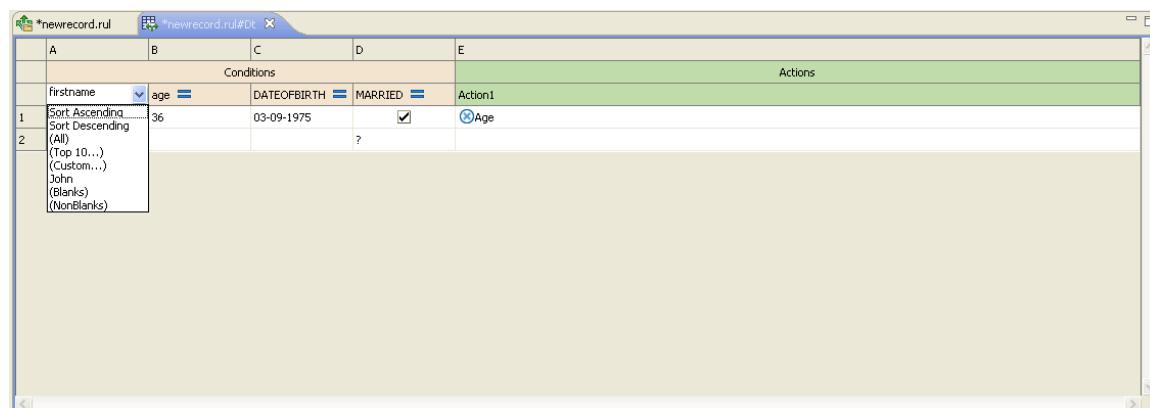
Filtering and Sorting Rows

Procedure

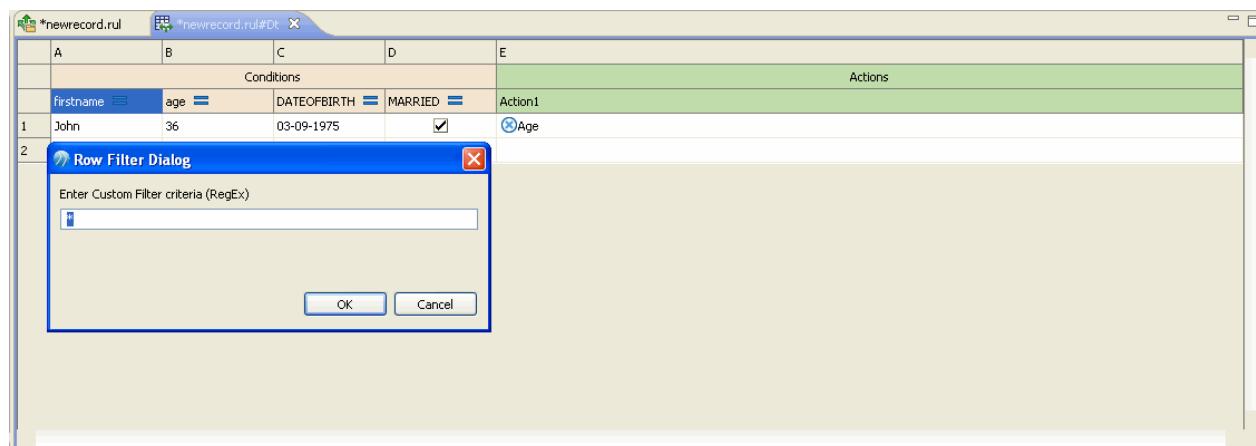
1. Select the Header containing the variable name along with operator symbol and right-click mouse. Click **Enable Filter**.



2. Double-click on 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.



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.



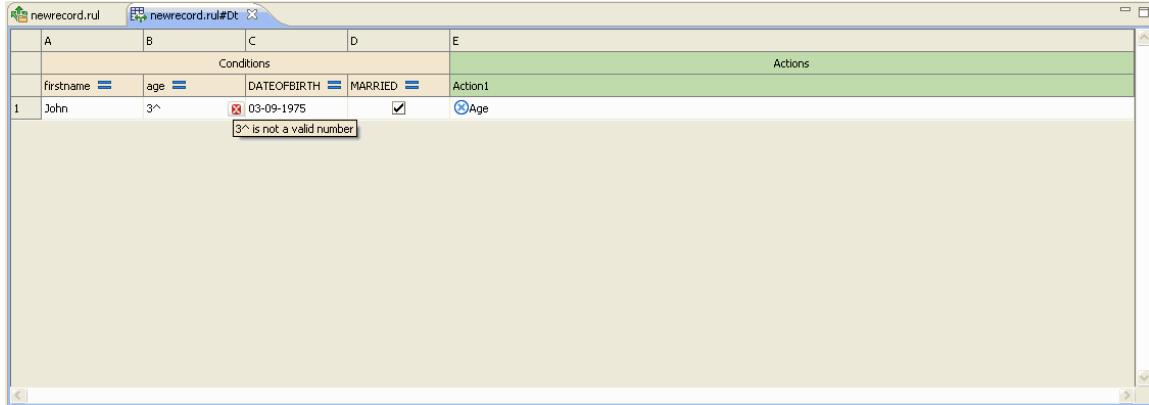
4. Enter the custom filter criteria in the **Enter Custom Filter Criteria (RegEx)** field. The filter criteria should be a regular expression.

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, 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.



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
Contains	*	NA
ContainsAll	NA	NA
Customs	NA	NA
In	NA	*
Neq	*	1
Gt	NA	1
Lt	NA	1
Geq	NA	1

Operator	Number of Values required for multi-value attribute	Number of Values required for Non multi-value attribute
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
- ddmmmyyyy
- yyyy-mm-dd
- yyyy/mm/dd
- dd-mm-yyyy
- dd/mm/yyyy
- dd-mm-yy
- timestamp

Cell Skipping

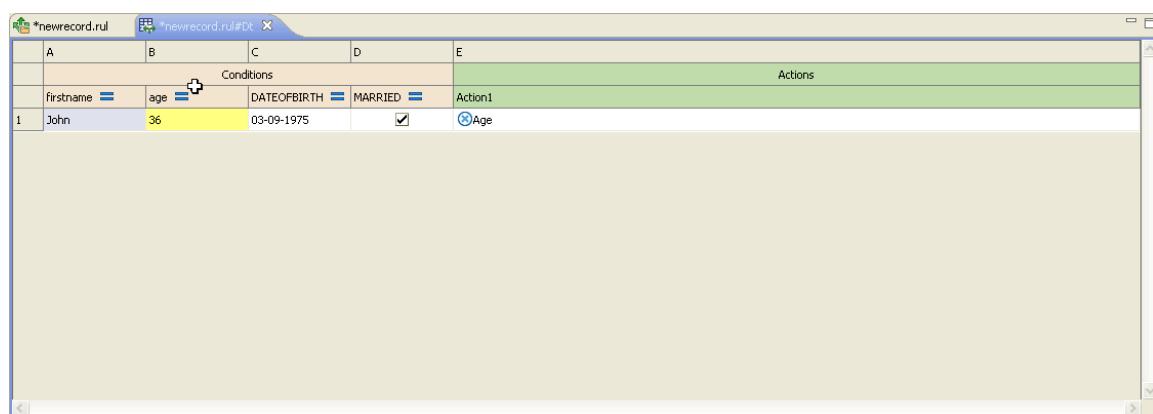
In decision table, skipping a cell means the condition which is of the form [Variable] [Operator] [cell Value] is not evaluated for a given row whose cell is skipped.

The cells which are skipped are marked in Yellow.

Skipping a Cell

Procedure

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



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

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

3. At any give point of time if you want to include the skipped cell, 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:

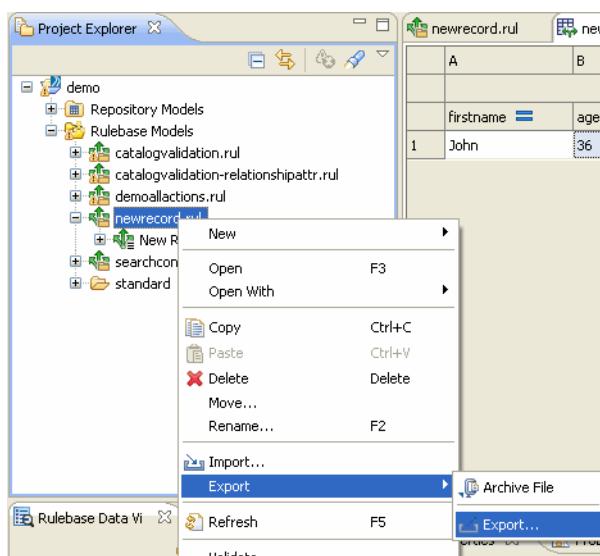
- **DT XML file:** The DT XML files contains all the conditions defined for each row in the decision table.
- **Actions XML file:** The Action XML file contains all the actions defined for each row in the decision table.
- **<rulebase filename>.xml** file: This is the main rulebase file.

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

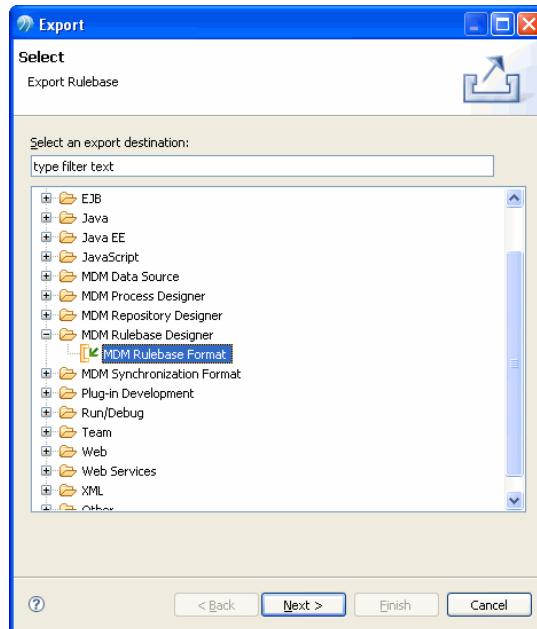
Exporting a Decision Table

Procedure

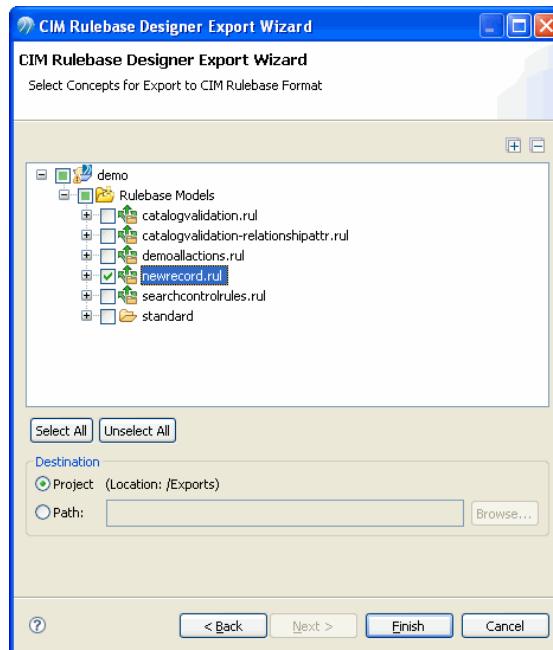
1. Right-click the **rulebase file** in the Project Explorer and click **Export > Export**.



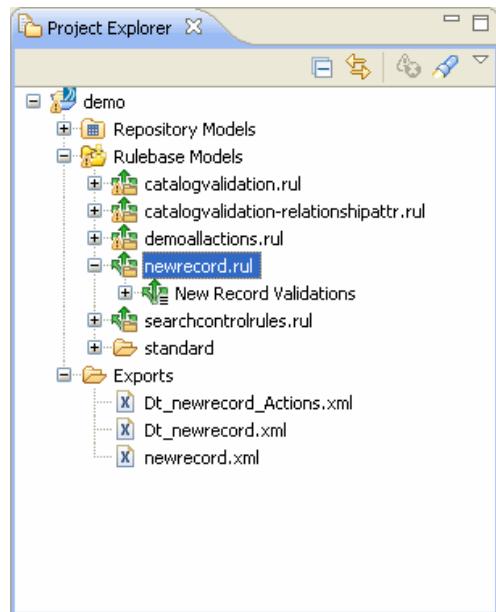
2. The Export Wizard is displayed. Expand **MDM Rulebase Designer** from export destination list, select **MDM Rulebase Format**, and click **Next**.



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**.



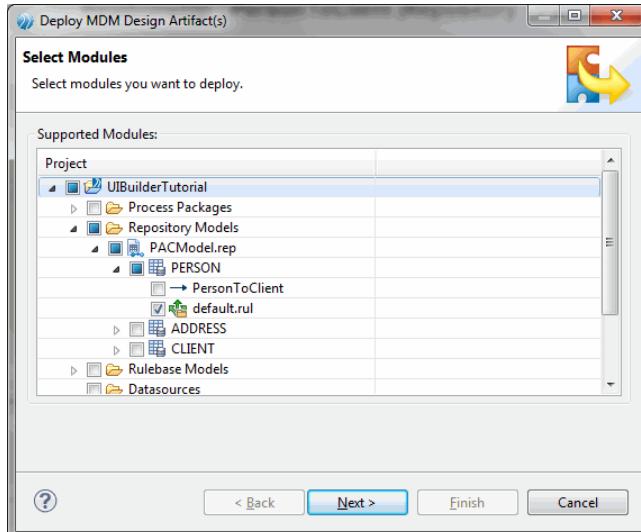
4. The **Exports** folder in the explorer displays the DT XML, Actions XML, and the `<rulebase filename>.xml` files.



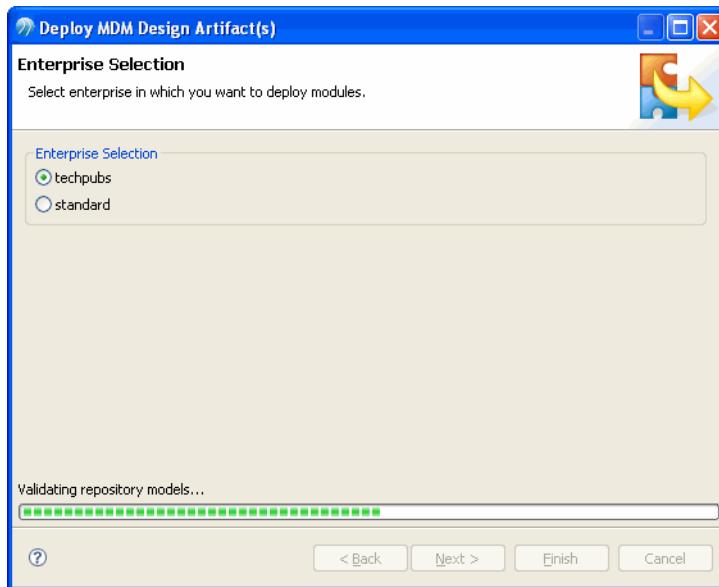
Direct Deploying the Decision Table

Procedure

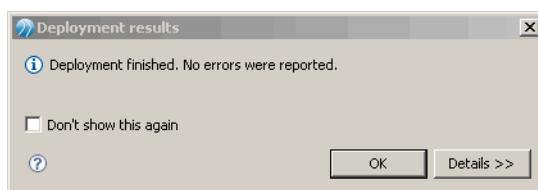
1. In the Deployment pane, right click the <MDM Server and select **Deploy Module**.
2. Select the rulebase module to deploy. Click **Next**.



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



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



Result

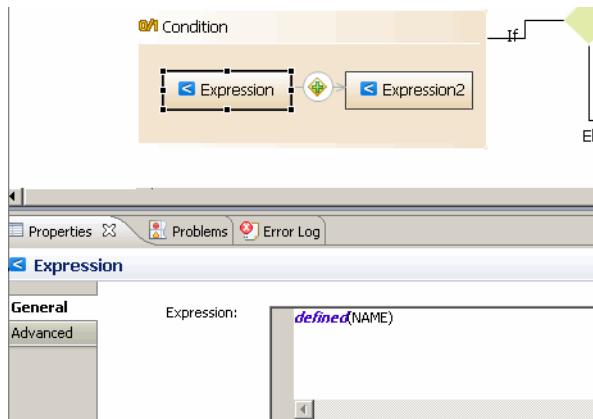
Once successfully deployed, you can log onto the MDM Server and check if your decision tables have got included.

Expressions



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.



Creating Expressions

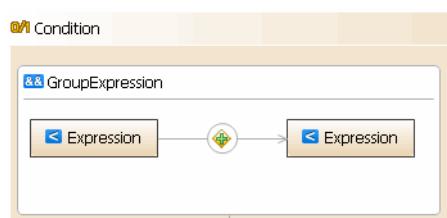
Expressions can be created using the following icons from the **Condition** section of the Palette and dropping in the Condition compartment.:

- **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.

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

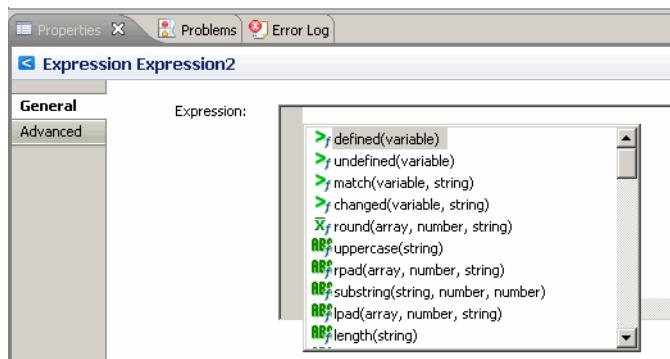
The Expression Editor provides user friendly expression definition functionality such as content assist, syntax completion, and syntax coloring (all functions and reserved words are colored).

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

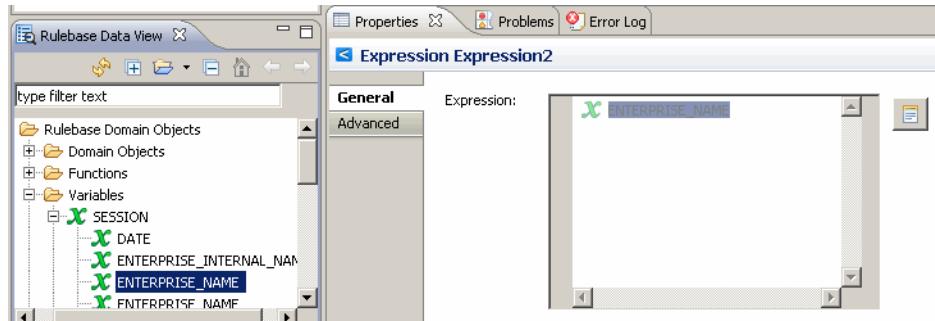
Click in the **Expression** textbox and use the **Ctrl+Spacebar** keyboard shortcut.

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



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.



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 saved 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.



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**.

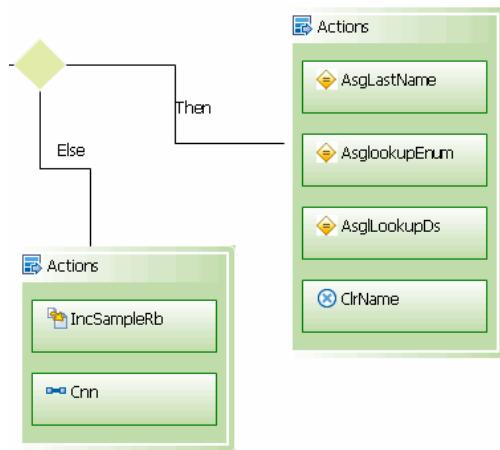
Actions



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

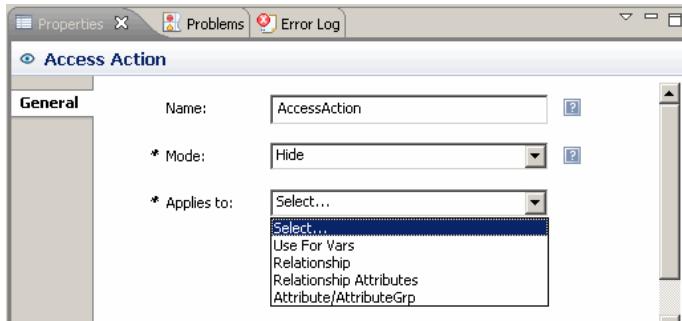
The **Access** action provides control over viewing, modifying, and visibility of Attributes, Attribute Groups and Records.

To define an Access action, select the **Access**

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:** 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.

 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.

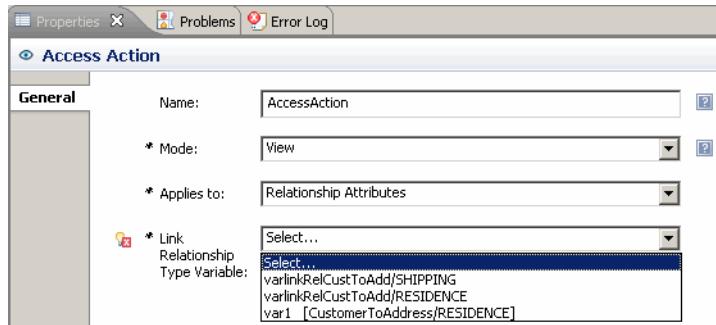
- (Mandatory) **Applies To:** This indicates what the Access action applies to. Values are:
 - Attribute/AttributeGrp
 - Use for Vars
 - Relationship
 - Relationship Attributes

 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 than **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.

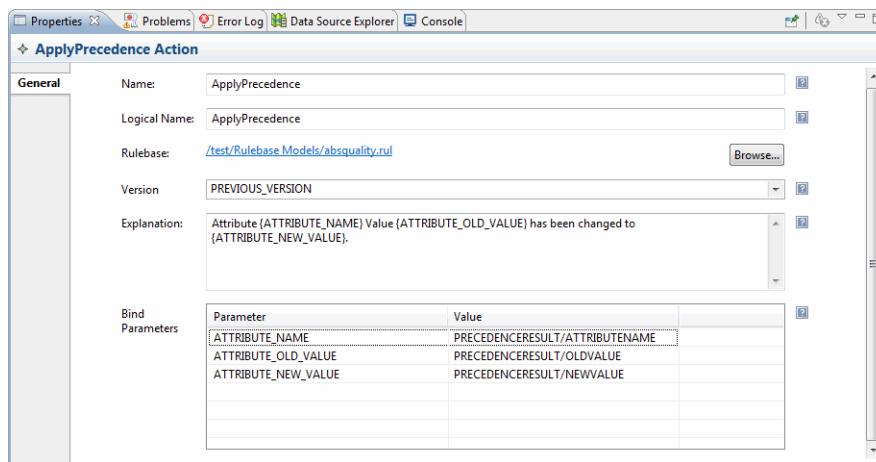


ApplyPrecedence Action Properties

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 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  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 in the appropriate (Else or Then) Action container.

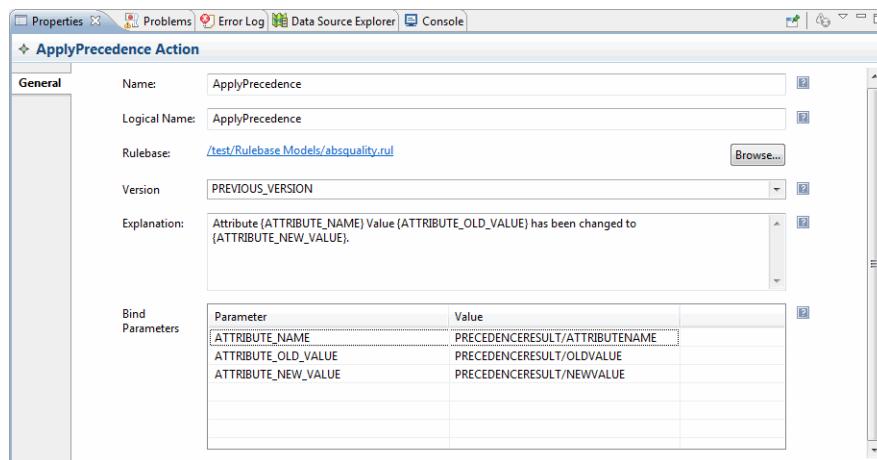
ApplyPrecedence Action Properties

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 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.

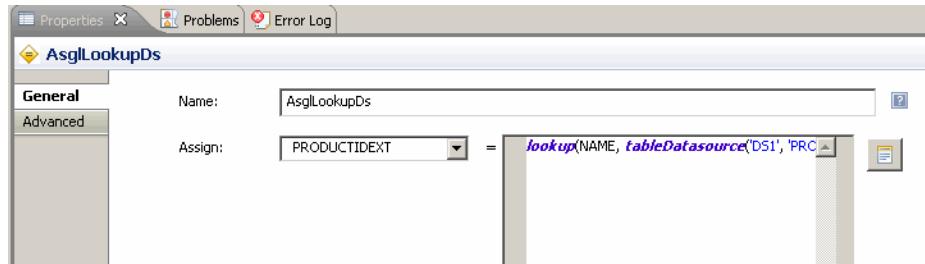
Values can be assigned to declared variables and repository attributes.

To define an **Assign** action, select the  **Assign** 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.



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')`.



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](#).



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 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



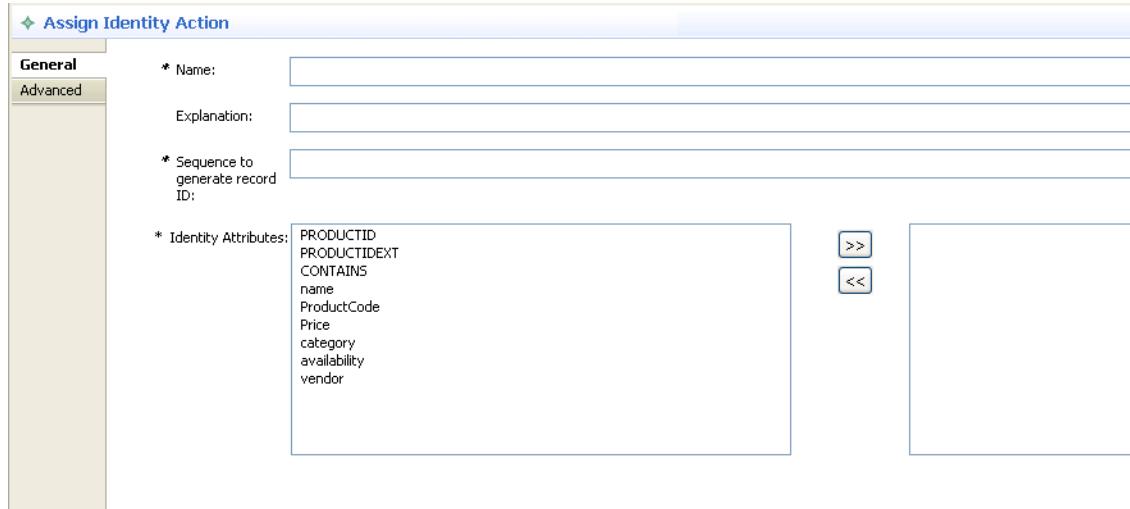
Assign Identity action icon from the palette and drop it in the appropriate (Else or Then) Action container.

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.
- **(Mandatory)Sequence to generate record ID:** The Sequence in which the product 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](#).



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** 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.

The screenshot shows the 'Categorize Action' properties window. The 'General' tab is selected. The 'Name' field is empty. The 'Explanation' field is empty. The 'Incremental' checkbox is checked. The 'Expression' field is empty.

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.

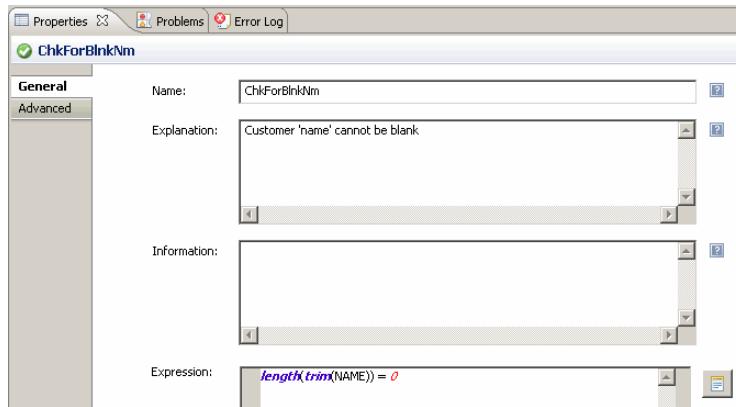
To define a **Check** action, select the  **Check** action icon from the palette and drop it in the appropriate (Else or Then) Action container.

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.



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

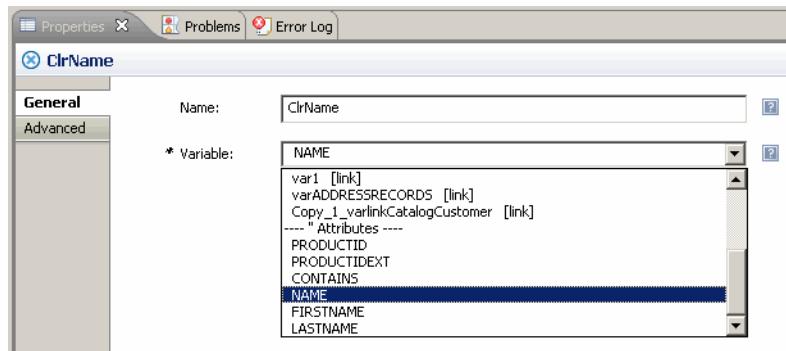
Clear Action Properties

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

General tab properties

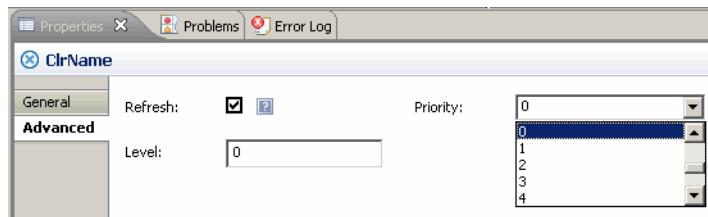
Name: Any logical name for the Check action.

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



Advanced tab properties

- **Refresh:** Refreshes the variable. For details, see [Refresh](#).
- **Priority:** For details, see [Priority](#).
- **Level:** For details, see [Level](#).



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

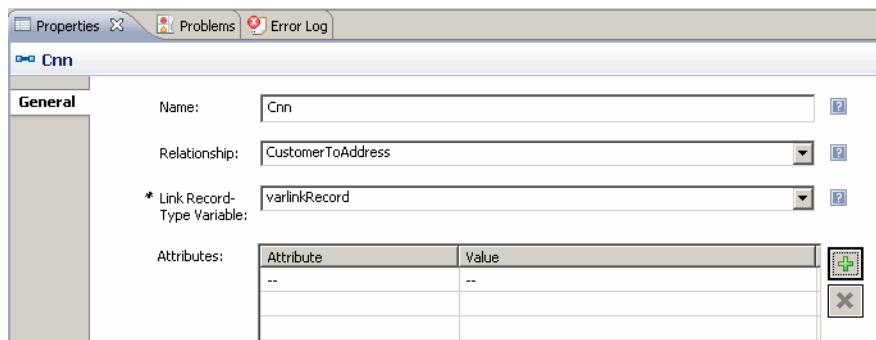
The **Connect** action is used to establish a relationship between records. It establishes a relationship between the record being processed and accessed records

To define a **Connect** action, select the **Connect** 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.



Connect Action Validation

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

Disconnect Action

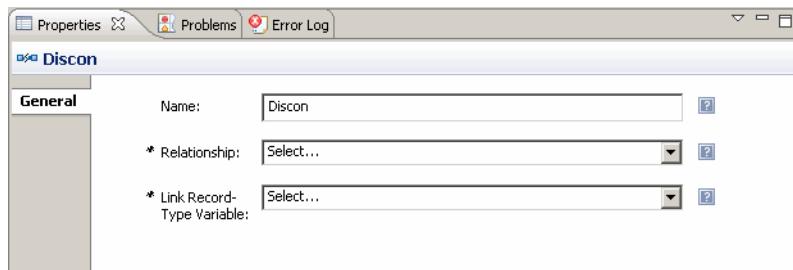
The **Disconnect** action is used to remove relationships between records. Any relationship attributes defined with the relationship are also removed.

To define a **Disconnect** action, select the **Disconnect** 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.



Disconnect Action Validation

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

Include Action

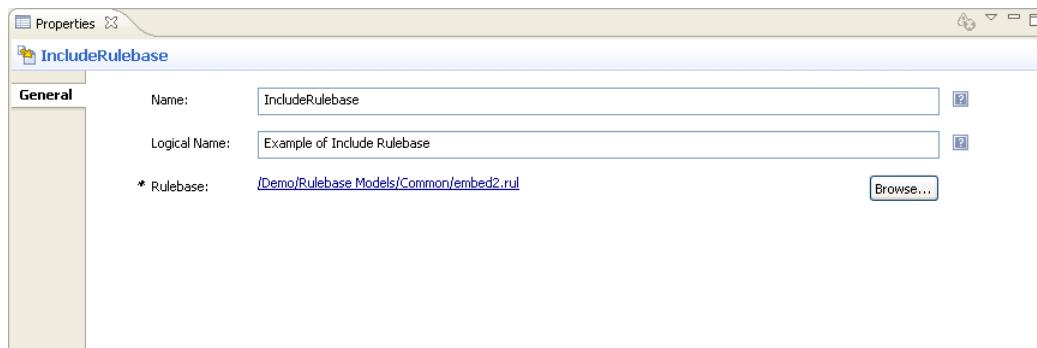
The **Include** action allows embedding one rulebase file into another.

To define a **Include** action, select the  **Include** 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.



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.

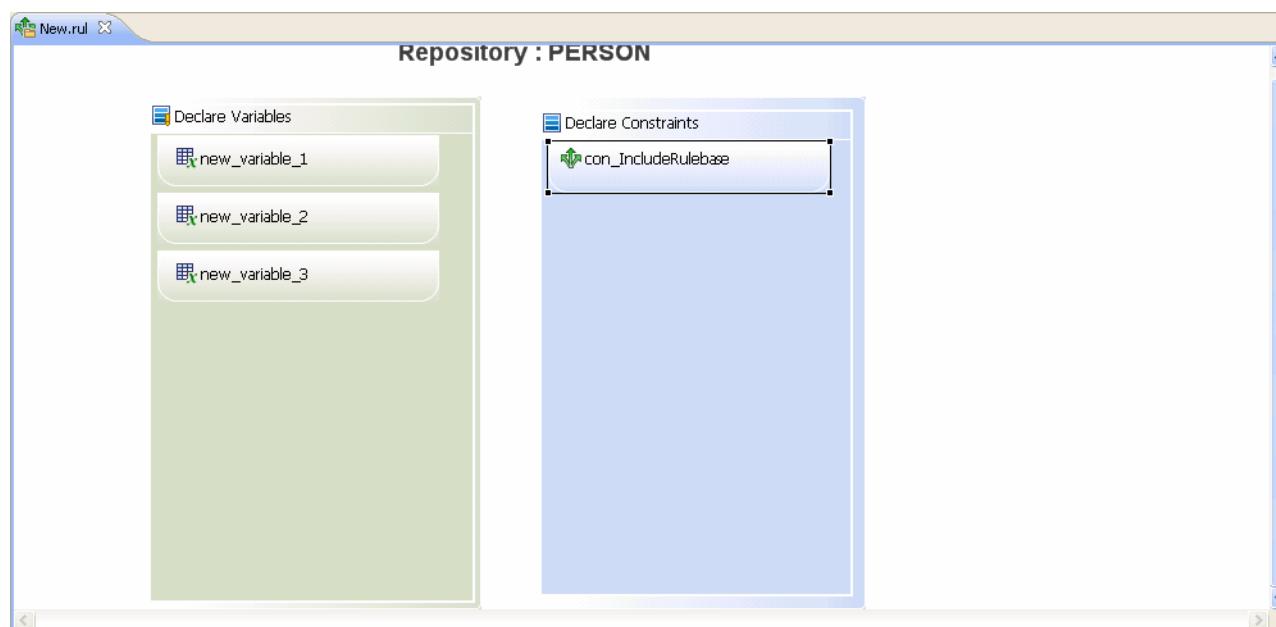
To list the variables defined in the included rulebase file in the variable listing of current rulebase file in the rulebase data view. In addition it is also available in the content assist of the expression editor.

Including a Rulebase in a Current Rulebase File

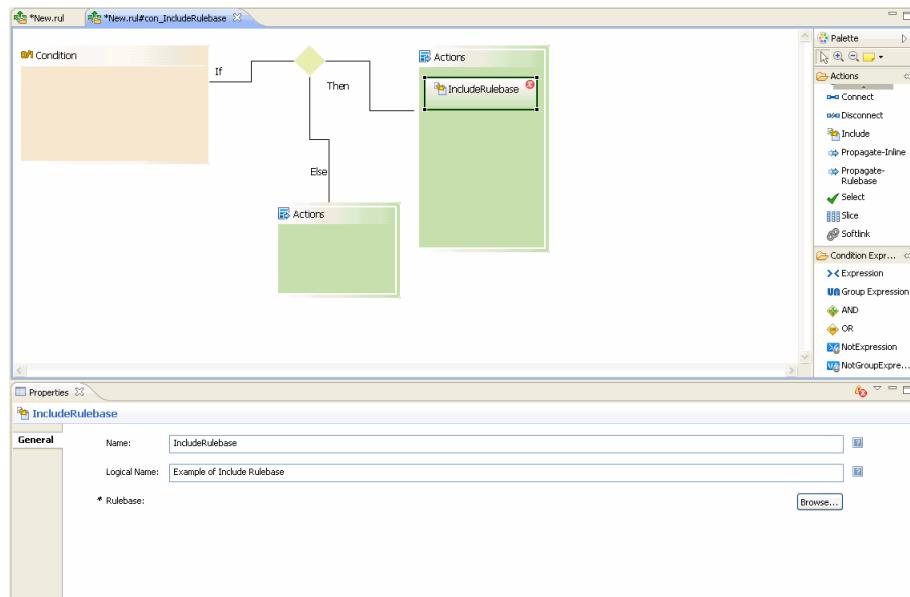
To include a rulebase in a current rulebase file perform the following steps:

Procedure

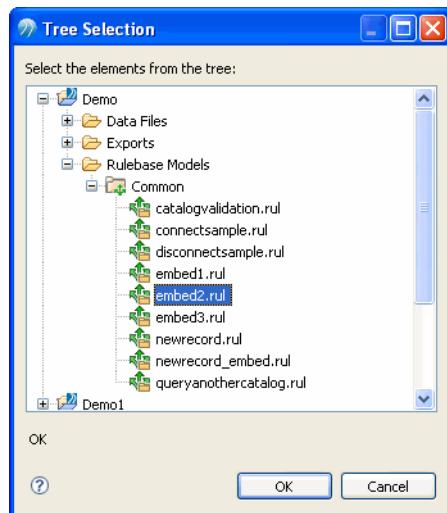
1. Create a rulebase file and declare the variables and constraints.



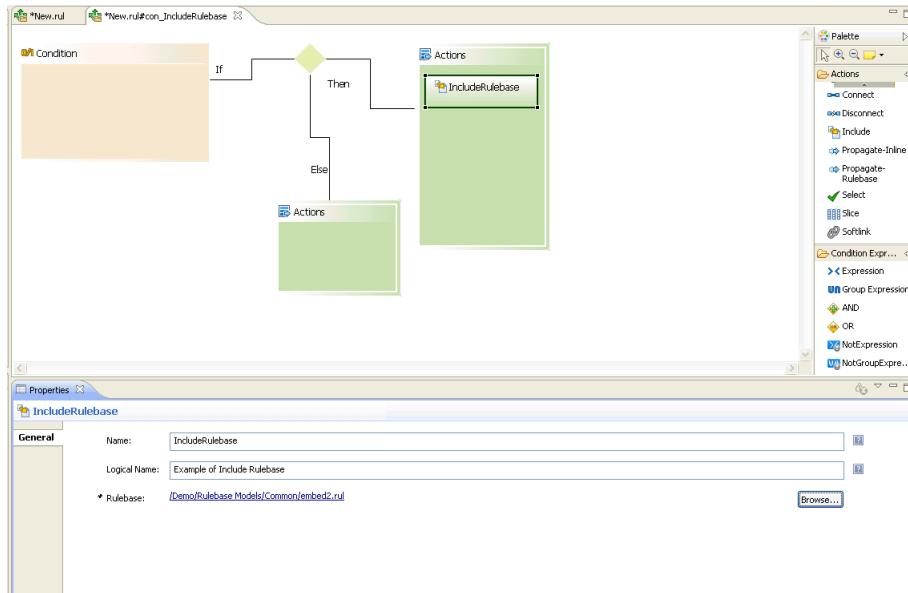
2. Add Include Action to the constraint.



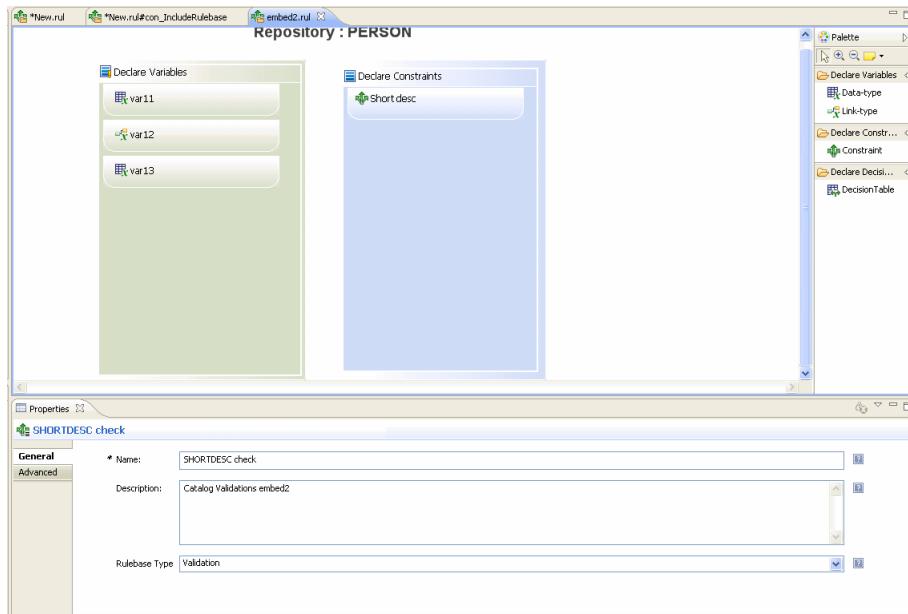
3. Click **Browse** in Include action properties section. The pop up window



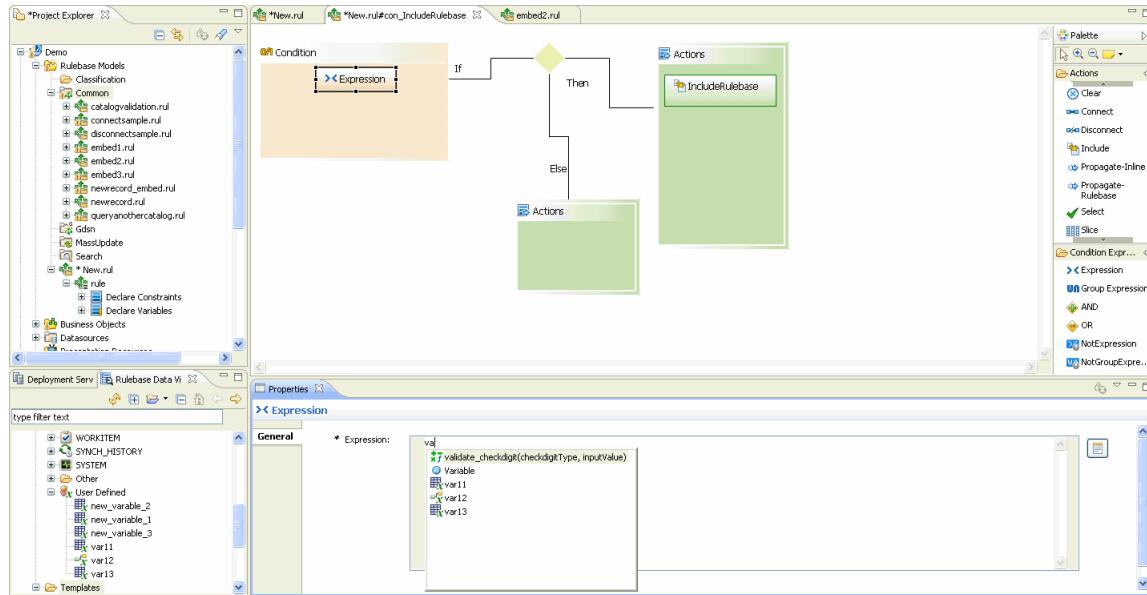
4. The pop up tree selection window is displayed. Navigate to the common folder and select the rulebase which you want to include and click **OK**.



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



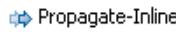
- The new variables defined in the included rulebase appears in the variable listing of current rulebases in the rulebase data view and also in the content assist of expression editor.



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

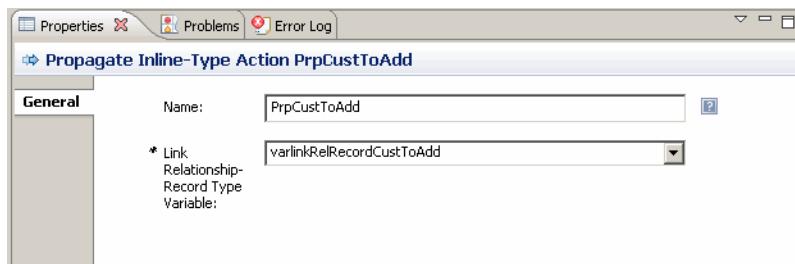


Propagate-Inline action icon from the palette and drop it in the appropriate (Else or Then) Action container.

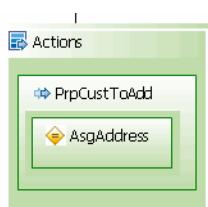
Propagate-Inline Action Properties

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.



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

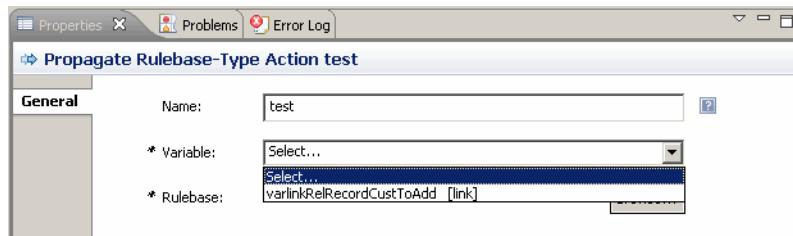


Propagate-Rulebase action icon from the palette and drop it in the appropriate (Else or Then) Action container.

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 Validation

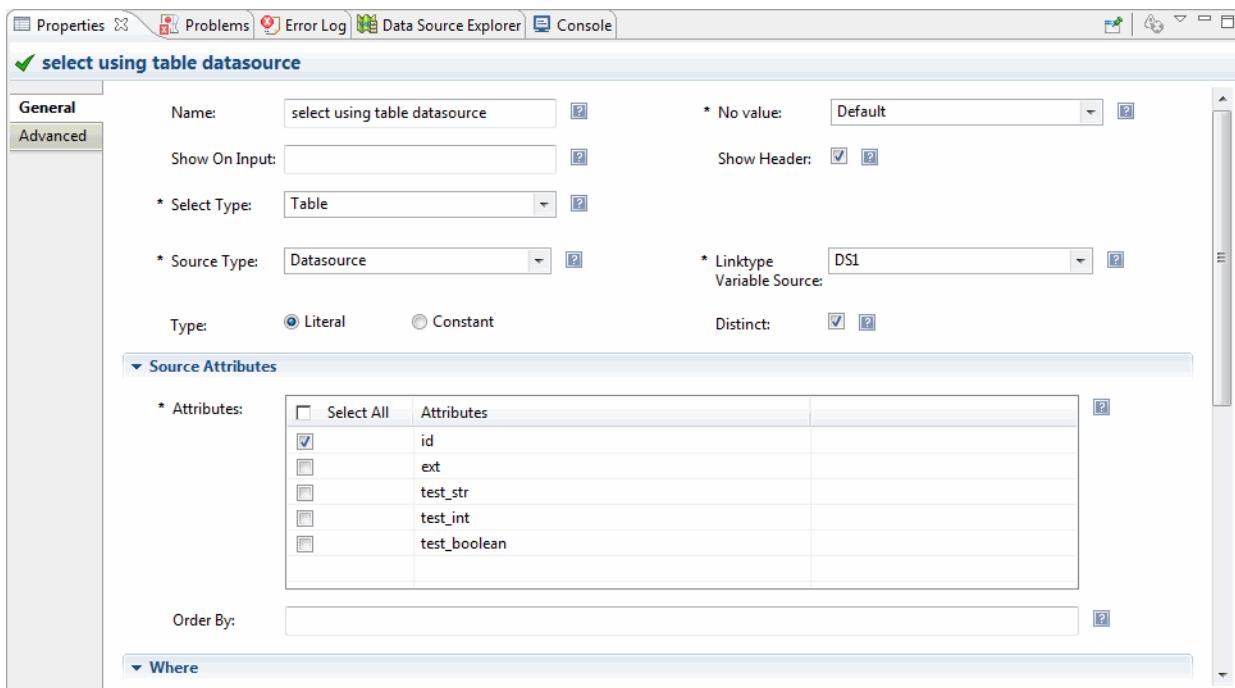
(Short Description to be added).

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 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.



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



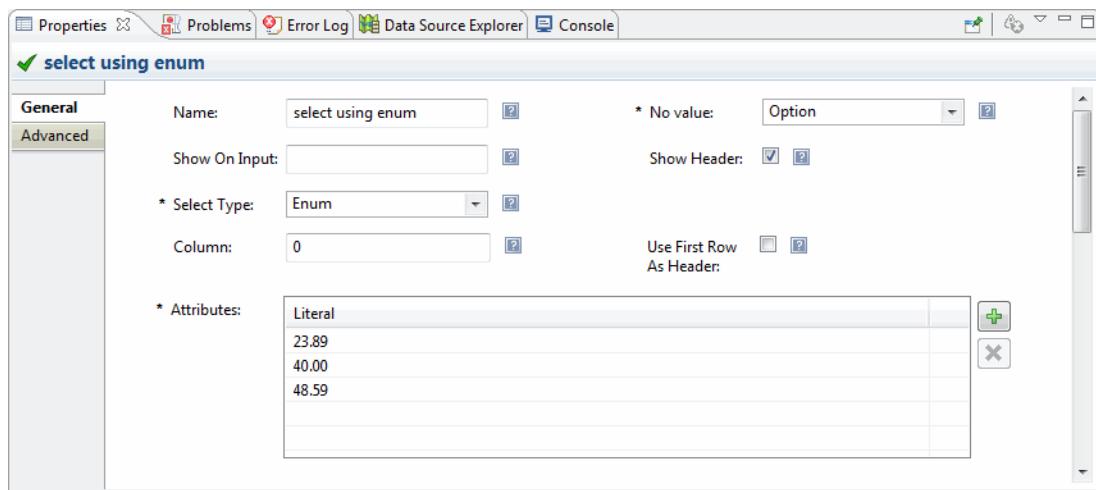
Select action icon from the palette and drop it in the appropriate (Else or Then) Action container.

Select Action Properties

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

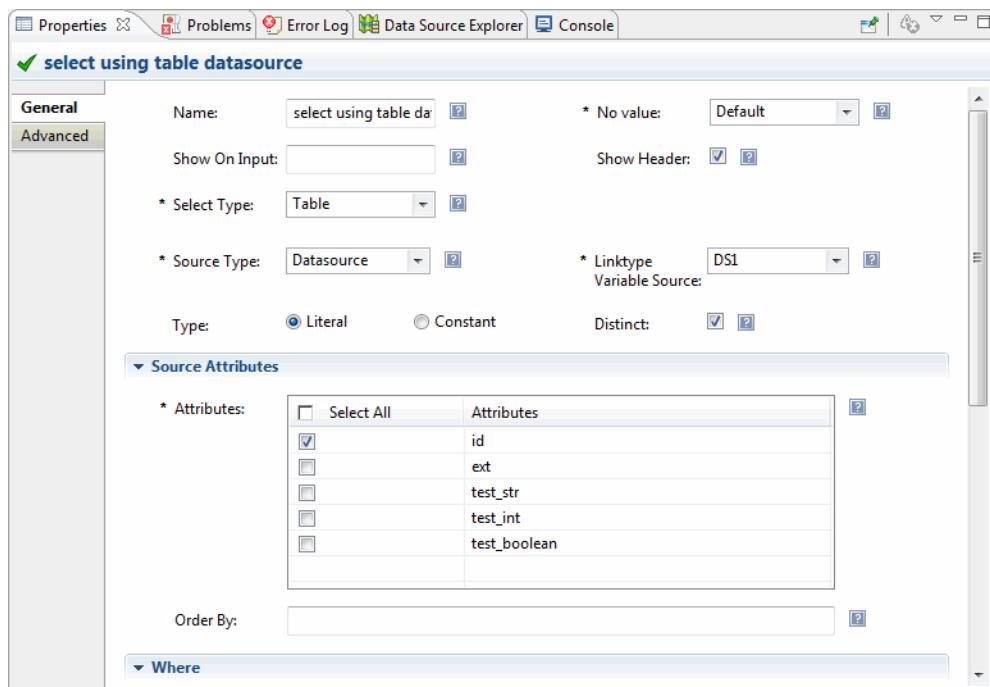
- **Name:** Any logical name for the Select action.
- **No Value:** This specifies where an empty value appears (in the list). The following options are available:
 - **Default:** The empty value appears as the first value in the list.
 - **Option:** The empty value appears at the bottom of the list.
 - **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:



- 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 the static constants values to the table using the Add button on the right of the table. Use the delete button to delete values from the table.

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



- 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="[-]<columnposition1>[-]<columnposition2>..."`

where:

- indicates descending order
- <columnposition> 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

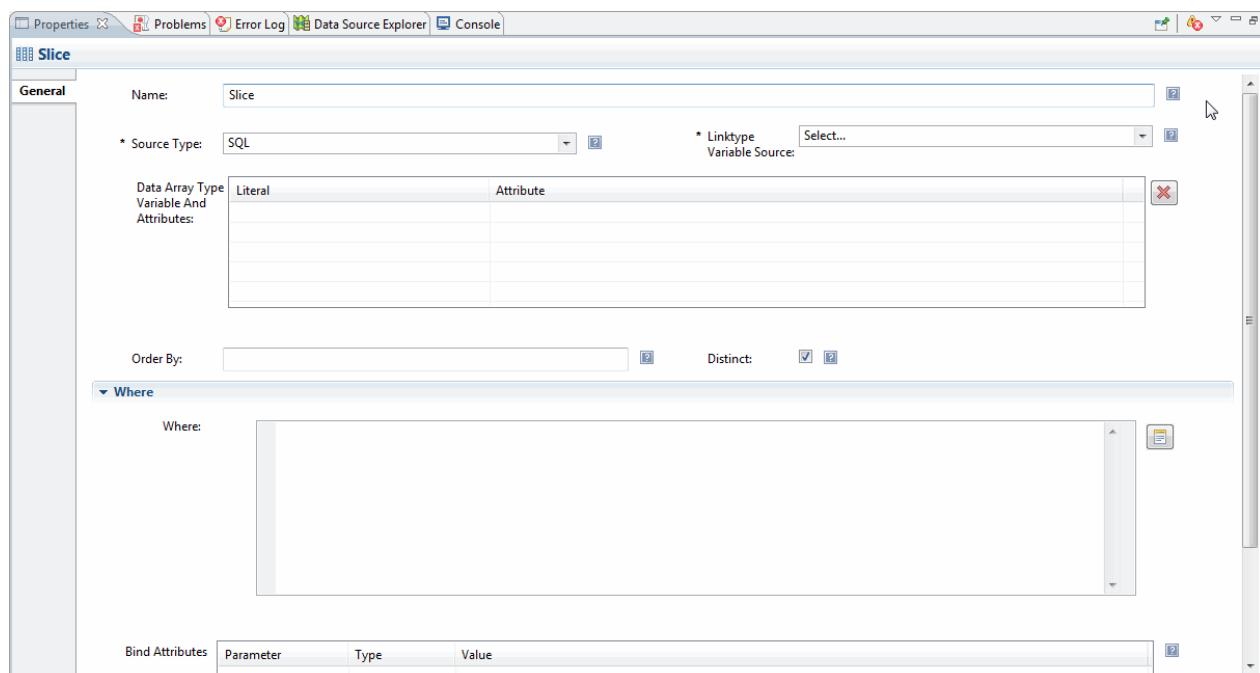


Slice action icon from the palette and drop it in the appropriate (Else or Then) Action container.

Slice Action Properties

(Short Description to be added).

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



- **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.
 - 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.

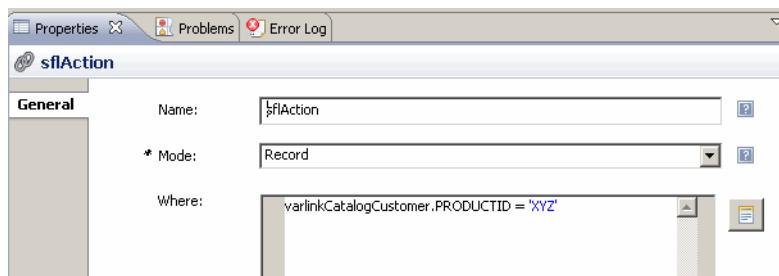
To define a **Softlink** action, select the



Softlink 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:** 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

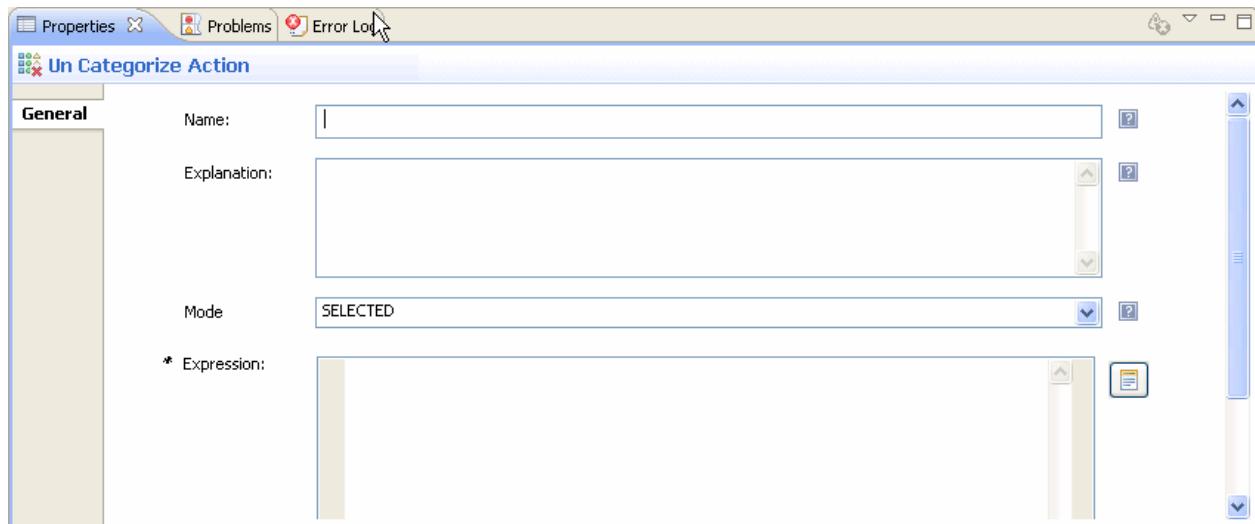


UnCategorize action icon from the palette and drop it in the appropriate (Else or Then) Action container.

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.



UnCategorize Action Validation

(Short Description to be added).

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

(Short Description to be added).

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

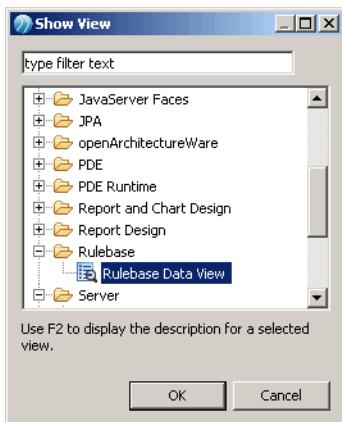
The **Rulebase Data View** contains elements that can be used when building rulebases.

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 Repositories Roles Users	For details, see Domain Objects .
Operators	Math Operators Comparison Operators	For details, see Operators .
Functions	Comparison	For details, see Functions .

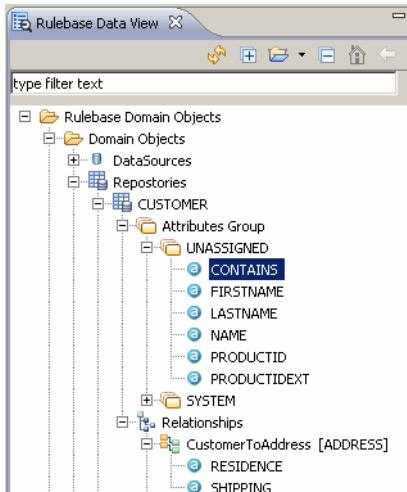
Components	Details
Math	
String	
Other	
Variables	Session For details, see Implicit Context Variables .
	Workitem
	Synch_History
	System
	Other
	Workflow
	AttributeHistory
	AttributeQuality
	PrecedenceResult
	User Defined
Templates	For details, see Templates .

Domain Objects

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



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

Operators

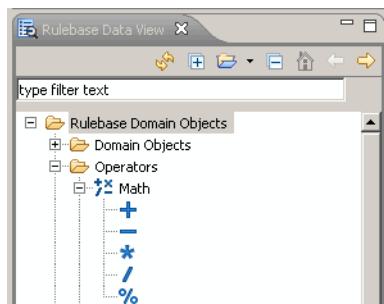
This section contains Math and Relational Operators that can be used in expressions.

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

(Short Description to be added).

- 
- [Minus](#)
- 
- [Div](#)
- 
- [Mult](#)
- 
- [Percent](#)
- 
- [Plus](#)



Minus

(Short Description to be added).

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

Div

(Short Description to be added).

Description	Operands	Returns	Example
Calculates division of values.	1...n numeric expressions.	Calculated numerical. Division by 0 returns null. By default, results are rounded to 8 decimal places.	Arg1 / Arg2 / Arg3 / ... Argn

Mult

(Short Description to be added).

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

Percent

(Short Description to be added).

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

Plus

(Short Description to be added).

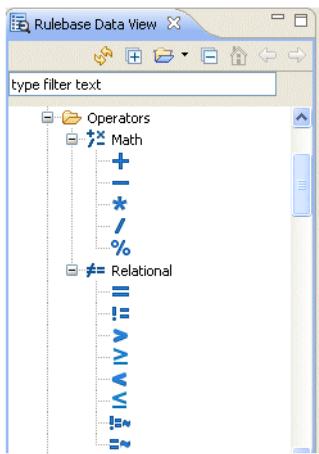
Description	Operands	Returns	Example
Calculates addition of values	1...n numeric expressions.	Calculated numerical.	Arg1 + Arg2 + Arg3 +... Argn

Relational Operators

(Short Description to be added).

-  [Equal to](#)
-  [Not Equal to](#)
-  [Scalar Matching - Not equal to](#)
-  [Less Than](#)
-  [Less Than or Equal](#)

- Less than equal to
- \leq
- Equal to
- \approx
- Scalar Matching - Equal to
- \gt
- Greater Than
- \geq
- Greater Than Equal To



Not Equal to

(Short Description to be added).

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

(Short Description to be added).

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 matches the Regular Expression Arg2..	Arg1 !=~ Arg2

Less Than

(Short Description to be added).

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

(Short Description to be added).

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

(Short Description to be added).

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

(Short Description to be added).

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

(Short Description to be added).

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

(Short Description to be added).

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](#)



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

Comparison Function	Math Functions	String Functions	Other Functions
		trim	filter
		uppercase	invokeJavaAPI
			syncOperationAttribute
		max	tableDatasource
		min	tableSql

Comparison Functions

(Short Description to be added).

changed

(Short Description to be added).

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') changed(SHORTDESC, PREVIOUS_CONFIRMED_VERSION)

defined

(Short Description to be added).

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

(Short Description to be added).

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%')

(Short Description to be added).



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

match

(Short Description to be added).

Description	Syntax	Parameters	Returns	Example
Regular expression match.	match(variable, regexStr)	variable: Variable to match regexStr: Regular Expression	true — if first operand matches second operand. false — if first operand does not match second operand.	match(GTIN, '/^\d{14}\$/')

undefined

(Short Description to be added).

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	<code>undefined(FIRST_NAME)</code>

Math Functions

(Short Description to be added).

round

(Short Description to be added).

Description	Syntax	Parameters	Returns	Example
Rounds to a defined set of decimal places.	round(num, decimalPlaces, roundingMethod)	<p>num: Number or array of numbers.</p> <p>decimalPlaces: Number of decimal places to round to (Optional). Default to 0.</p> <p>roundingMethod: Rounding method (Optional)</p> <ul style="list-style-type: none"> 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 result. UP - increment and round up to zero. 	The array of number values.	<pre>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')</pre>

String Functions

(Short Description to be added).

concat

(Short Description to be added).

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

(Short Description to be added).

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

lpad

(Short Description to be added).

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

(Short Description to be added).

Description	Syntax	Parameters	Returns	Example
Pads a string on the right.	<code>rpad(strToPad, paddedStrLen, charToPad)</code>	<p>strToPad: string or array of strings.</p> <p>paddedStrLen: final length of padded string.</p> <p>charToPad: character to pad with (Optional, Default is SPACE).</p>	String.	<code>rpad('ABC', 4)</code> <code>rpad({'ABC', 'DEF'}, 4)</code>

substring

(Short Description to be added).

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

trim

(Short Description to be added).

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

uppercase

(Short Description to be added).

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

(Short Description to be added).

checkdigit

(Short Description to be added).

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

(Short Description to be added).

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

(Short Description to be added).

Description	Syntax	Parameters	Returns	Example
Returns distinct values.	distinct(values)	values: Array of values	Distinct values.	distinct(CHILDCITIES) count(distinct(SIBLING_REL.UOM))

duplicate

(Short Description to be added).

Description	Syntax	Parameters	Returns	Example
Checks for records with duplicate values.	duplicate(values)	values: value or array of values.	true : otherwise. false : if duplicate.	duplicate(RECORD_TYPE) duplicate(NAMES)

enum

(Short Description to be added).

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

(Short Description to be added).

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

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

invokeJavaAPI

(Short Description to be added).

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

lookup

(Short Description to be added).

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

max

(Short Description to be added).

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

(Short Description to be added).

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

(Short Description to be added).

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

(Short Description to be added).

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

(Short Description to be added).

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.	<pre>sequence('MQ_SEQUENCE_1') sequence(mysequence) concat('HS=' lpad(sequence(' MQ_SEQUENCE_1'), , 9, '0'), '-' JAL')</pre>

strip

(Short Description to be added).

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

(Short Description to be added).

Description	Syntax	Parameters	Returns	Example
Gets record synchronization status.	Synchstatus(markplace, operation)	<p>marketplace: Marketplace/ Datapool Organization name.</p> <p>operation: Operation name (Optional, Default is ADD. PUBLISH, LINK, COMMIT are other options).</p>	<p>true - if record has ever been Added/ Published/ Linked/ Committed to this Marketplace/ Datapool.</p> <p>false - otherwise.</p>	<pre>synchstatus('UC Cnet') synchstatus('UC Cnet', 'PUBLISH')</pre>

syncOperationAttribute

(Short Description to be added).

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

tableDatasource

(Short Description to be added).

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

tableSql

(Short Description to be added).

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 (Optional): The valid values are DISTINCT_TRUE and DISTINCT_FALSE.	Array of records.	tableSql(var.co untryname, where COUNTRYNAME like('Arg%')) Where var is a link type variable pointing to the repository.

toDate

(Short Description to be added).

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

(Short Description to be added).

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](#).

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

(Short Description to be added).

Procedure

- 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"))
{
```

```
returnValue = checkNumber(inArgs);
}
```

Compile the RulebaseCustomFunction.java. For example:

```
javac RulebaseCustomFunction.java -classpath $MQ_HOME/lib/mq/AlleCMClasses.jar:  
$MQ_HOME/lib/external/log4j-1.2.14.jar
```



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.

Custom Rulebase Class Example

(Short Description to be added).

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

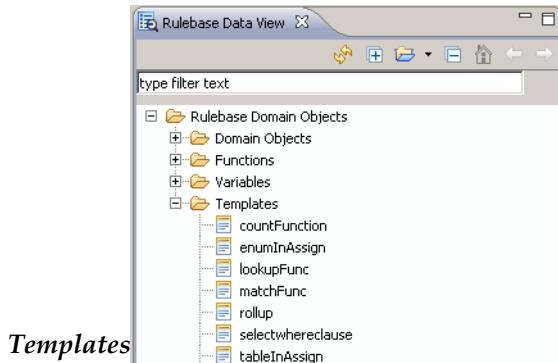
Variables

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.



Template	Description	Syntax
countFunction	<p>This template uses the count and filter functions (count and filter).</p> <p>The result of filter is assigned to a temporary variable, i.e. PERISHABLE_LIST, that the count function operates on.</p>	<pre>count(PERISHABLE_LIST := filter(CHILD_RECORDS, where PERISHABLE = 'YES' and DD = 'X')) > 1</pre>
lookupFuncUsingEnum	<p>This template uses the enum and lookup functions (enum and lookup) to look for TEMPCODE in the enum list.</p>	<pre>lookup(TEMPCODE, enum({'FA<sep / >Fahrenheit', 'CE<sep / >Celsius'},2))</pre>

Template	Description	Syntax
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)
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	<p>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.</p> <p>CONTAINSREL and CONTAINS_RECORD are Linktype-Relationship type user defined variables.</p>	+(CONTAINSREL.QUANTITY * nvl(CONTAINS_RECORD.NETWEIGHT,100))
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='ssssssss')
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

(Short Description to be added).

This chapter explains direct deployment of rulebases created in 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 a MDM Deployment Server

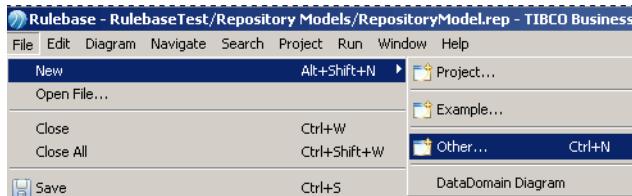
(Short Description to be added).

MDM network deployment service requires administrative privileges.

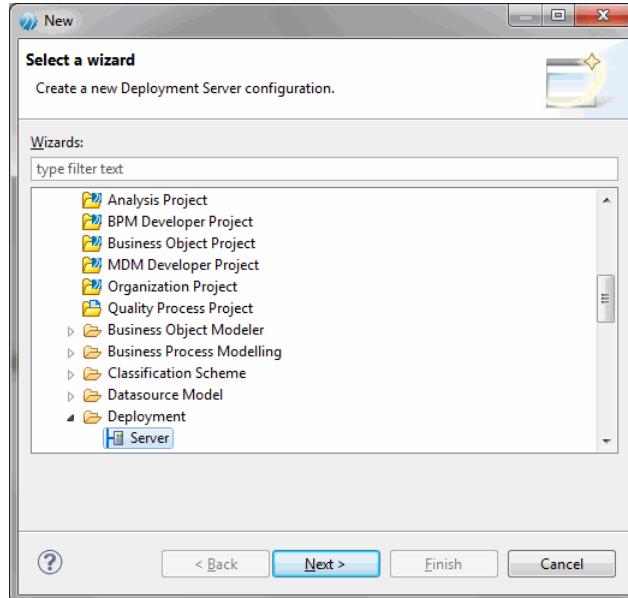
In case you do not see the Deployment Server Pane, go to **Window > Show view > Other** and select **Deployment Server** under **Studio**. Click **OK**.

Procedure

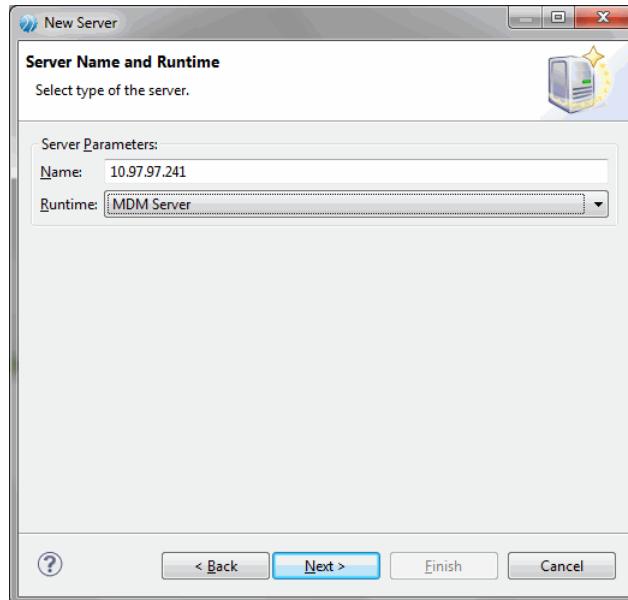
1. Select **File > New > Other**.



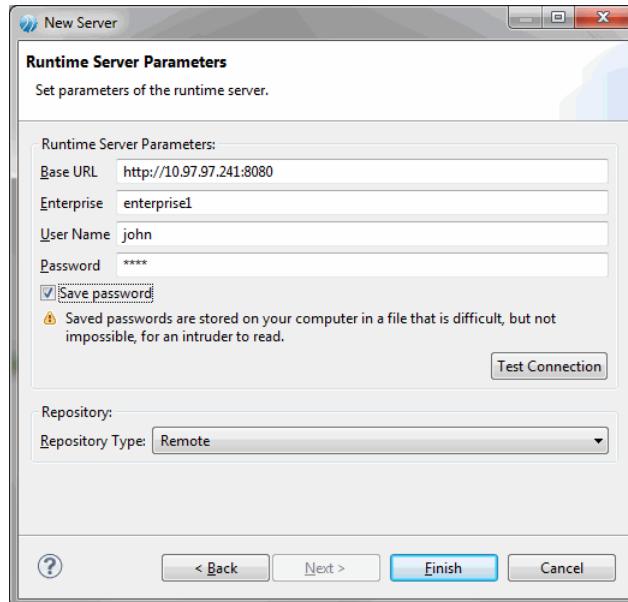
2. Select **Business Modeling > Deployment > Server**.



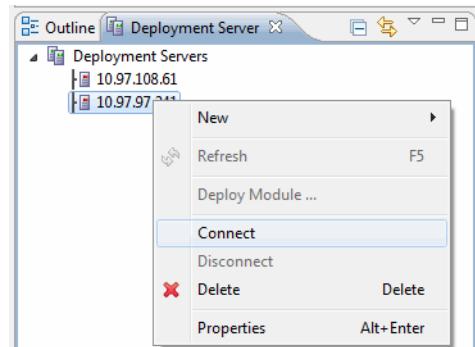
3. Provide the Server Name; select **MDM Server** as Runtime.



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 receive a successful connection message. Click **Finish**.



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

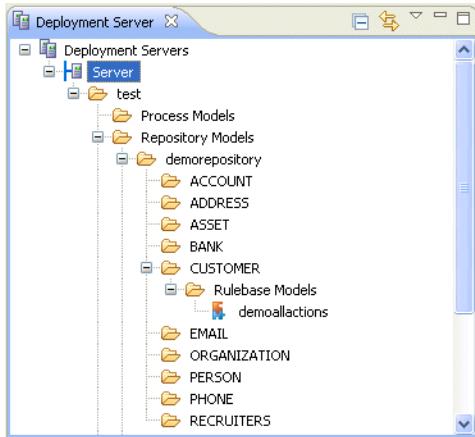


If you did not choose to save the password, you will be prompted to enter the password.



Result

This establishes a connection between the MDM Server and 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



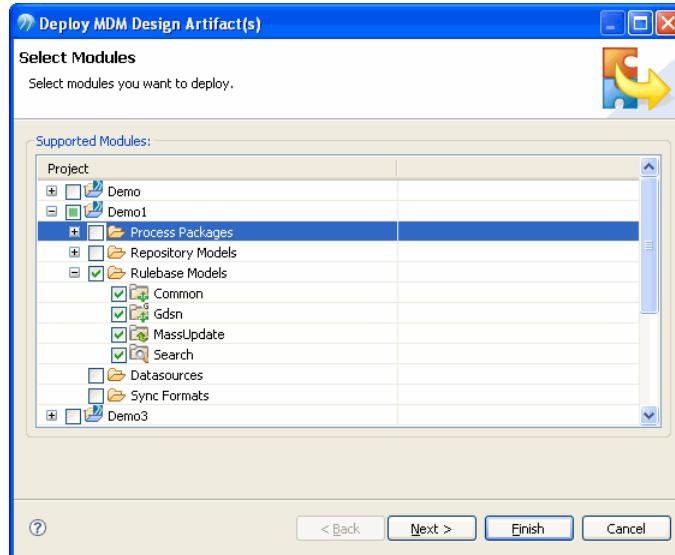
If you want to use SSL enabled URL, enter the URL in **Base URL** field as "https://<hostname>:<port>".

Direct Deploying of Rulebases

The following are the steps involved in directly deploying rulebases:

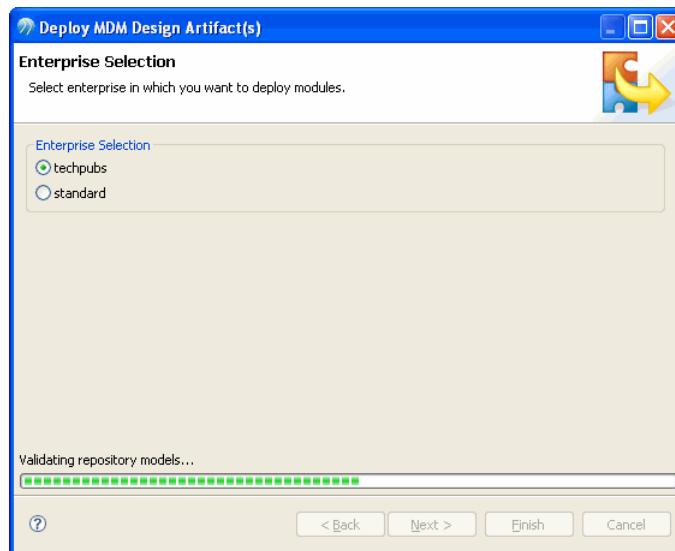
Procedure

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



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**.



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 MDM Server and check if your rulebases and repositories have got included.

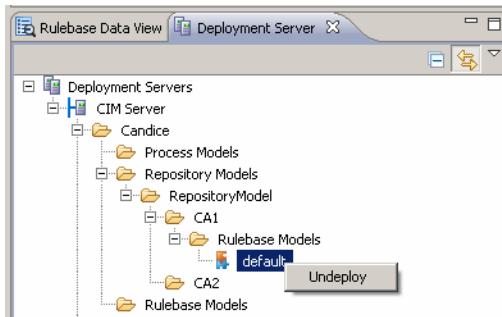
Undeploying Rulebases

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

(Short Description to be added).

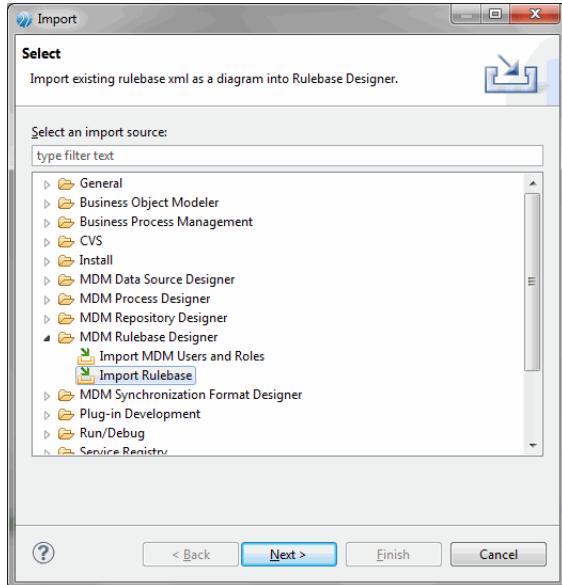
This chapter explains importing MDM rulebases into MDM Studio and exporting rulebases created in MDM Studio.

Importing Rulebases

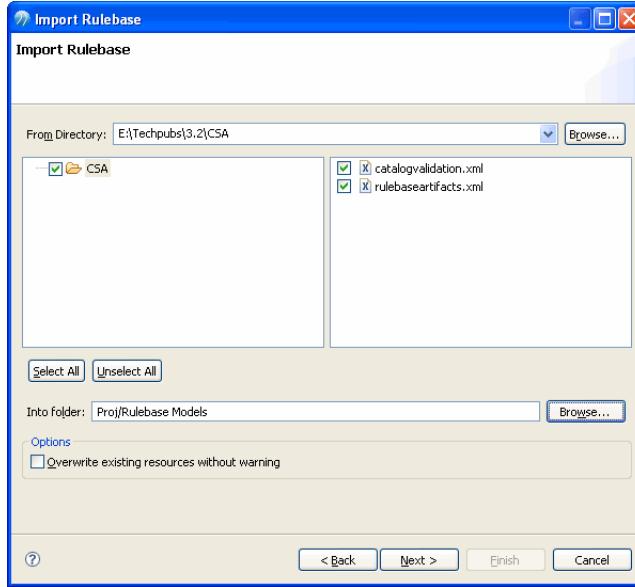
Existing MDM rulebases can be imported into the Rulebase Designer. Follow these steps to import rulebases:

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**.

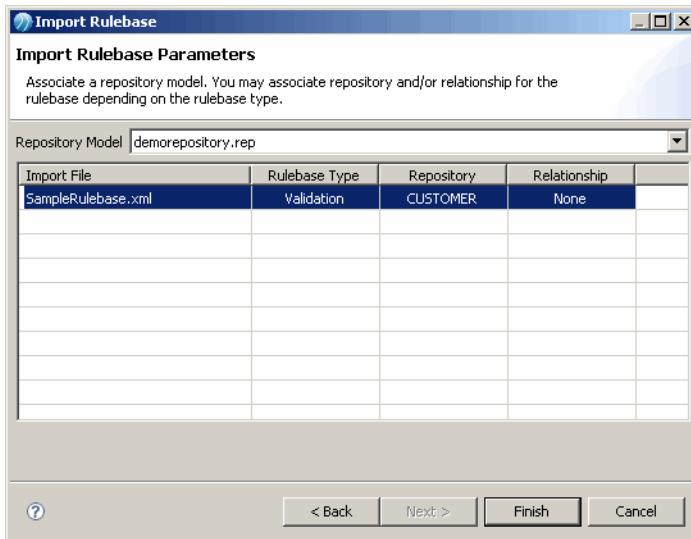


3. Browse to the location (folder) of the Rulebase (xml) and select the checkbox against the appropriate file. You can also Browse to change the folder that the Rulebase should be imported into. Click **Next**.



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



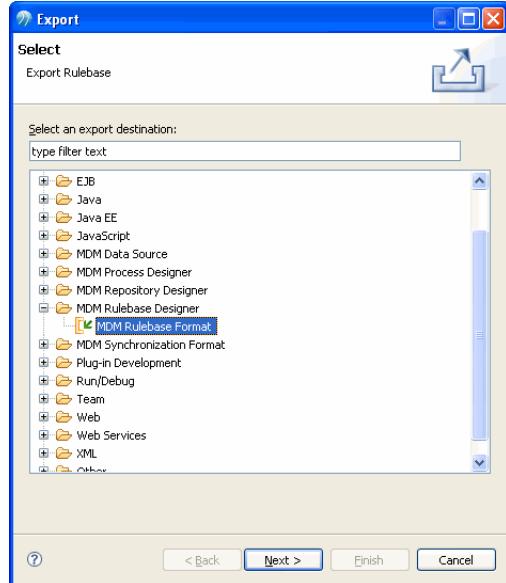
The imported rulebase will then be displayed in the appropriate folder (the one selected for import into.)

Exporting Rulebases

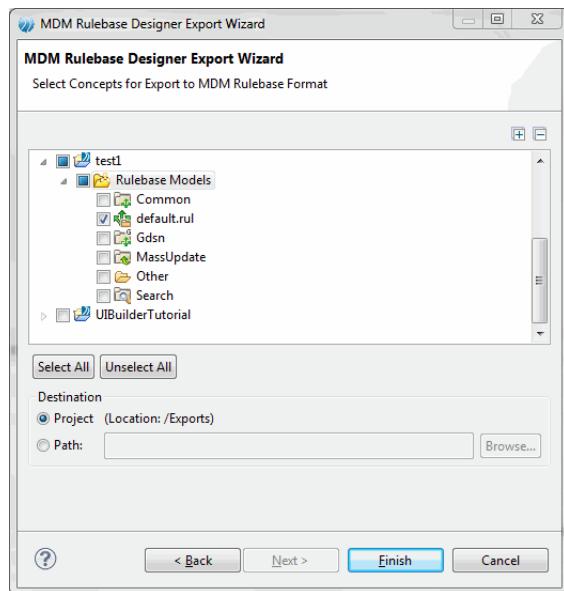
Follow this process to export rulebases designed into the Rulebase Designer into MDM.

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

(Short Description to be added).

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

Sample - 1

(Short Description to be added).

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

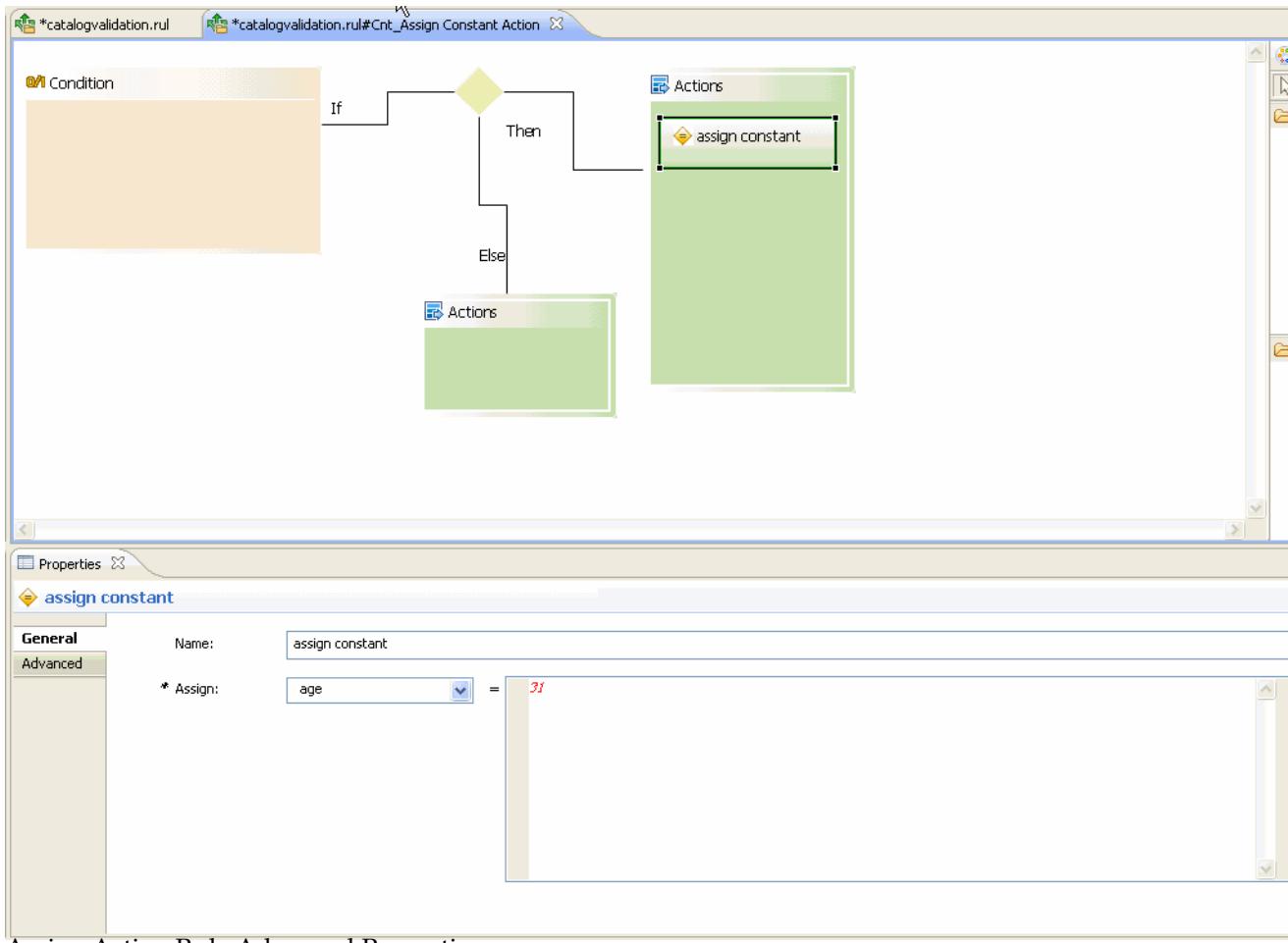


Assign Action Constant

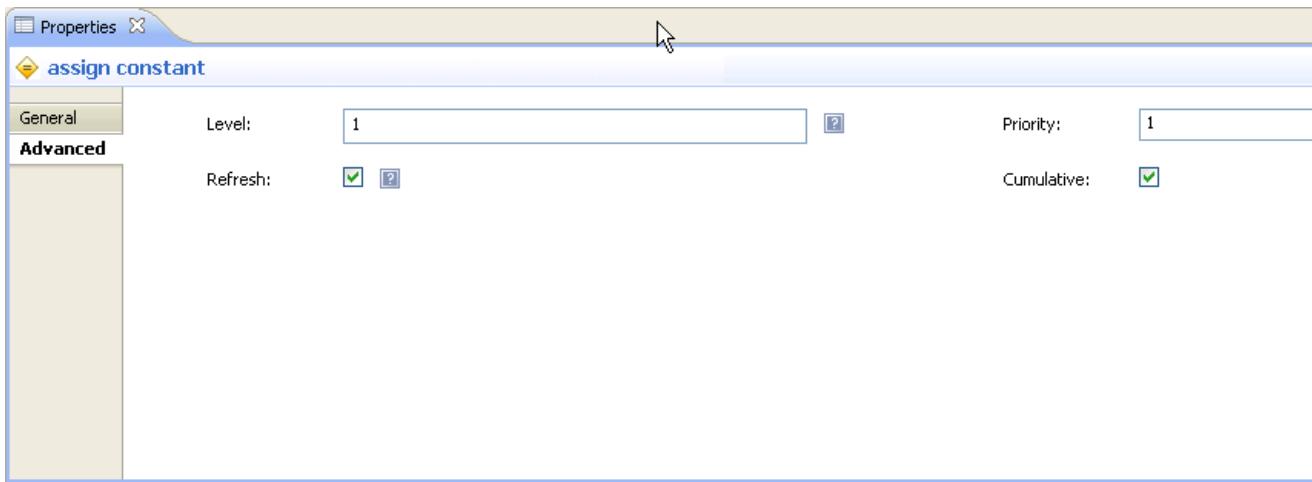
In this example, a constant value is assigned to a variable. Double click on the Cnt_Assign Constant Action container, the rulebase opens in a new editor.

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

Assign Action with General Properties



Assign Action Rule Advanced Properties

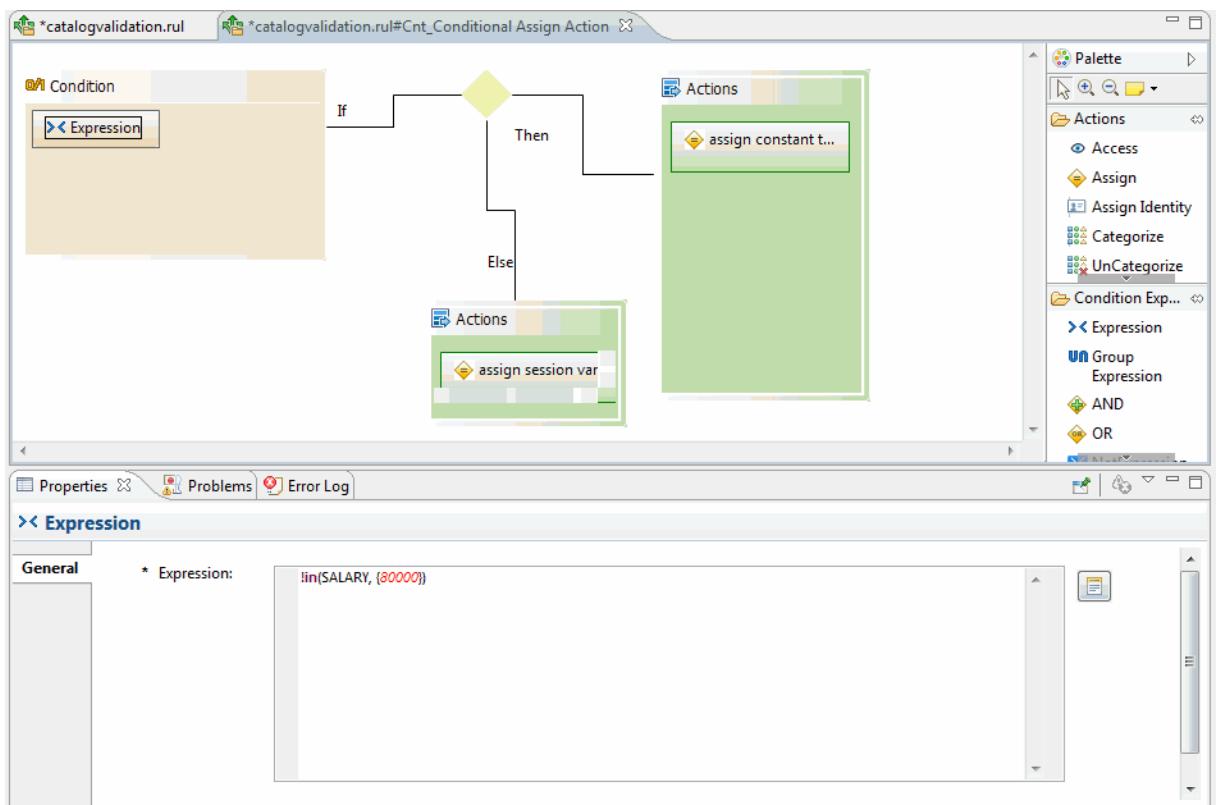


Assign Action Conditional

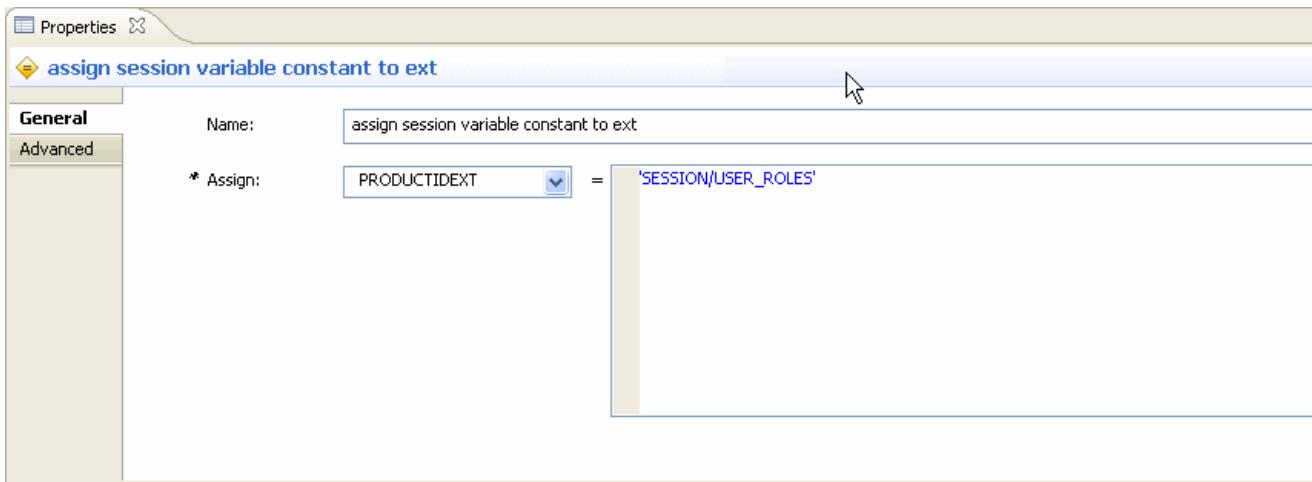
In this example a conditional value is assigned, if salary is other than 80000, then productidext is Manager else assign session variable constant.

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

Assign Action with Conditional General Properties



Assign Action with Else condition General Properties

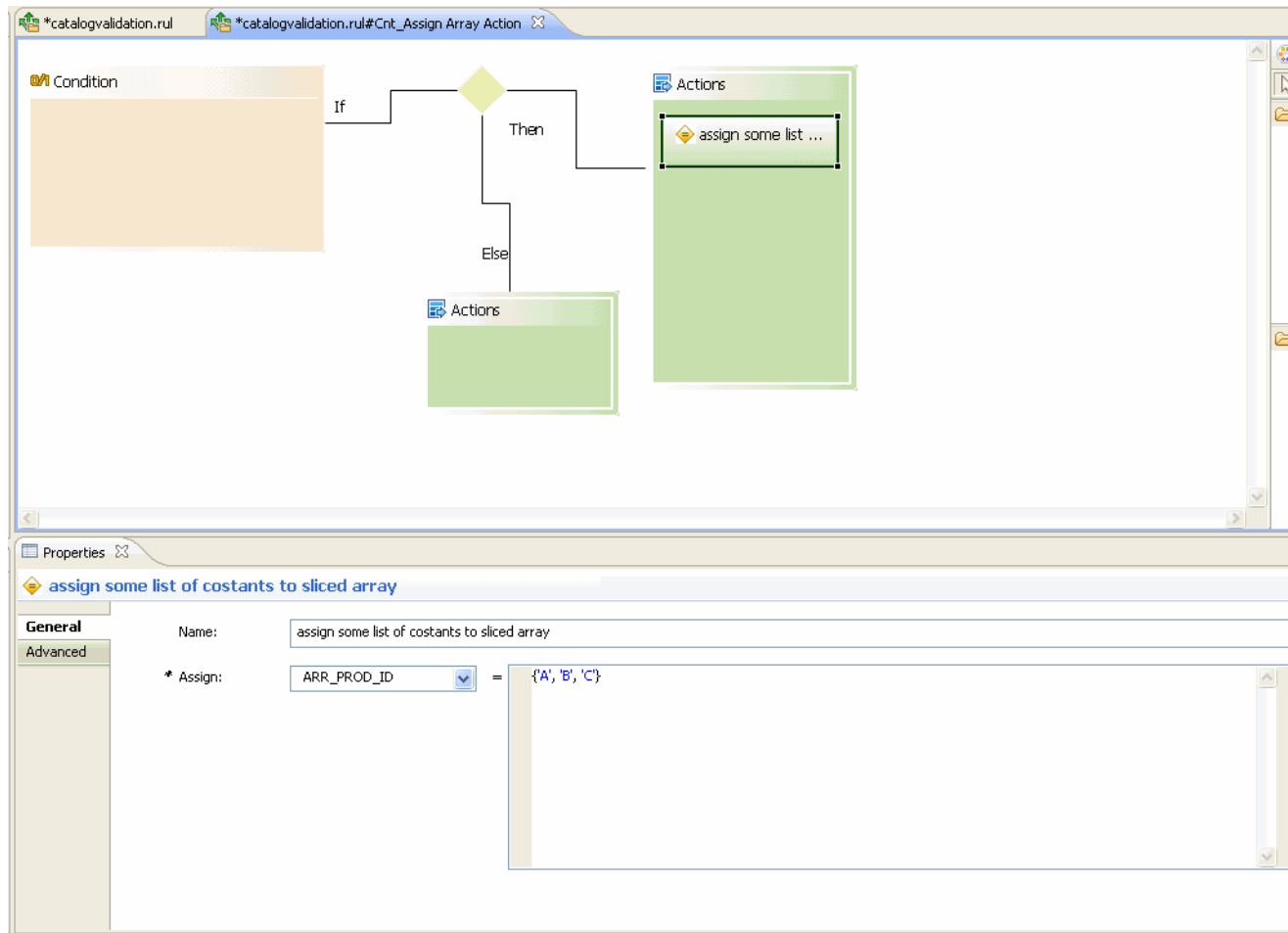


Assign Action

(Short Description to be added).

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

Assign Action with array with General Properties

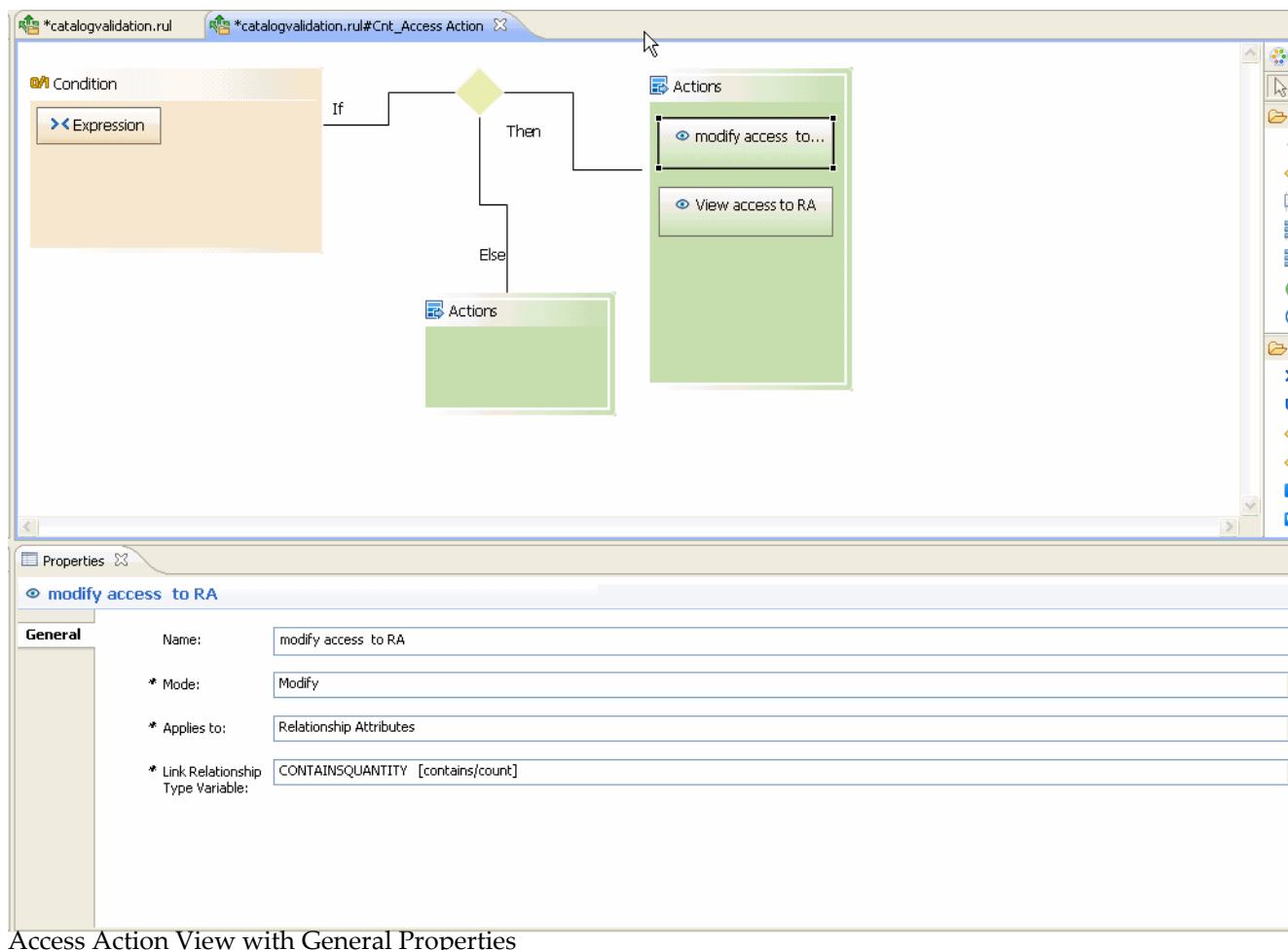


Access Action

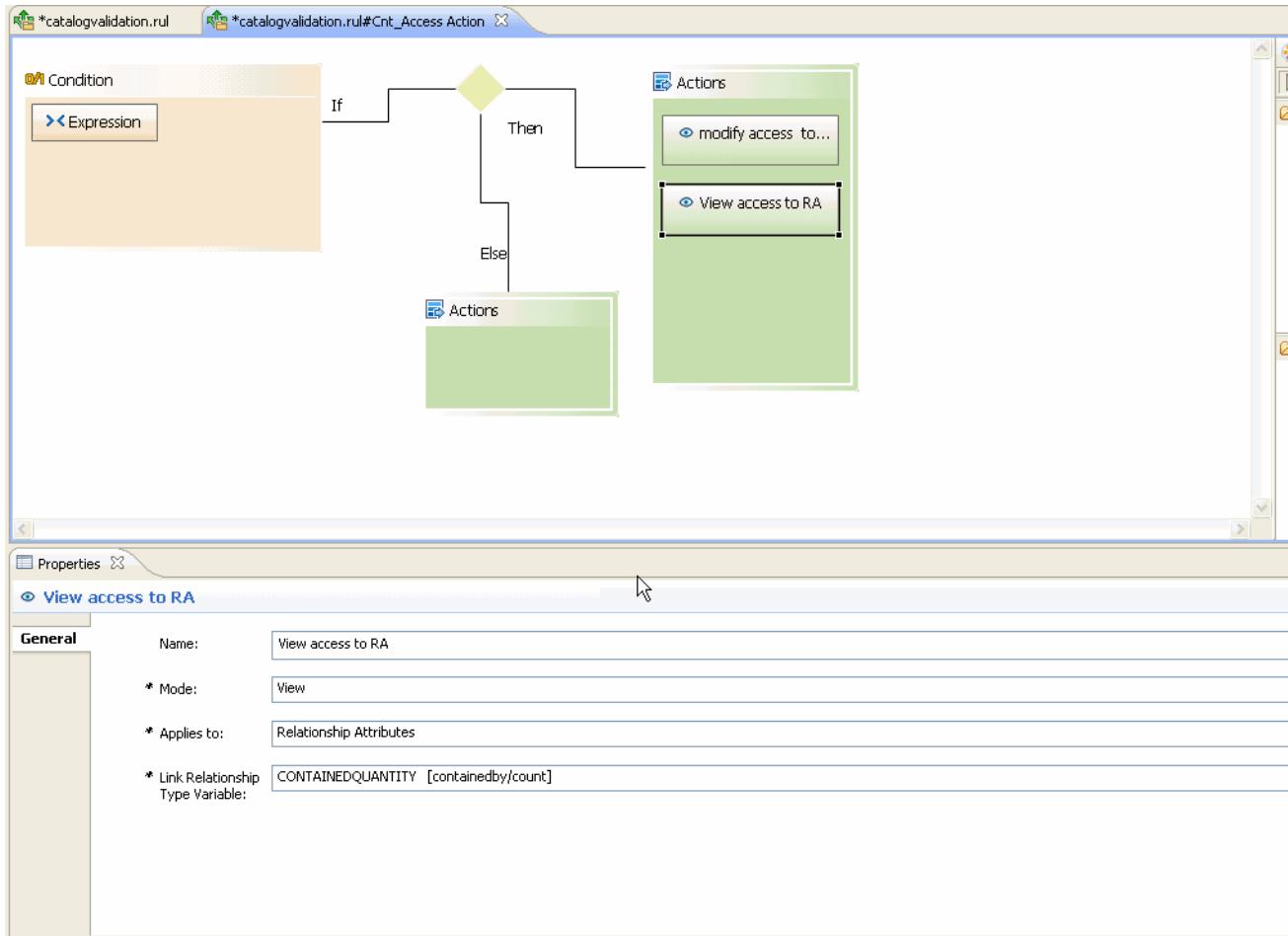
In this example a access action is defined.

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



Access Action View with General Properties

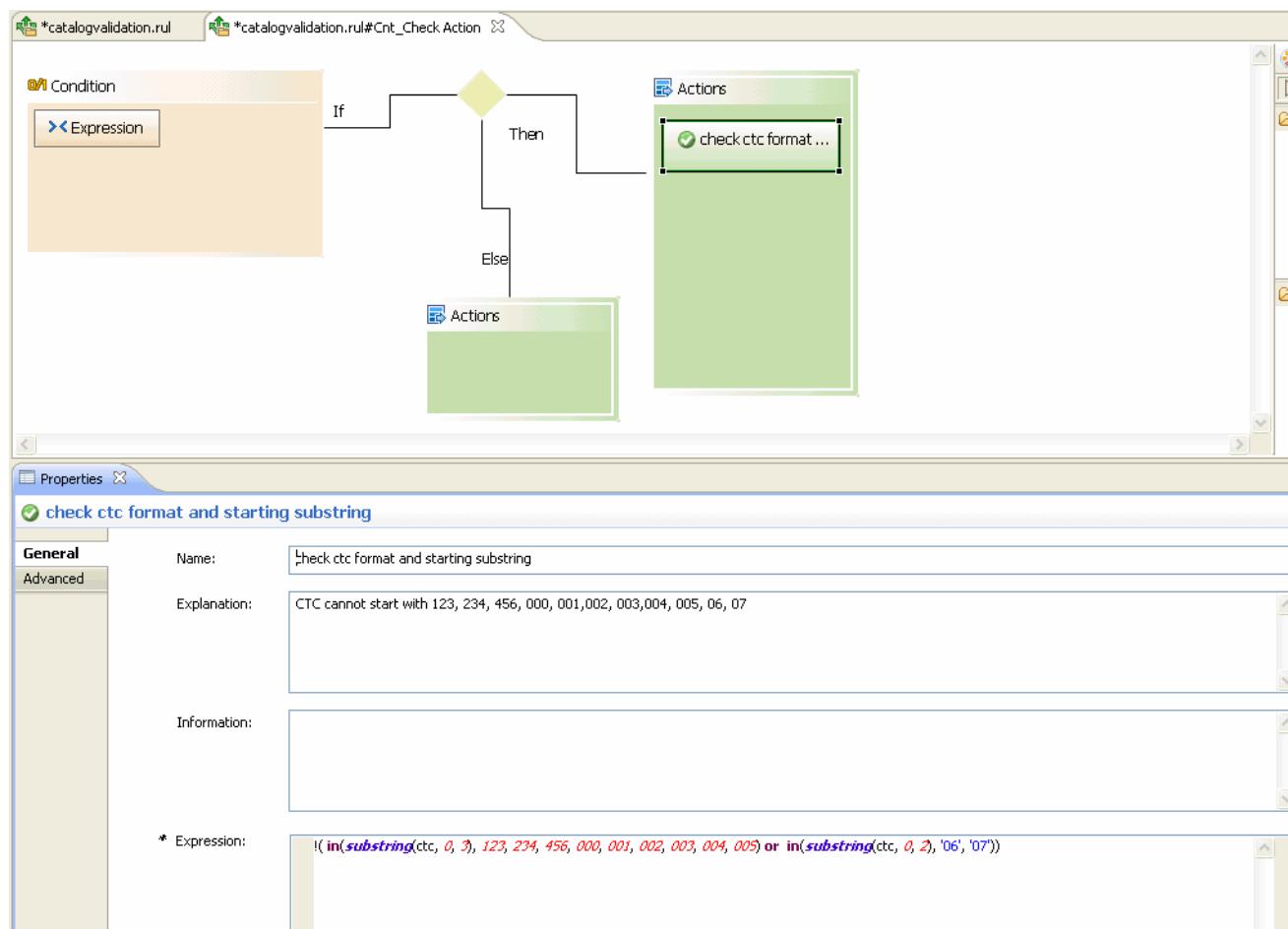


Check Action

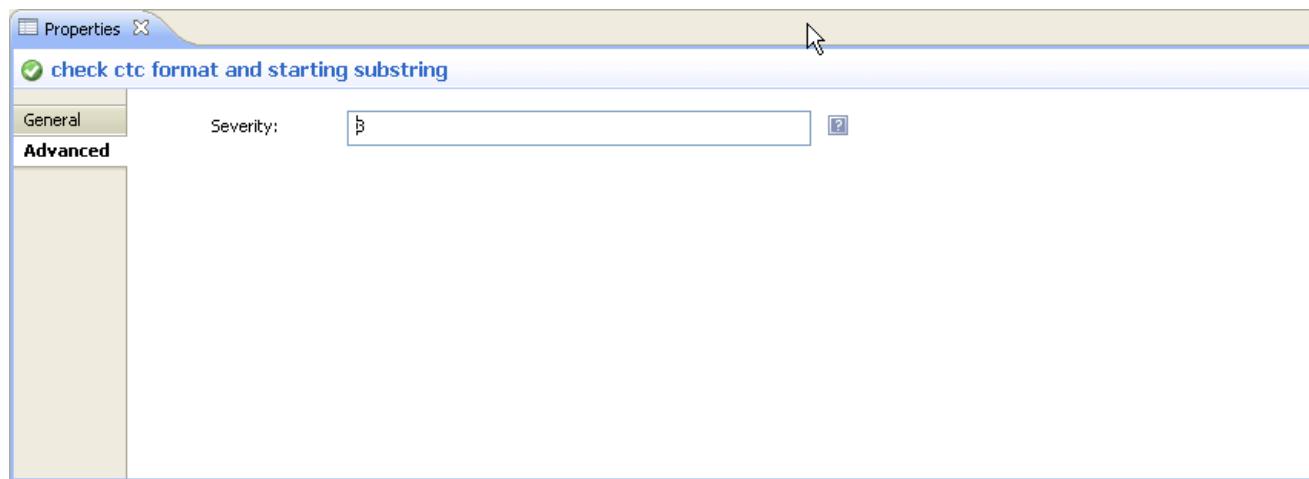
In this example, a check action is defined.

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

Check Action with General Properties



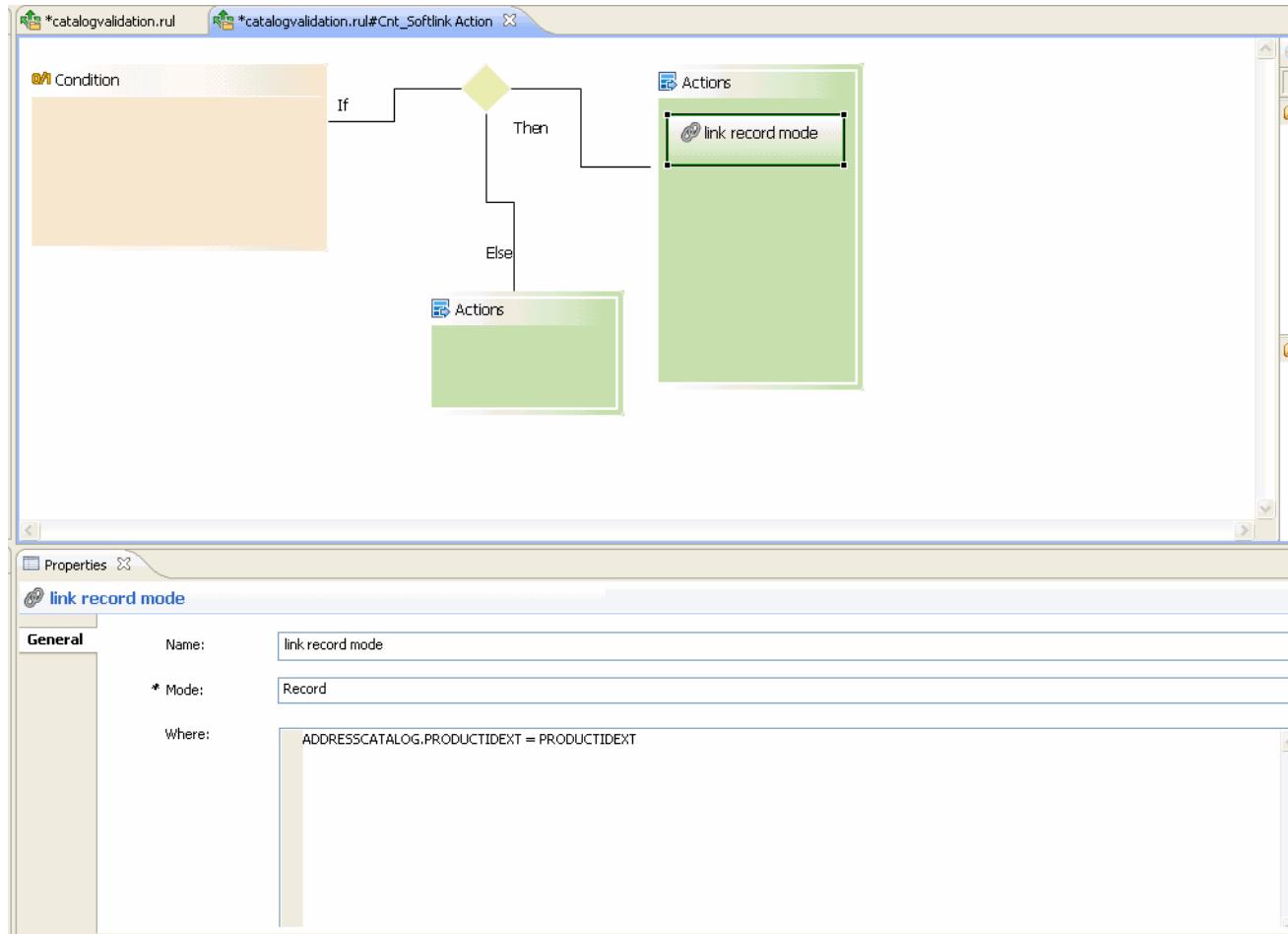
Check Action Rule Advanced Properties



Softlink Action

(Short Description to be added).

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

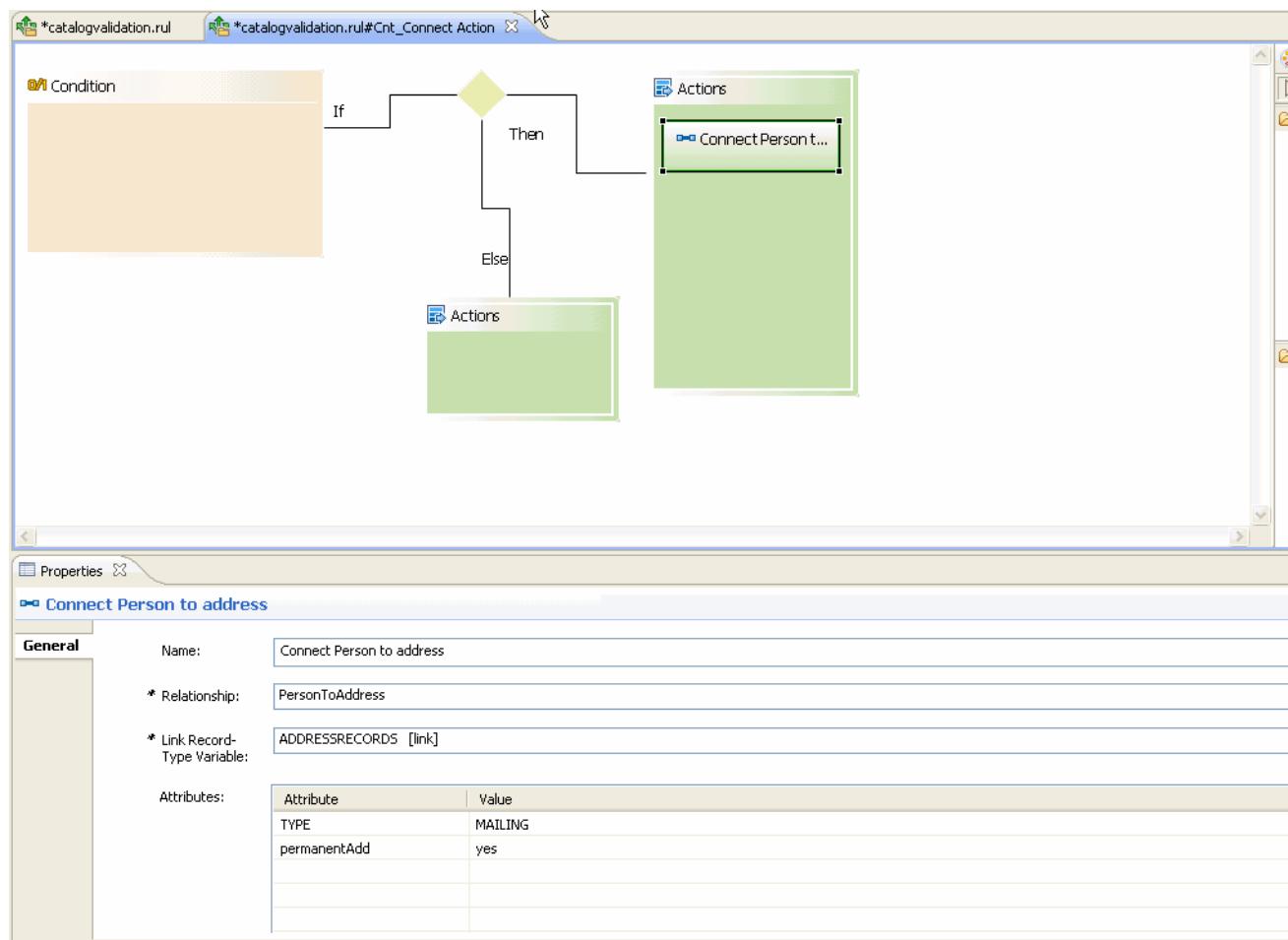


Connect Action

In this example, a connect action is defined.

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

Connect Action with General Properties

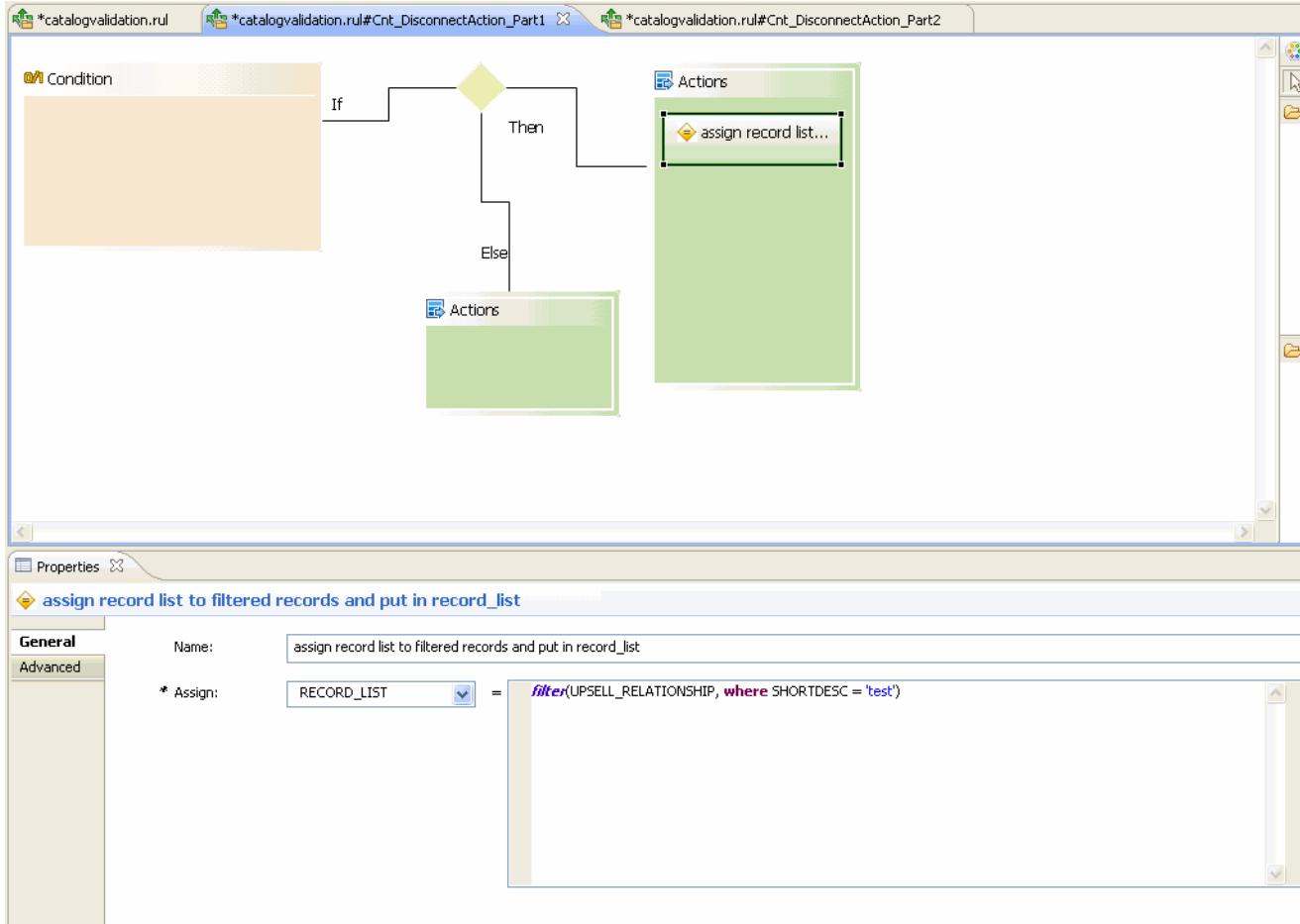


Disconnect Action

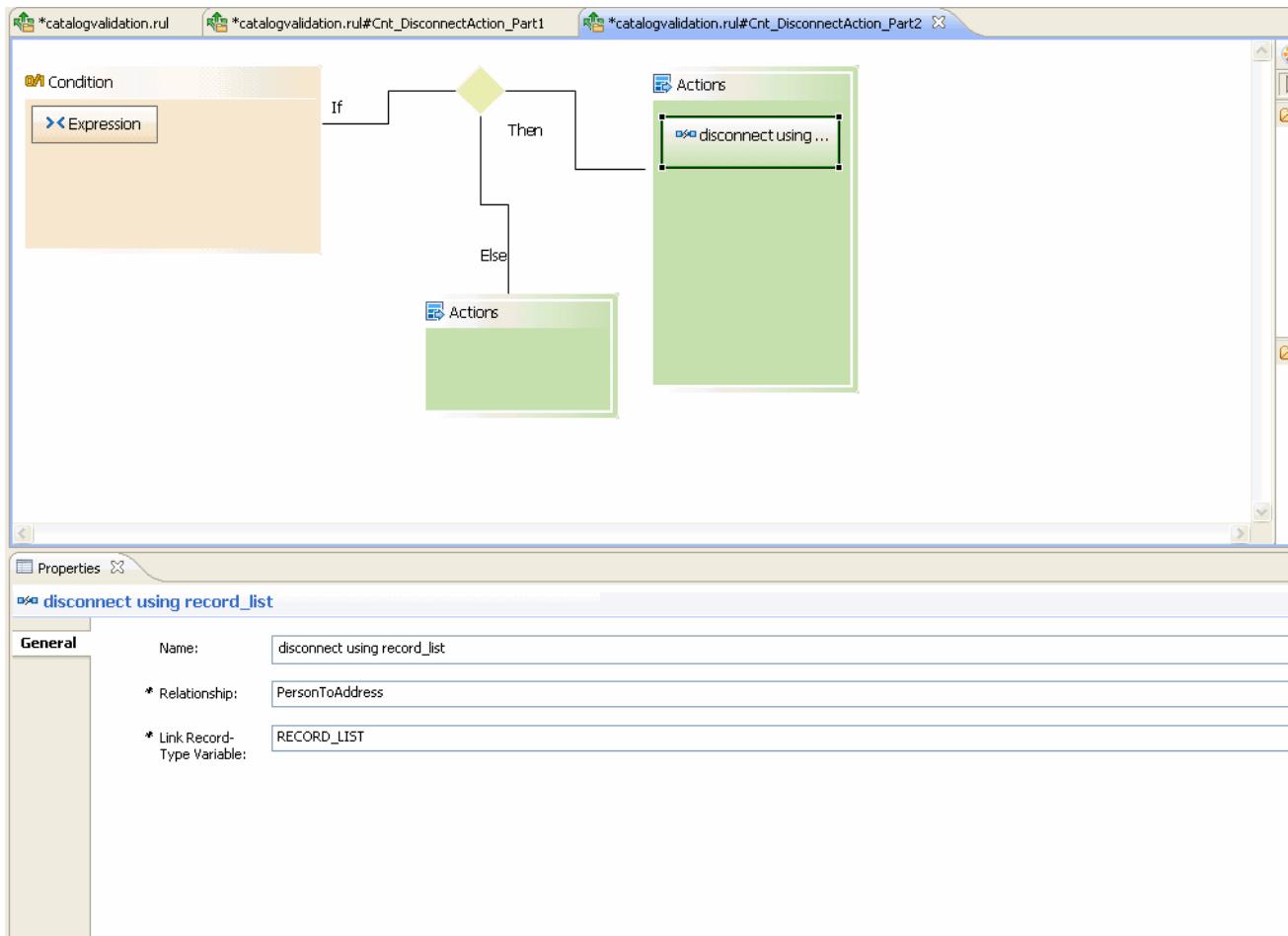
In this example, a disconnect action is defined.

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

Disconnect Action with assigned records with General Properties



Disconnect action using record list with General Properties

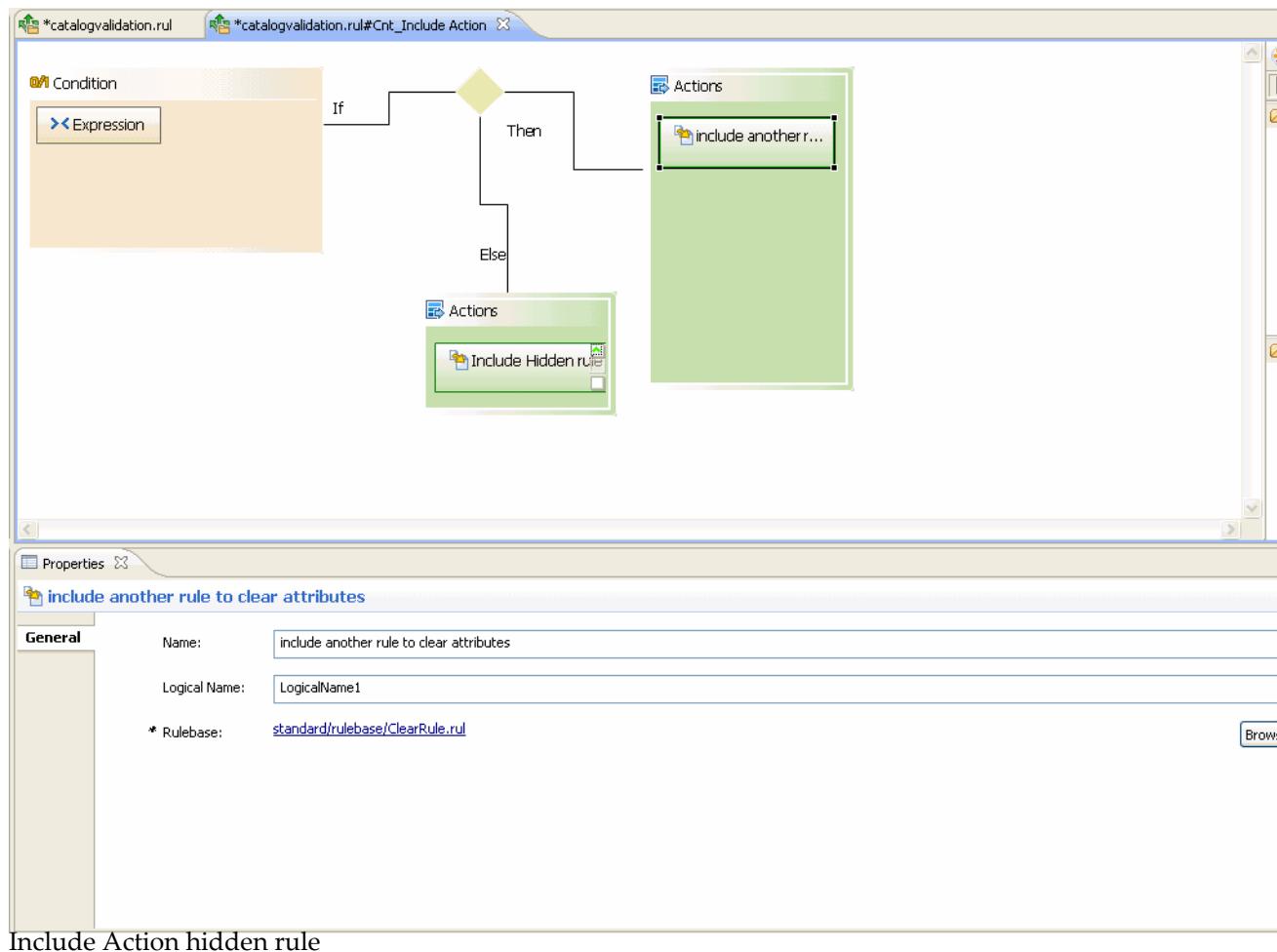


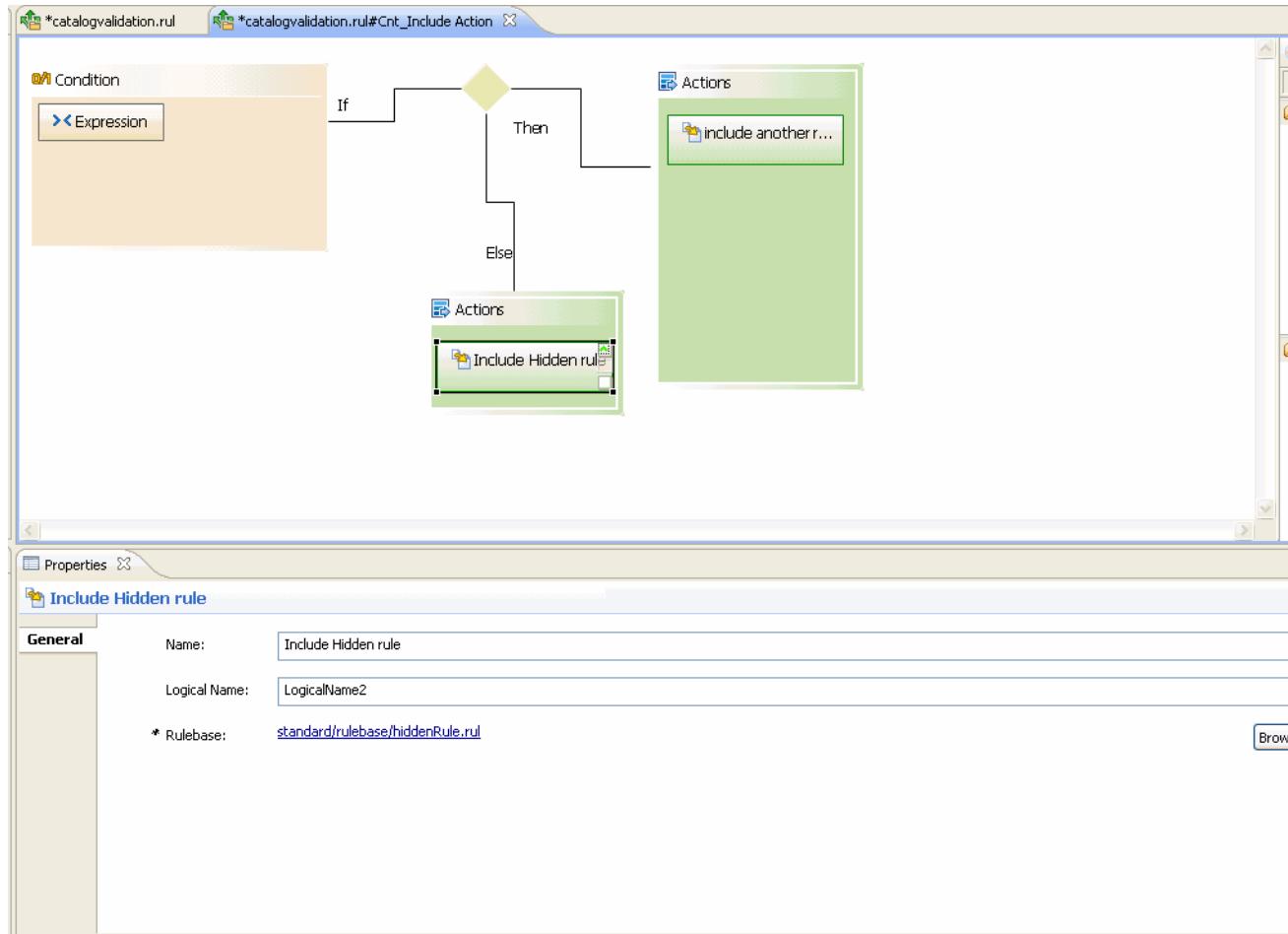
Include Action

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



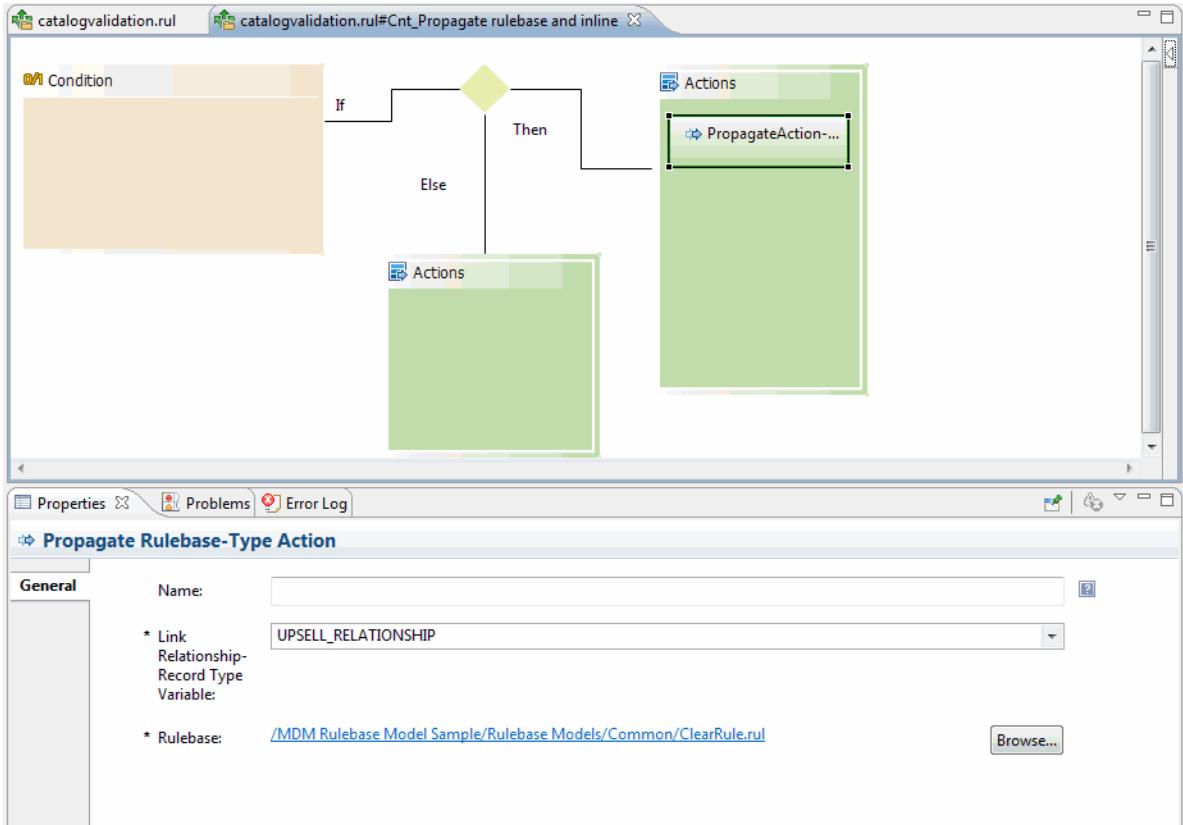


Propagate rulebase and inline Action

(Short Description to be added).

In this example, a propagate action is defined.

Propagate Action with General Properties



Select Action enum

(Short Description to be added).

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

Select Action with enum with General Properties

The screenshot shows the TIBCO MDM Studio Rulebase Designer interface. At the top, there are two tabs: 'catalogvalidation.rul' and 'catalogvalidation.rul#Cnt_Select Action with enum'. The main workspace displays a rule flow diagram. A 'Condition' node (orange) is connected to a decision diamond. The diamond has three outgoing branches: 'If' (top), 'Then' (right), and 'Else' (bottom). The 'Then' branch leads to an 'Actions' node (green), which contains a single action: 'select using enum'. The 'Else' branch leads to another 'Actions' node (green) which is currently empty. Below the workspace is a toolbar with icons for Properties, Problems, Error Log, Data Source Explorer, and Console. The 'Properties' tab is selected. A detailed view of the 'select using enum' action properties is shown in a central panel. The 'General' tab is selected, showing the following configuration:

- Name:** select using enum
- No value:** Option
- Show On Input:** (empty)
- Show Header:** checked
- Select Type:** Enum
- Column:** 0
- Use First Row As Header:** (unchecked)
- Attributes:** Literal
23.89
40.00
48.59

Below this, the 'Advanced' tab is also visible. The 'Properties' tab is also shown at the top of the properties panel.

Select Action Tables

(Short Description to be added).

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

Select Action with table datasource table

catalogvalidation.rul catalogvalidation.rul#Cnt_Select Action with table datasource and sql

Condition

If Then Else

Actions

Actions

Properties Problems Error Log Data Source Explorer Console

✓ select using table datasource

General

Name: select using table datasource * No value: Default

Show On Input: Show Header:

* Select Type: Table * Linktype Variable Source: DS1

* Source Type: Datasource

Type: Literal Constant Distinct:

Source Attributes

Select Action with SQL table

catalogvalidation.rul catalogvalidation.rul#Cnt_Select Action with table datasource and sql

Condition

If Then Else

Actions

Actions

Properties Problems Error Log Data Source Explorer Console

✓ select with table sql

General

Name: select with table sql * No value: Default

Show On Input: Show Header:

* Select Type: Table * Linktype Variable Source: COUNT(varlinkCatalogACCOUNT)

* Source Type: SQL

Type: Literal Constant Distinct:

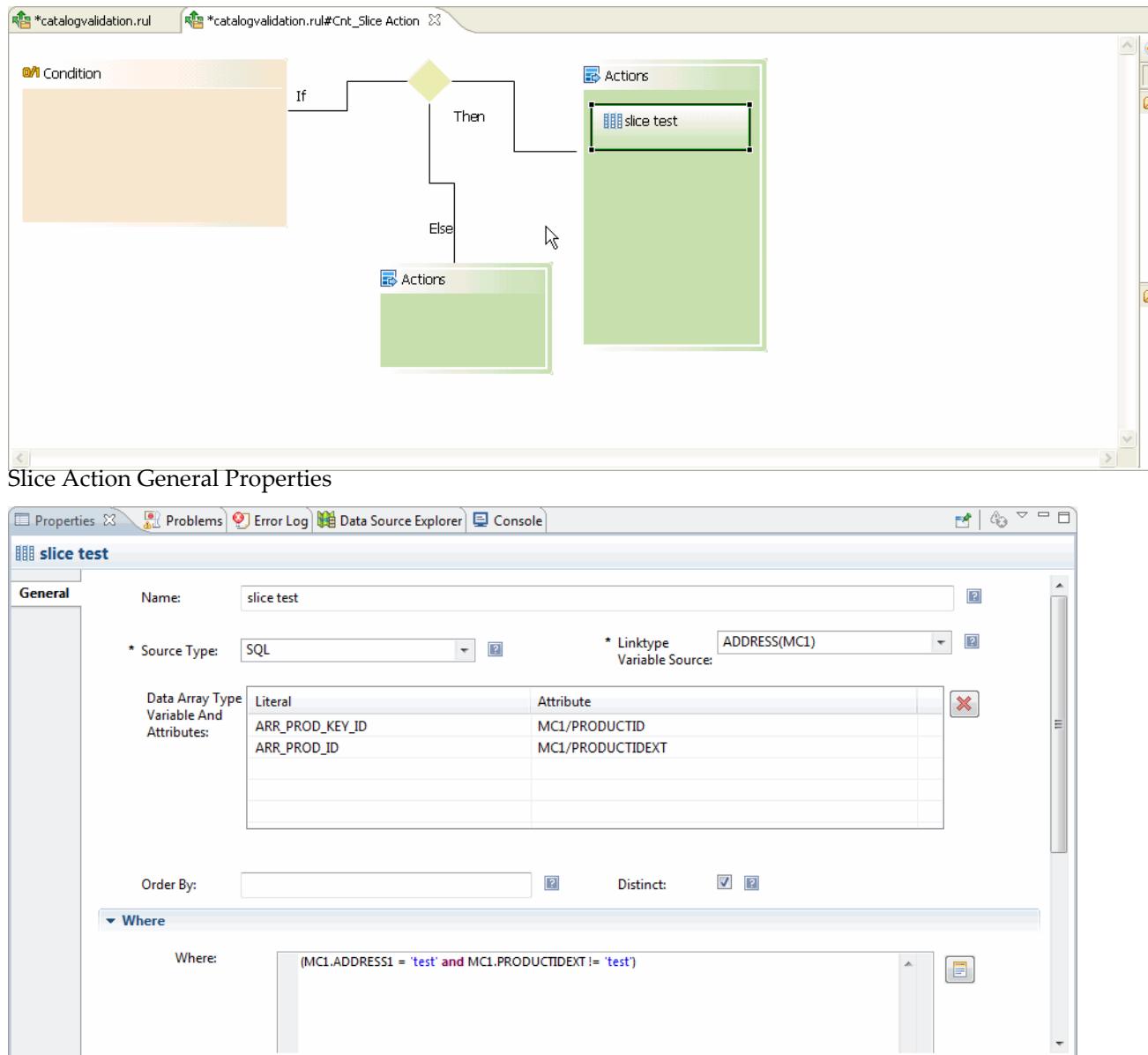
Source Attributes

Slice Action

(Short Description to be added).

In this example, a slice action is defined.

Slice Action



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

Repository : CUSTOMER

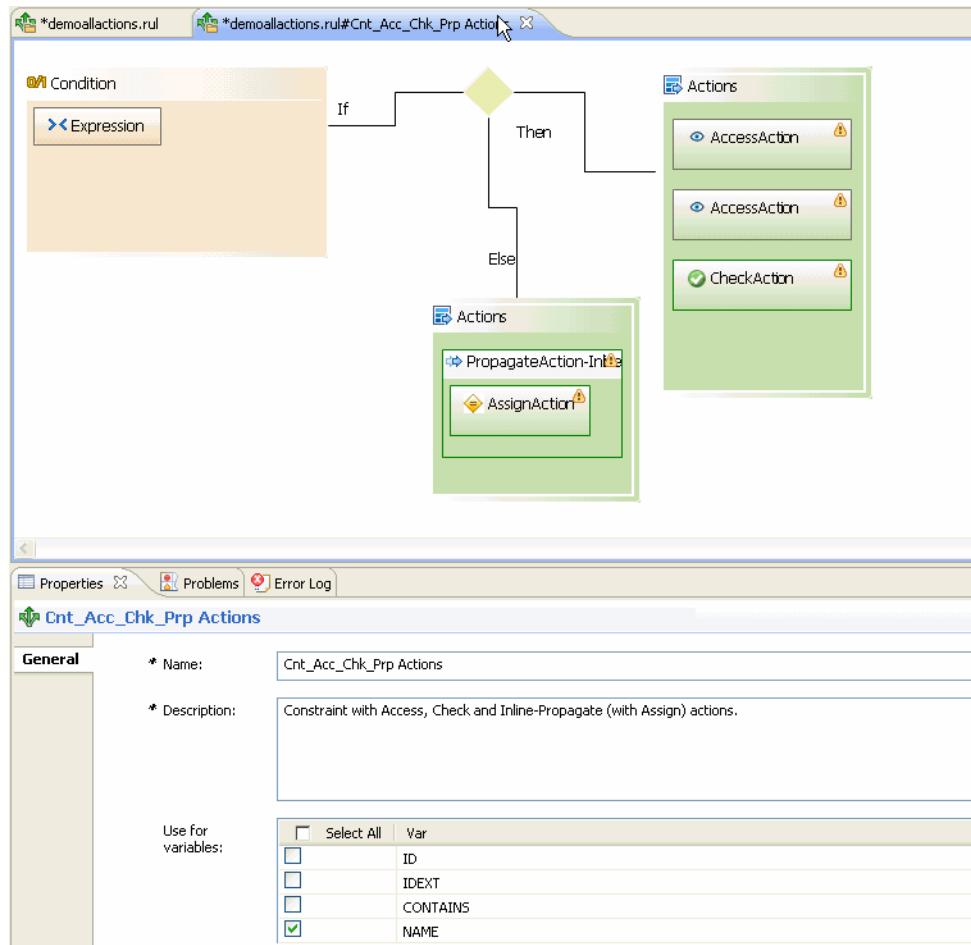


Constraint with Access Check and Inline-Propagate (with Assign) actions

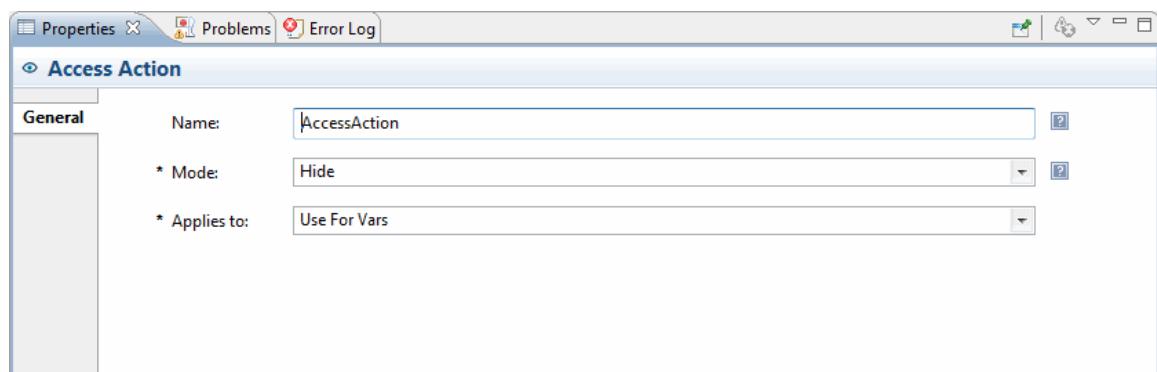
(Short Description to be added).

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

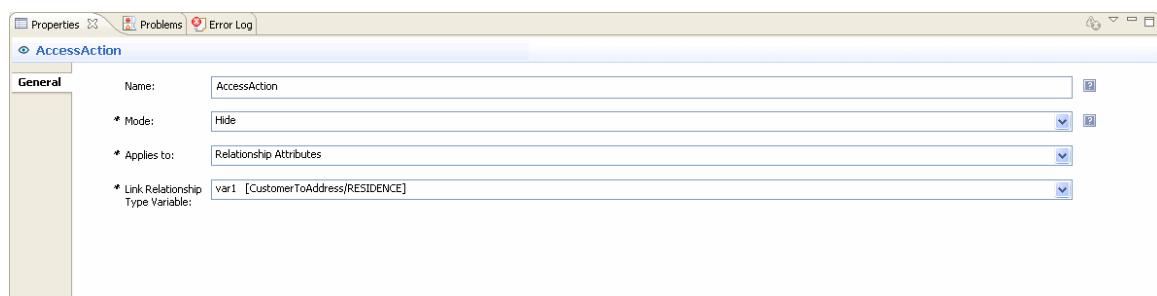
Constraint with Action, Check and Inline propagate action



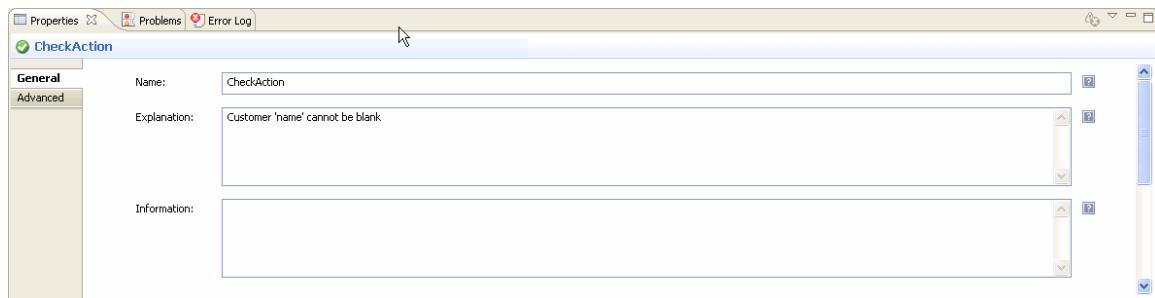
Access Properties



Access Action Properties



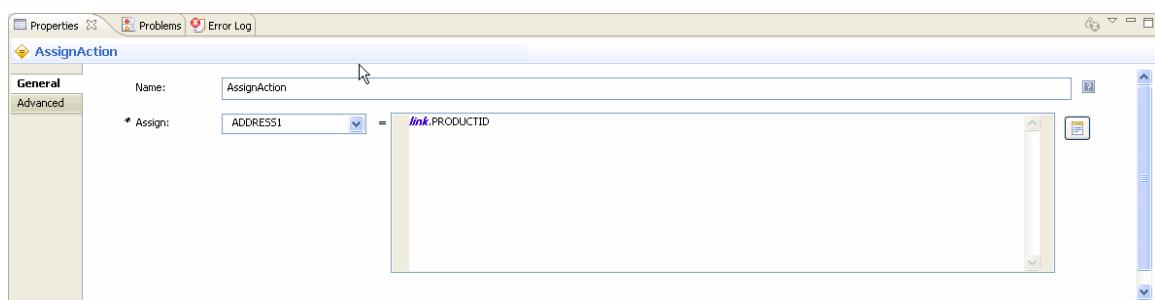
Check Action Properties



Propagate InlineAction Properties



Assign Action Properties

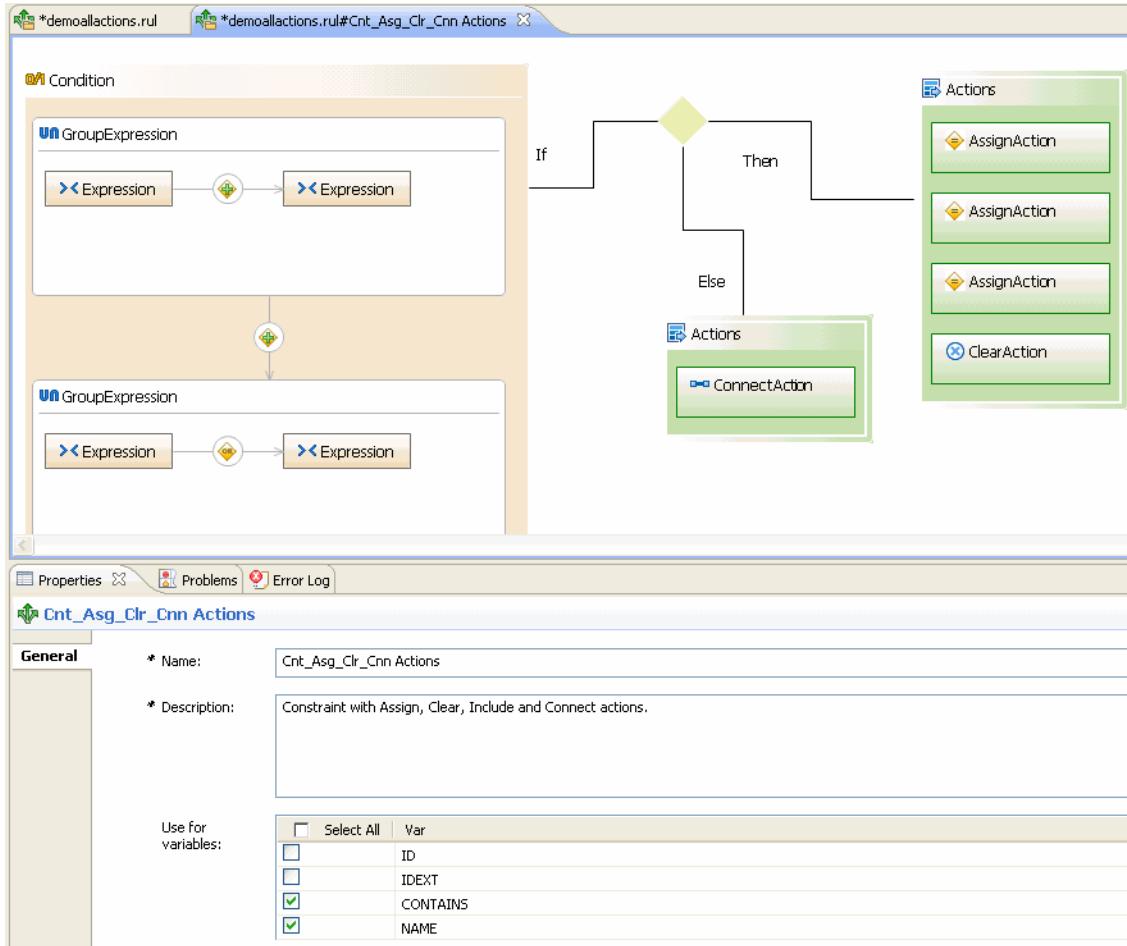


Constraint with Assign Clear Include and Connect actions

(Short Description to be added).

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

Constraint with Assign, Clear, Include and Connect actions

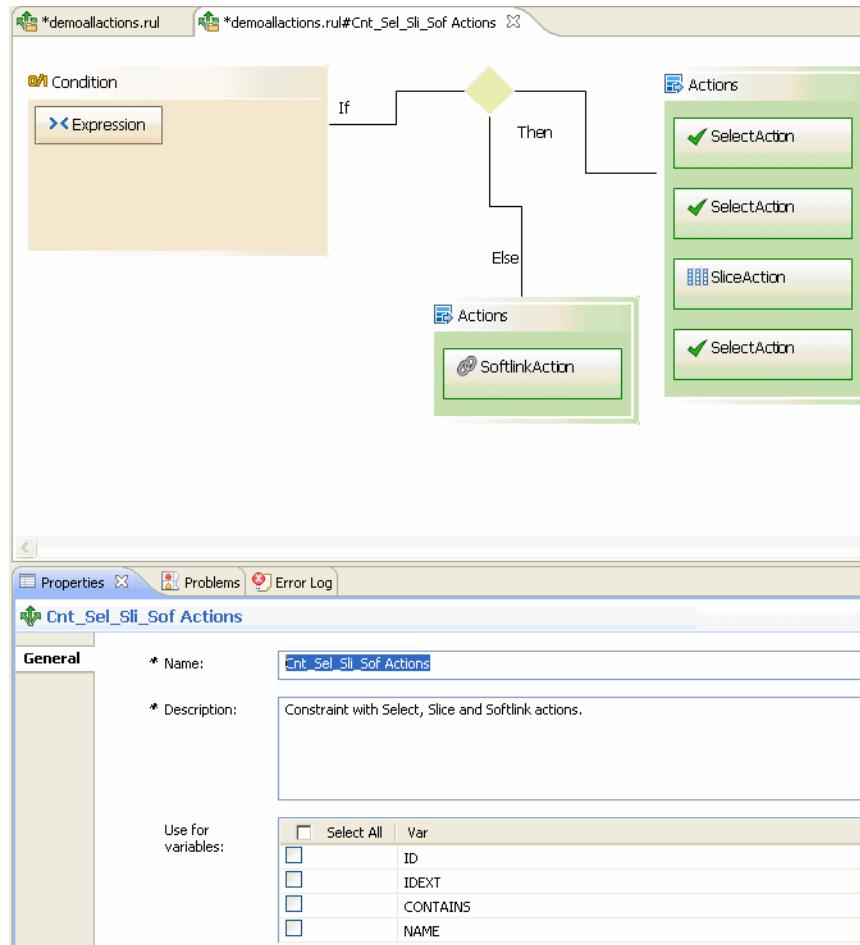


Constraint with Select Slice and Softlink actions

(Short Description to be added).

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

Constraint with Select, Slice and Softlink actions

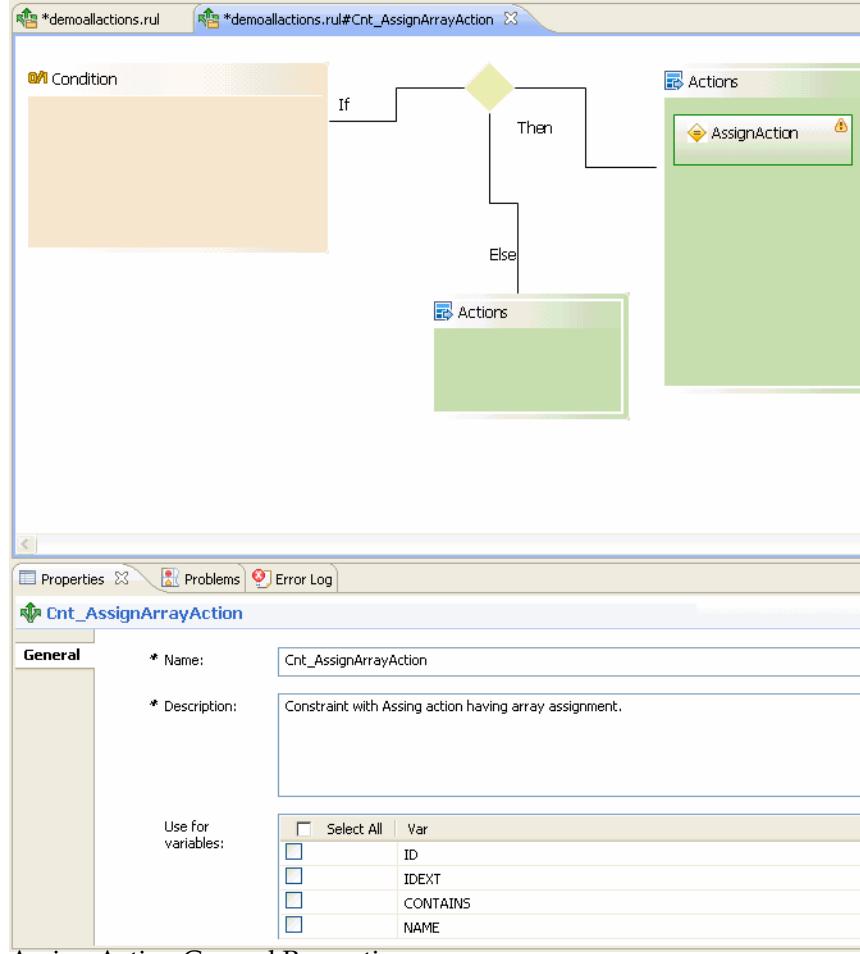


Constraint with Assign action having array assignment

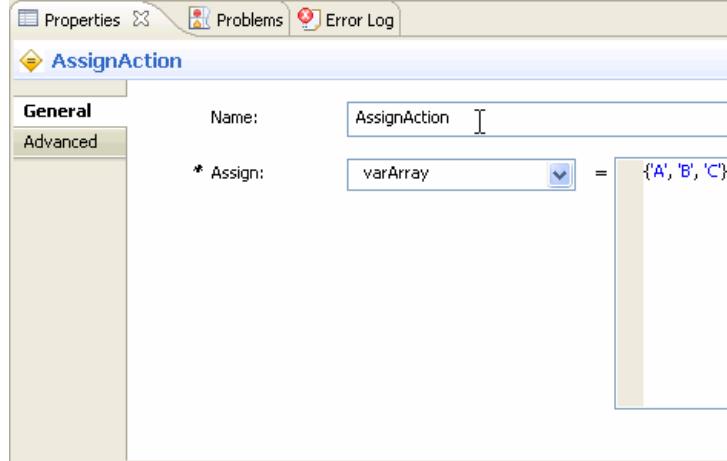
(Short Description to be added).

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

Constraint with assign action having array assignment



Assign Action General Properties

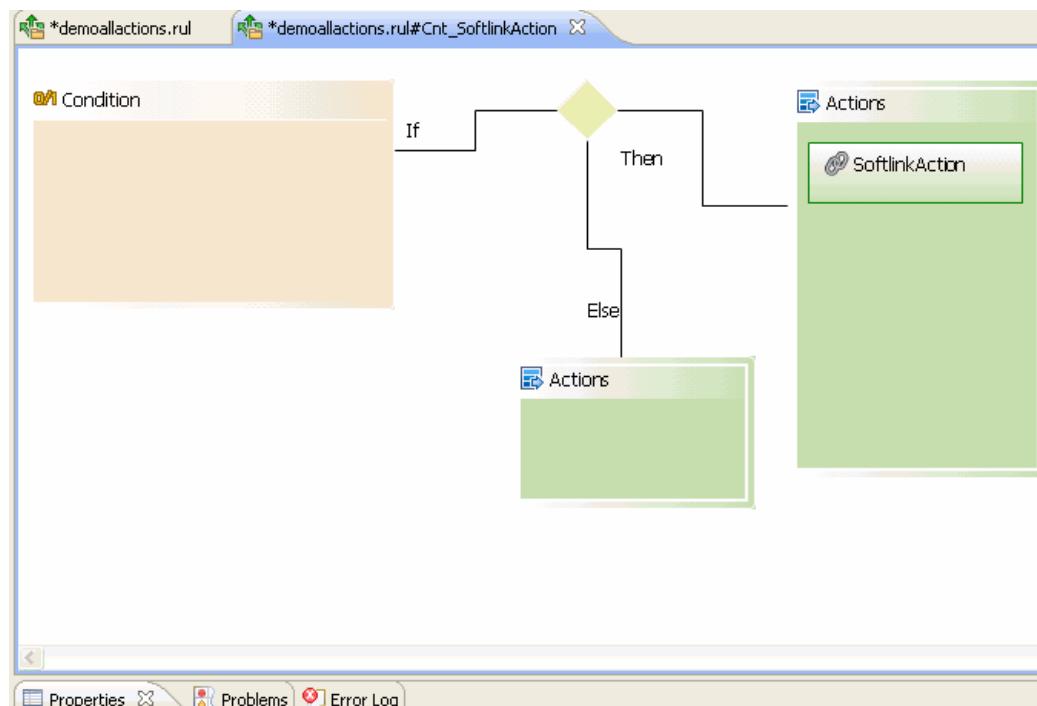


Constraint with Softlink action

(Short Description to be added).

In this example constraint with softlink action is described.

Constraint with Softlink Action



Properties Problems Error Log

Cnt_SoftlinkAction

General

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

Use for variables:

<input type="checkbox"/> Select All	Var
<input type="checkbox"/>	ID
<input type="checkbox"/>	IDEKT
<input type="checkbox"/>	CONTAINS
<input type="checkbox"/>	NAME

Softlink Action General Properties

Properties Problems Error Log

SoftlinkAction

General

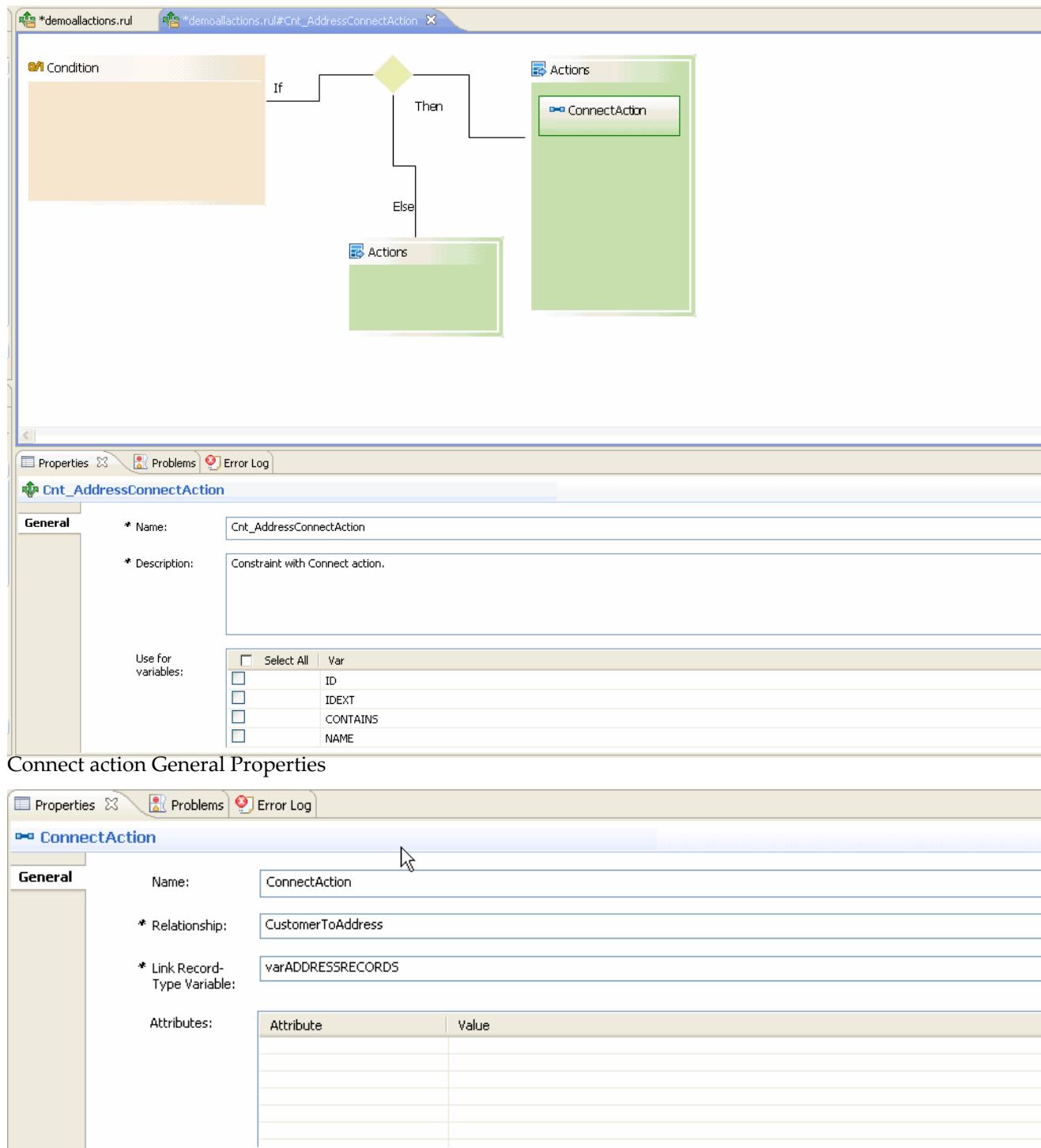
Name: SoftlinkAction
 * Mode: Record
 Where: varlinkCatalogCustomer.PRODUCTID = 'XXX'

Constraint with Connect action

(Short Description to be added).

In this example constraint with connect action is described.

Constraint with Connect action



Context Variables

(Short Description to be added).

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.
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.

Variable	Type	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>Set Language</name>
<description>Get Lanaguage from SESSION.</description>
<action>
<assign>
<var>language</var>
<var>SESSION/LANGUAGE</var>
</assign>
</action>
</constraint>
<constraint>
<name>Set Country</name>
<description>Get Country from SESSION.</description>
<action>
<assign>
<var>country</var>
<var>SESSION/COUNTRY</var>
</assign>
</action>
</constraint>
<constraint>
<name>Set Language Sel</name>
<description>Get ui language selected SESSION.</description>
<action>
<assign>
<var>uolang</var>
<var>SESSION/LANGSEL</var>
</assign>
</action>
</constraint>
```



After selecting language and country in User Accounts screen, the 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.
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 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.



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.



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.



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

(Short Description to be added).

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

Tips

(Short Description to be added).

- 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

(Short Description to be added).

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 classification scheme name.	the object of type classification scheme.	<pre><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></pre>

(Short Description to be added).

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.	<pre><op func="isRecordCategorizedUnderScheme"> <var>LINK_CSCHME</var> or any expression that evaluates to classification scheme </op></pre>

(Short Description to be added).

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). Child classificationCode: Child classification code string.	classification code object	<pre><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></pre>

(Short Description to be added).

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	<pre><op func="getClassificationCodeByName"> <var>LINK_CCODE</var> or any expression that evaluates to classification code <const type="string">codename</ const> <!-- the code NAME to retrieve --> </op></pre>

(Short Description to be added).

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	<pre><op func="getClassificationCodeForCodesInPath"> <var>LINK_CSCHEME</var> or any expression that evaluates to classification scheme <const type="string">rootcode/firstLevel/secondLevel/mycode</const> </op></pre>

(Short Description to be added).

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	<pre><op func="getClassificationCodeForCodeNamesInPath"> <var>LINK_CSCHEME</var> or any expression that evaluates to classification scheme <const type="string">rootcode/firstLevelCodeName/secondLevelCodeName/mycodeName</const> </op></pre>

(Short Description to be added).

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	<pre><op func="isRecordCategorizedUnderCodesPath"> <var>classificationScheme</var> <const type="string">rootcode/ firstLevel/secondLevel/ mycode</const> </op></pre>

(Short Description to be added).

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	<pre><op func="isRecordCategorizedUnderCodeNamesPath"> <var>classificationScheme</var> <const type="string">rootcode/ firstLevelCodeName/ secondLevelCodeName/ mycodeCodeName</const> </op></pre>

(Short Description to be added).

isRecordCategorizedUnderMultipleCodePaths

Description	Parameters	Returns	Example
Checks whether current record is categorized under all provided array of code paths	classificationScheme: the classification scheme link object arrayOfTreepathOfCode s:an array of full treepath of codes from the root code	true or false	<pre> <op func="isRecordCategorizedUnderMultipleCodePaths" > <var>classificationScheme</var> <const 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> </pre>

(Short Description to be added).

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	<pre> <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> </pre>

(Short Description to be added).

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.	<pre> <op func="getClassificationCodePathsForRecord"> </op> </pre>

(Short Description to be added).

getClassificationCodeNamePathsForRecord

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

(Short Description to be added).

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>

(Short Description to be added).

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>

(Short Description to be added).

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	<pre><op func="getClassificationCodeLevel"> <var>LINK_C_CODE</var> or any expression that evaluates to classification code </op></pre>

(Short Description to be added).

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	<pre><op func="isSubCategoryOfCode"> <var>LINK_CSCHHEME</var> or any expression that evaluates to classification scheme <const type="string">parentCode</const> <const type="string">childCode</const> </op></pre>

(Short Description to be added).

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	<pre><op func="isSubCategoryOfCodeName"> <var>LINK_CSCHHEME</var> or any expression that evaluates to classification scheme <const type="string">parentCodeName</const> <const type="string">childCodeName</const> </op></pre>

(Short Description to be added).

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 arrayOfTreepathOfCode: an array of full treepath of codes from the root code	array of classification code objects for provided treepaths	<pre><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></pre>

(Short Description to be added).

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	<pre><op func="stringTreepathOfCodeNamesToClassificationCode"> <const type="string">repositoryName</const> <const type="string">schemeName</const> <const type="string">rootcode/cat1/.... /codeName1</const> <const> <const type="string">rootcode/cat2/.... /codeName2</const> <const> <const type="string">rootcode/cat3/.... /codeName3</const> </op></pre>

(Short Description to be added).

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	<pre><op func="isRecordCategorized" d"> <var>LINK_C_SCHEME</ var> <var>LINK_C_CODE</ var> </op></pre>

(Short Description to be added).

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 classification 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.	<pre><op func="isRecordCategorizedUnderAll"> <var>LINK_C_SCHEME</var> <op func="stringTreepathOfCodesNamesToClassificationCode"> <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></pre>

(Short Description to be added).