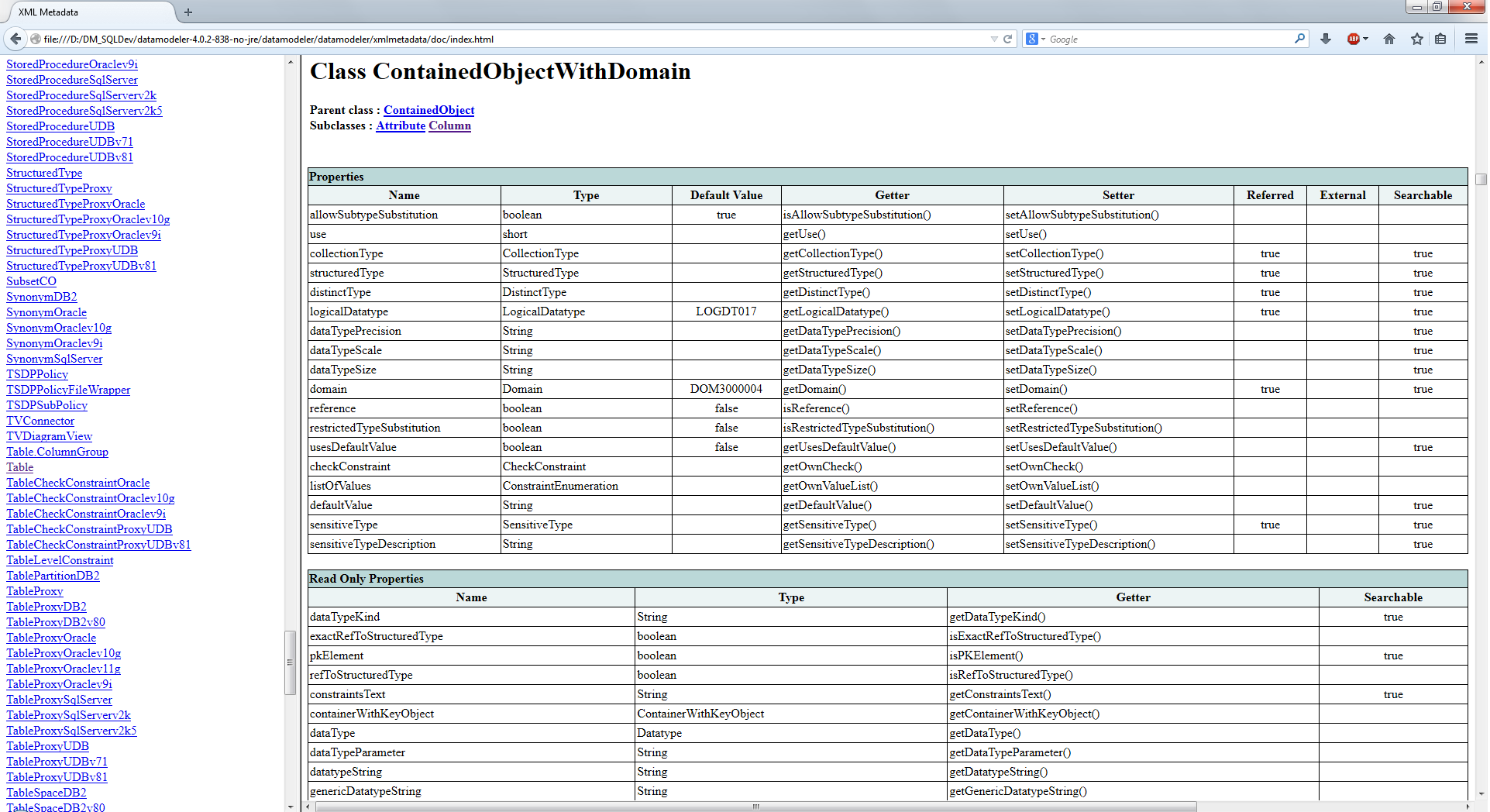
1. XML metadata files are part of the product

XML files in XML metadata directory ( datamodeler\datamodeler\xmlmetadata ) describe properties and structure of objects that can be created with Oracle SQL Developer Data Modeler UI or using scripting. They are used by Data Modeler (DM) in its functionality thus any changes introduced to them could break normal functionality of DM.

1. Information described in XML files

XML files follow internal object model used by DM and all properties of specific object can be found following parent class link. There is an HTML presentation of XML files in doc directory (use index.html) which allow faster navigation in class hierarchy – direct descendents of each class are also presented there.



Two important groups can be found in each XML file – properties and collections. They can be marked as external which means that property or collection is stored outside the file that represents the object. As example – Table has internal collection “columns” i.e. columns are stored in the file that represents table; RelationalDesign class (representing relational model) has external collection Tables. Design class (representing the whole design) has external property LogicalModel and external collection RelationalModels.

Each property has getter and setter method defined that can be used to get property or set it using scripting. There are simple and complex properties. Simple properties (data types String, boolean, int, long…) are parts of persistence of the object. Complex properties can be contained (their persistence is part of persistence of the object) or referred - ID of referred object is stored in objects persistence though object is involved in setter and getter method:

<property name="structuredType" dataType="oracle.dbtools.crest.model.design.datatypes.StructuredType" defaultValue="" xmlName="structuredType" relatedProperty="use" relatedValue="3" xmlType="element" getter="getStructuredType" setter="setStructuredType" reffered="true" search="true"/>

In some case simple String property holds object ID or list of object IDs – that’s related to dependencies between objects during loading – it’s possible that referred object(s) may not be loaded when that property is set.

Property can be marked as searchable and thus can be used in search, custom reports and set common properties functionality. Some read only properties also are marked as searchable.

Complex properties could have methods for creating them or nullifying them (example – listOfValues in ContainedObjectWithDomain) :

<property name="listOfValues" dataType="oracle.dbtools.crest.model.design.constraint.ConstraintEnumeration" isPropertyMap="true" xmlName="listOfValues" xmlType="element" getter="getOwnValueList" setter="setOwnValueList" createPMap="createConstraintEnumeration" itemXmlName="valueDef" keyXmlName="value" valueXmlName="description" removeItem="clearOwnValueList"/>

So listOfvalues can be created using instance of Column or Attribute – here is an example (script is bound to logical model, gets entity by name, gets attribute by name, creates list of values if it’s not set for attribute, set it on attribute and adds two values with description; :

ent = model.getEntitySet().getByName("PRODUCT INFORMATION");

if(ent!=null){

attr = ent.getElementByName("PRODUCT STATUS");

if(attr!=null){

attr.setUseDomainConstraints(false);

ce = attr.getOwnValueList();

if(ce==null){

ce = attr.createConstraintEnumeration();

attr.setOwnValueList(ce);

}

// clear existing properties if need that

//ce.clearProperties();

ce.add("aaa","desc aaa");

ce.add("bbb","desc bbb");

attr.setDirty(true);

}

}

Collection definition provide createItem method that can be used to create item of that collection using instance of surrounding object – back to Tables collection we see that table can be created using model.createTable(). And columns collection in Table shows the method for creating column:

col = table.createColumn();

ModelObject is the root of the hierarchy and each model is also instance of DesignPart class.

Changed object should be marked as changed using setDirty(true) method otherwise they won’t be saved during save operation.

Each collection with type xxxxxSet (EntitySet – logModel.getEntitySet(),TableSet – relModel.getTableSet()) allows objects to be found by name ( method getByName(name) ) and by object ID (method getObjectByID(objectID) ). Collections for tables and views in relational model also allows object to be found by schema and name using method getBySchemaAndName(schema, name). Method iterator() can be used to iterate over objects or toArray() to get them in array.

Each object has unique ID (method getObjectID() ) that can be used to get the object using method of Design instance – in scripting it’ll be model.getDesign().getDesignObject( ID ) - that doesn’t cover objects in physical models. Objects in relational and data types models have counterpart in physical model that has the same ID and can be found through instance of physical model (StorageDesign class) – for relational model – model.getStorageDesign().getStorageObject( ID ).

Tables in physical model are in TableProxySet – there is a special method for getting table by schema name and table name due specific implementation of shemaObject in relational model and user that implements it in physical model - getBySchemaImplAndName(String schema, String name) - model.getStorageDesign().getTableProxySet().getBySchemaImplAndName(schema, name)

1. Setting data type for column and attribute

Properties are described in ContainedObjectWithDomain.xml file.

<property name="use" dataType="short" defaultValue="" xmlName="use" xmlType="element" getter="getUse" setter="setUse" reffered="false" presentationName="Data Type Kind">

<codedEnum>

<item code="0" value="Domain"/>

<item code="1" value="Logical Type"/>

<item code="2" value="Distinct Type"/>

<item code="3" value="Structured Type"/>

<item code="4" value="Collection type"/>

</codedEnum>

</property>

<property name="collectionType" dataType="oracle.dbtools.crest.model.design.datatypes.CollectionType" defaultValue="" xmlName="collectionType" relatedProperty="use" relatedValue="4" xmlType="element" getter="getCollectionType" setter="setCollectionType" reffered="true" search="true"/>

<property name="structuredType" dataType="oracle.dbtools.crest.model.design.datatypes.StructuredType" defaultValue="" xmlName="structuredType" relatedProperty="use" relatedValue="3" xmlType="element" getter="getStructuredType" setter="setStructuredType" reffered="true" search="true"/>

<property name="distinctType" dataType="oracle.dbtools.crest.model.design.datatypes.DistinctType" defaultValue="" xmlName="distinctType" relatedProperty="use" relatedValue="2" xmlType="element" getter="getDistinctType" setter="setDistinctType" reffered="true" search="true"/>

<property name="logicalDatatype" dataType="oracle.dbtools.crest.model.design.LogicalDatatype" defaultValue="LOGDT017" xmlName="logicalDatatype" relatedProperty="use" relatedValue="1" xmlType="element" getter="getLogicalDatatype" setter="setLogicalDatatype" reffered="true" search="true"/>

<property name="dataTypePrecision" dataType="java.lang.String" defaultValue="" xmlName="dataTypePrecision" xmlType="element" getter="getDataTypePrecision" setter="setDataTypePrecision" search="true" reffered="false"/>

<property name="dataTypeScale" dataType="java.lang.String" defaultValue="" xmlName="dataTypeScale" xmlType="element" getter="getDataTypeScale" setter="setDataTypeScale" search="true" reffered="false"/>

<property name="dataTypeSize" dataType="java.lang.String" defaultValue="" xmlName="dataTypeSize" xmlType="element" getter="getDataTypeSize" setter="setDataTypeSize" search="true" reffered="false"/>

<property name="domain" dataType="oracle.dbtools.crest.model.design.Domain" defaultValue="DOM3000004" xmlName="domain" relatedProperty="use" relatedValue="0" xmlType="element" getter="getDomain" setter="setDomain" reffered="true" search="true"/>

Data type can be set using logical data type, domain, structured type, collection type or distinct type. Setting needs two steps 1) setting “**use**” property with related value as it’s described in meta data; 2) setting related type – example for logical data type:

column.setUse(1);

column. setLogicalDatatype(ldt);

Logical type, domain, structured type, ... can be get using related set and getByName() method:

lt\_varchar = model.getDesign().getLogicalDatatypeSet().getByName("VARCHAR");

domain = model.getDesign().getDomainSet().getByName("my\_domain");

Structured types set:

model.getDesign().getDataTypesDesign().getStructuredTypeSet()

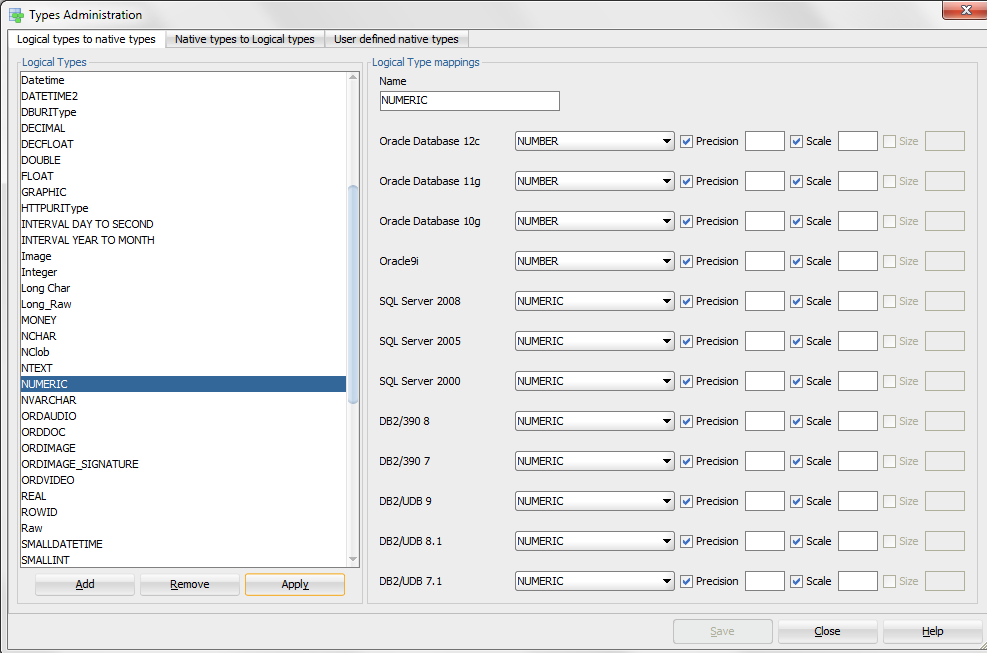
Collection types set:

model.getDesign().getDataTypesDesign().getCollectionTypeSet()

Distinct types set:

model.getDesign().getDataTypesDesign().getDistinctTypeSet()

Size or precision and scale can be set together with logical data type depending on data type presented – character or numeric. Size, precision and scale will be displayed on diagram and included in DDL only if they are enabled for logical data type set (Type administration)



1. Properties order of setting and additional processing

Properties are processed by DM in order they are in XML metadata. Dependency is defined in case one property need to be processed before other property, also processing can be defined after properties are set. As example FKIndexAssociation – defines properties for foreign keys:

<propertyDependency propDef="name" dependOnProperty="localFKIndex" />

<processing afterInit="addToTable" />

First definition defines that name of FK should be set after property localFK Index is set.

And when all properties are set then method fk.addToTable() will be invoked.

Well this describes how DM do processing when model is loaded. Otherwise here is an example how to create foreign key – Table2 has primary key PK\_Table2 and column Col1 (in table1) match PK column in Table2:

col1 = table1.getElementByName("Col1");

if(col1!=null){

list = new java.util.ArrayList();

list.add(col1);

fk = table1.addForeignKey(PK\_Table2,list);

fk.setName("fk name goes here");

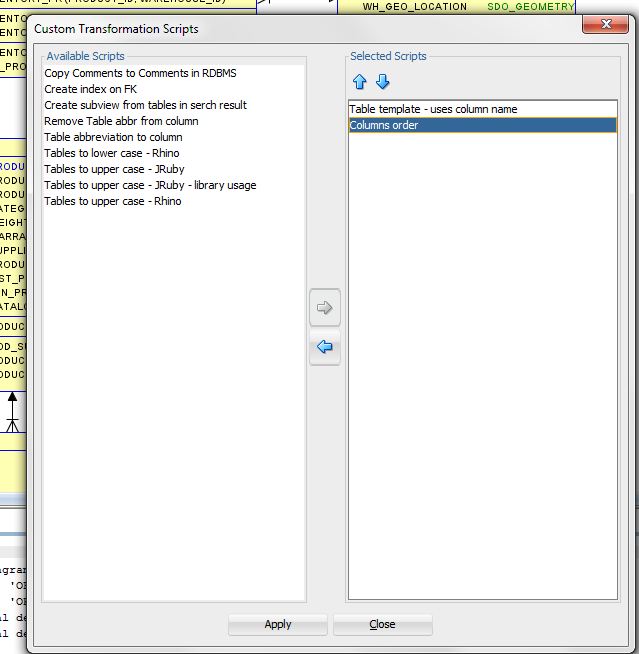
}

1. Transformation scripts

Transformation script can be created using “Tools>Design Rules>Transformations” and then new script can be created – need to set name, execution engine (Mozilla Rhino for Javascript comes with JDK), and model to which scripts is bound – logical or relational. Variable that is initialized with related model is named **model** . In its simplest form transformation script can be written in script window, save and executed or debugged (for Javascript only).

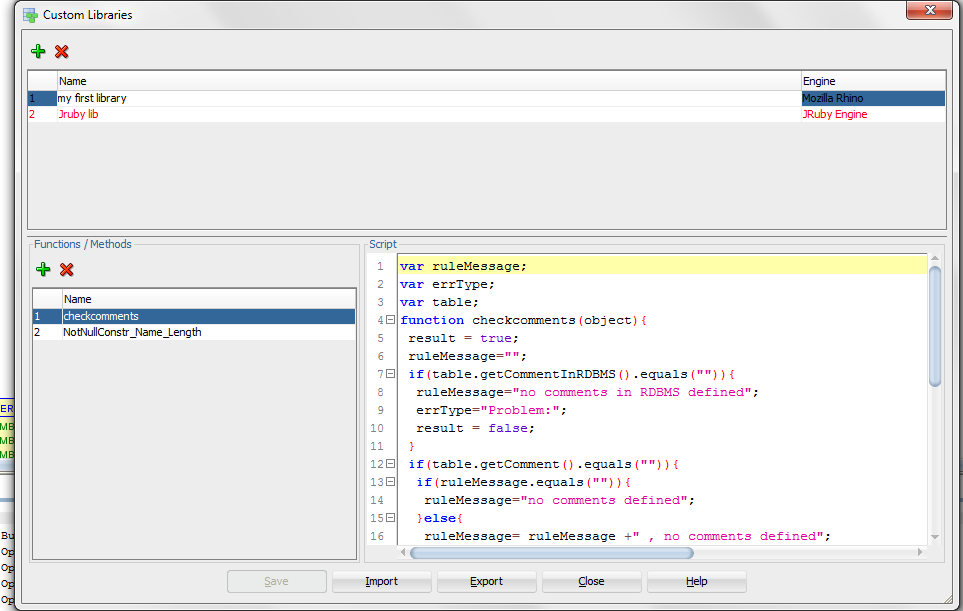
Changes made with the script cannot be undone, so it’s recommended that design to be saved before executing/debugging the script.

Once defined scripts can be executed from context menu for relational and logical model in the browser (“Apply transformation scripts” in pop-up menu) – one or mere scripts can be executed at once:



1. Code reuse - libraries

Libraries can be created in order code to be reused in different transformation scripts:

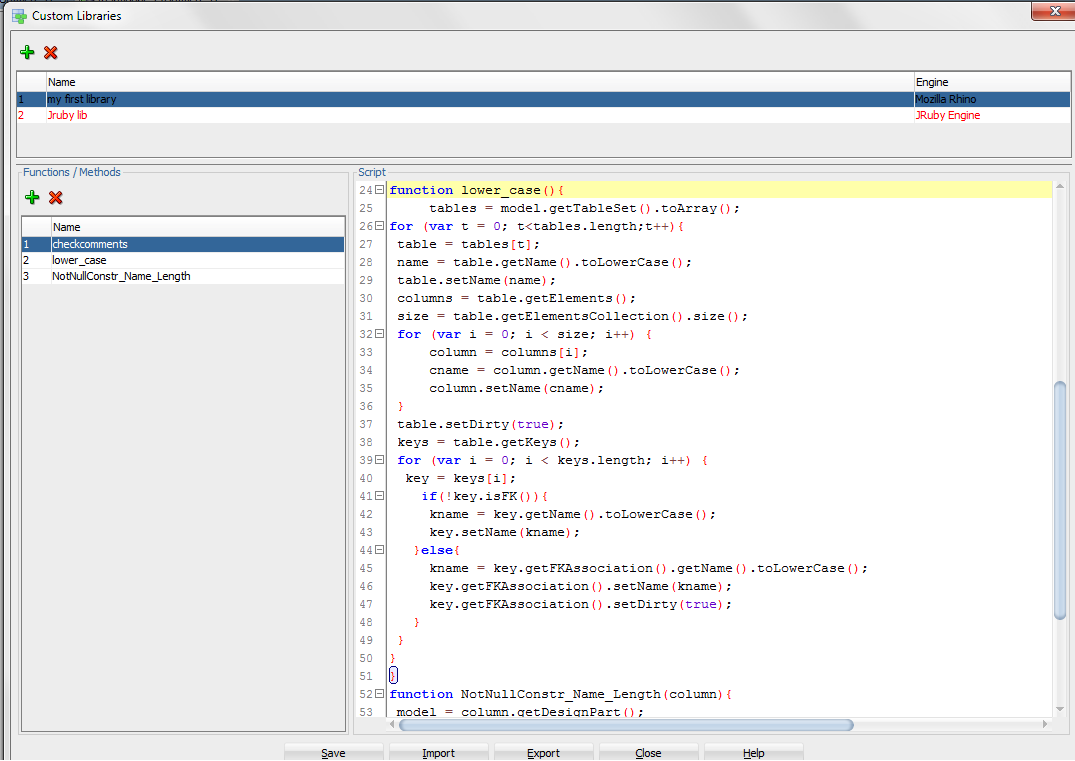


Library is defined with name, execution engine, list of functions that can be used in scripts or design rules and the code that defines functionality required.

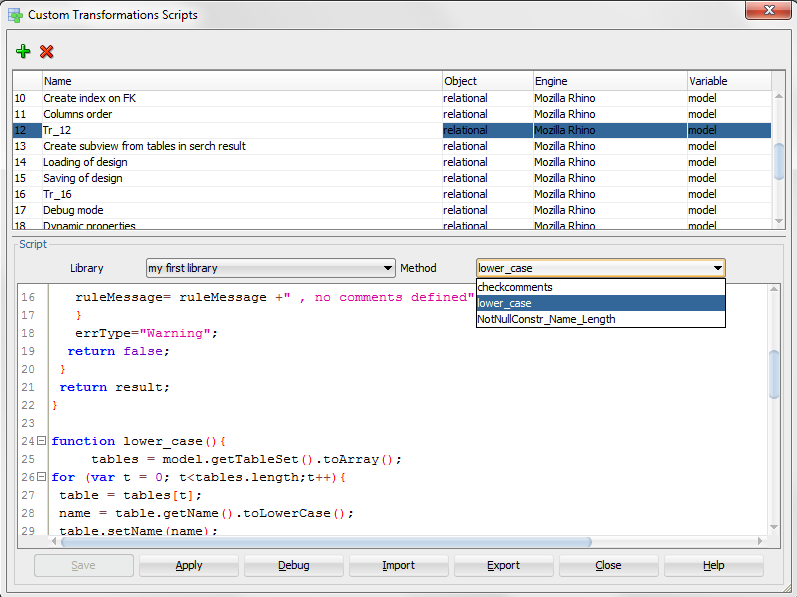
The code can be tested first in plain transformation script then can be moved in library for reuse. List of functions defines functions that can be used in definitions of transformation scripts and design rules. There is no parsing of the code so list of functions need to be created manually with names of functions that are defined in the code and can be used in transformations and rules.

1. Transformation script with library

DM comes with predefined scrip named “Tables to lower case – Rhino” – here is the code transformed to function in library – the code is surrounded by function definition



After code is transformed to function (lower\_case) in library and function is defined in list of functions we can create transformation script that uses library:

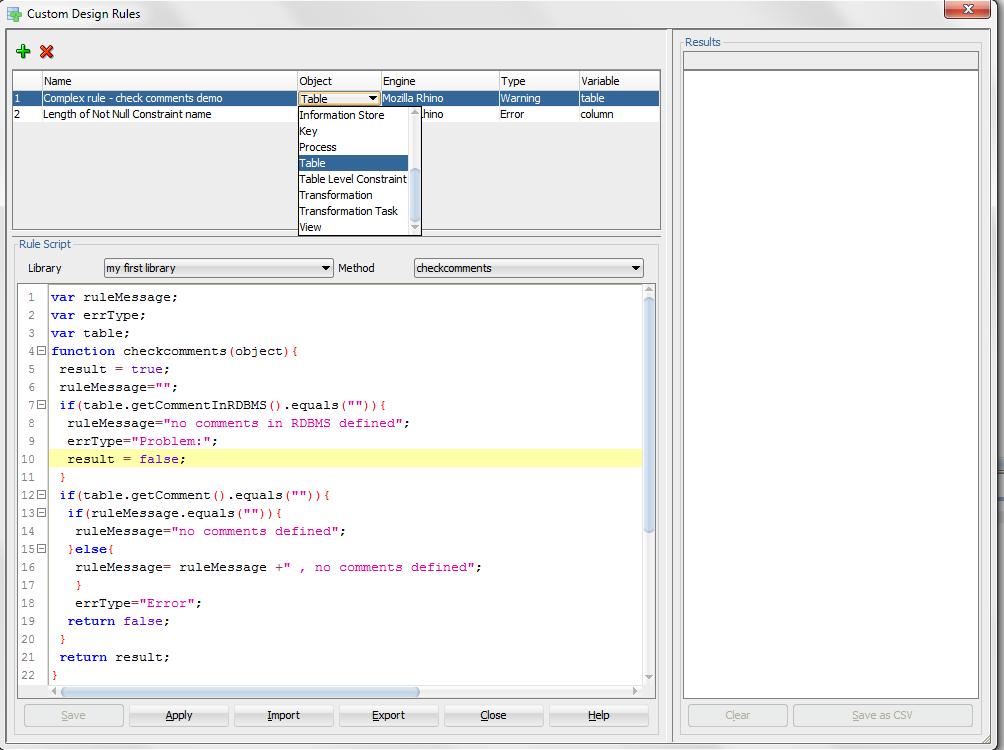


Library needs to be selected and function from library that will be executed. Library code appears in script window but it cannot be edited and debugged.

1. Custom Design rules

Difference between transformation and design rule – design rule is bound to specific type of object (table, column, entity..) and is invoked for each object in the related model while transformation is invoked only once. Design rule should return true if it’s valid for processed object or false otherwise. Design rules also return to DM error type (in variable errType) and error message (in variable errMessage). Error type could be Error (red color), Warning (blue color) or something else set by customer (objects will be colored in yellow). Design rule should not change the status if provided objects.

Following is defined for custom rule – name of the rule, object type, execution engine, error type (warning or error – this can be changed in the script trough text set in errType variable). Variable field shows the name of variable that will be initialized with provided object. Also library and function from that library need to be selected. Library code is shown in “rule script” window but it’s not editable as it’s with transformation script based on library.



1. Usage of result from search functionality

Objects appearing in search result are available using method getLastSearchResult() of model or physical model. Following code will cycle through all objects in search result:

objs = model.getLastSearchResult().toArray();

for(var i = 0;i<objs.length;i++){

obj = objs[i];

}

DM 4.0.2 comes with transformation that creates subview from table in search result:

objs = model.getLastSearchResult().toArray();

var dp;

var sv;

//create subview and add tables to it

for(var i = 0;i<objs.length;i++){

obj = objs[i];

// use "Entity" for entities and logical model

if(obj.getObjectTypeName().equals("Table")){

if(dp==null){

dp = obj.getDesignPart();

sv = dp.createDesignPartSubView();

// uncomment next line if want to set name

//sv.setName("Name");

sv.getPlaceHolder().setVisible(true);

}

sv.addViewFor(obj);

}

}

if(dp!=null){

//add foreign keys if you want to

for(var i = 0;i<objs.length;i++){

obj = objs[i];

// use "Entity" for entities and logical model

if(obj.getObjectTypeName().equals("Table")){

tv = obj.getFirstViewForDPV(sv);

if(tv!=null){

//for entities and logical model use

//tv.addTVRelations();

tv.addTVFKRelations();

}

}

}

//add Arcs if you want to

for(var i = 0;i<objs.length;i++){

obj = objs[i];

if(obj.getObjectTypeName().equals("Table")){

tv = obj.getFirstViewForDPV(sv);

if(tv!=null){

tv.addArcs();

}

}

}

//arrange diagram

sv.rearrangeNewDiagram();

}

1. Objects in Subview

Subview is instance of Class DesignPartView and its descendants. Each table, view (class TableView), entity, view in logical model (class EntityView), foreign key (class FKIndexAssociation) and relationship (class Relation) has method *getFirstViewForDPV(dpv)*

where dpv is existing subview – here is code that will process only tables in specific subview:

//get subview by name

dpv = model.getSubViewByName(" the name of subview is here");

if(dpv!=null){

tables = model.getTableSet().toArray();

for (var t = 0; t<tables.length;t++){

table = tables[t];

//check for presentation in subview

tv = table.getFirstViewForDPV(dpv);

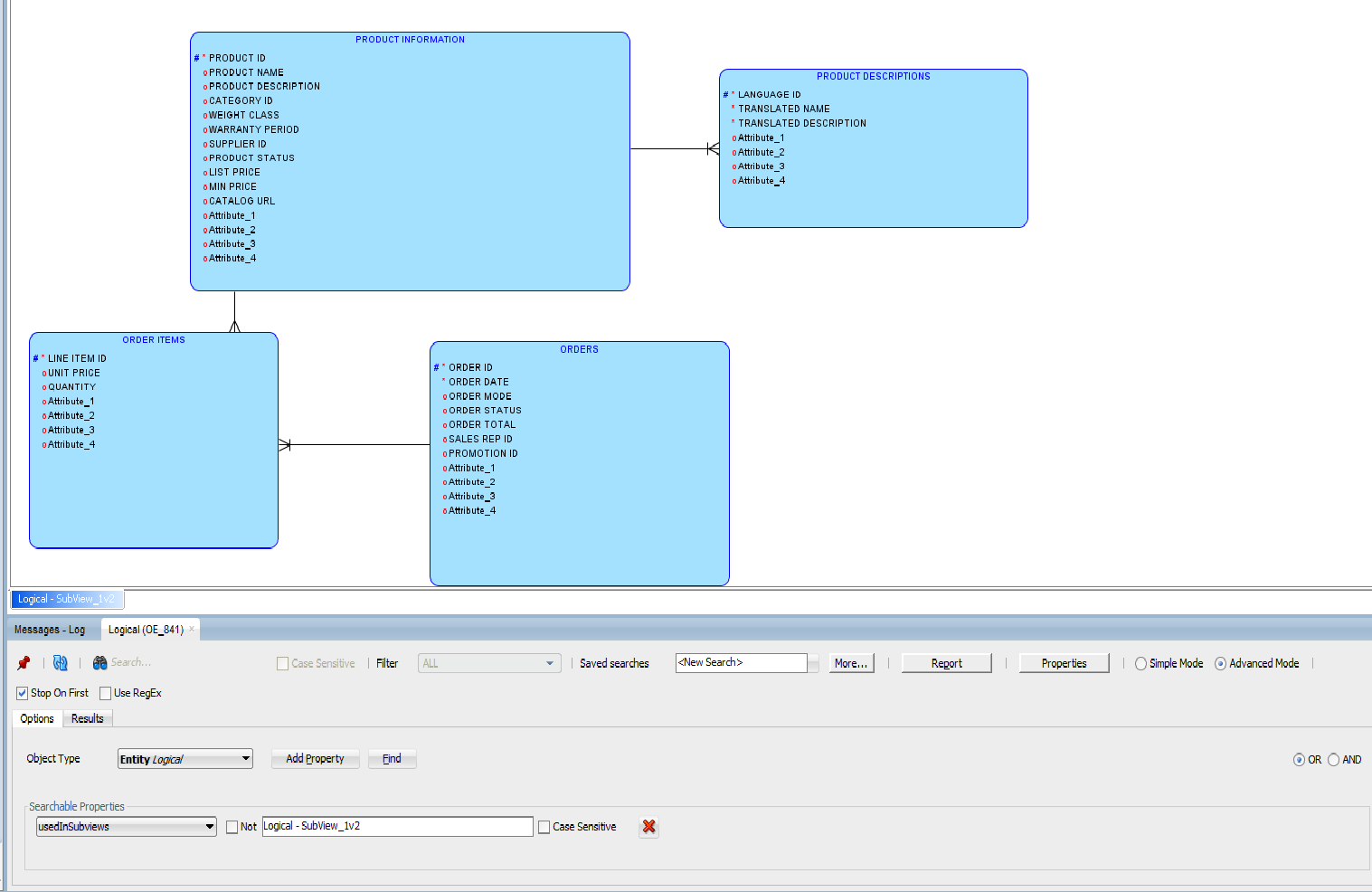
if(tv!=null){

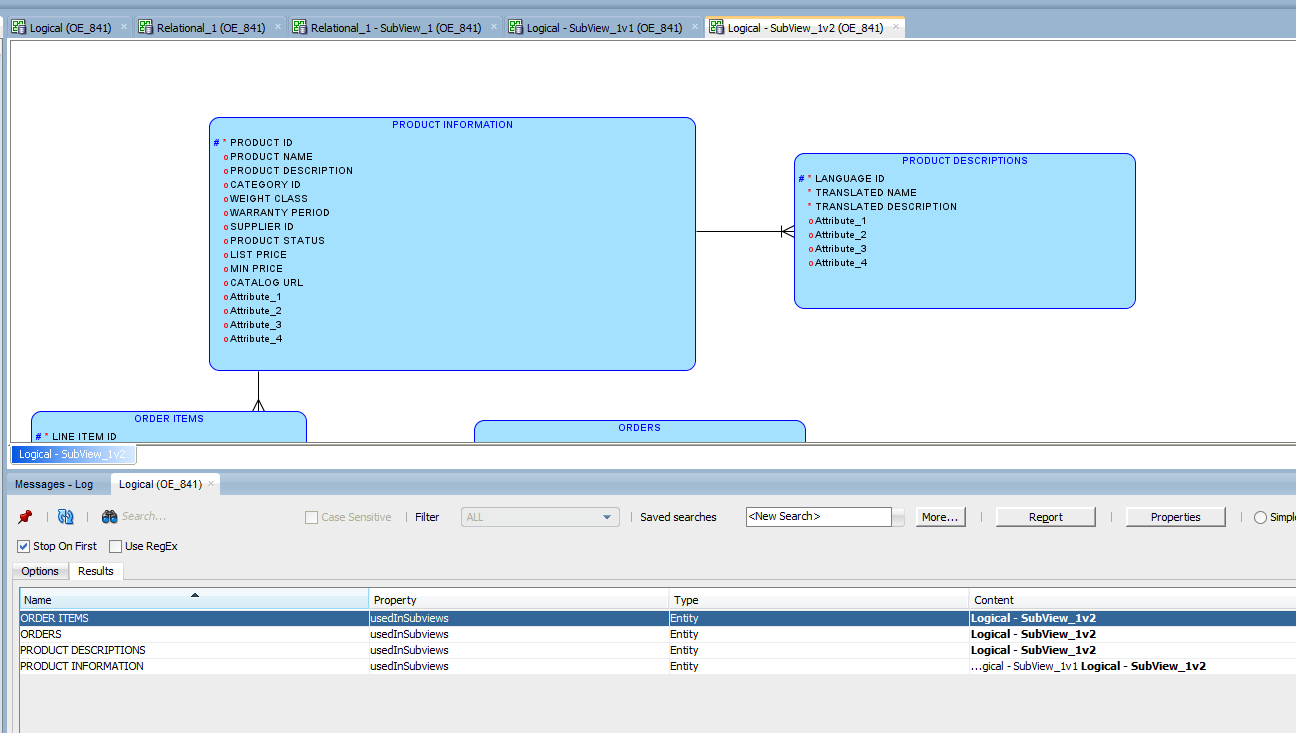
// processing for table goes here

Again using search functionality this can be done with less coding

1. Use search functionality – example is for entities in subview

Entities and tables has read-only property usedInSubviews – below are pictures showing usage of that property in Advanced search:





1. Use search result

objs = model.getLastSearchResult().toArray();

for(var i = 0;i<objs.length;i++){

entity = objs[i];

// still can check if it's entity

if(obj.getObjectTypeName().equals("Entity")){

1. Other examples

XI.1 Entity template

DM comes with transformation script called “Table template” which adds columns from table name “table\_template” to all other tables in the model. Here is script that do the same for logical model and enmities there using entity named "entity\_template" (transformation should be bound to logical model:

var t\_name = "entity\_template";

template = model.getEntitySet().getByName(t\_name);

if(template!=null){

tcolumns = template.getElements();

tables = model.getEntitySet().toArray();

for (var t = 0; t<tables.length;t++){

table = tables[t];

// compare name ignoring the case

if(!table.getName().equalsIgnoreCase(t\_name)){

for (var i = 0; i < tcolumns.length; i++) {

column = tcolumns[i];

col = table.getElementByName(column.getName());

if(col==null){

col = table.createAttribute();

}

column.copy(col);

table.setDirty(true);

}

}

}

}

The script can be modified to be used together with search functionality – then different subset of entities can get different common attribute simply by changing template entity and using different search criteria.

XI.2 Removing columns created with table template and related attributes in logical model

var t\_name = "table\_template";

var p\_name = "ctemplateID";

template = model.getTableSet().getByName(t\_name);

if(template!=null){

tcolumns = template.getElements();

tables = model.getTableSet().toArray();

for (var t = 0; t<tables.length;t++){

table = tables[t];

// compare name ignoring the case

if(!table.getName().equalsIgnoreCase(t\_name)){

for (var i = 0; i < tcolumns.length; i++) {

column = tcolumns[i];

col = table.getColumnByProperty(p\_name,column.getObjectID());

if(col!=null){

attr = col.getEngAttribute();

if(attr!=null){

ent = attr.getEntity();

attr.remove();

if(ent!=null){

ent.setDirty(true);

}

}

col.remove();

table.setDirty(true);

}

}

}

}

}

Though DM store design in set of directories and files, one can use scripting to store the whole design (or one model) into single file and to restore the status from such file:

* Saving:

model.getAppView().getXMLTransformationManager().saveObjectWithExternals("D:/SVN\_Local\_files/handy\_whole\_des\_big.xml",model.getDesign());

* Loading

Since design will take the status as it’s stored in the file it’s good to be sure design is empty – method isEmpty() can be used for models (DesignPart class).

model.getAppView().getXMLTransformationManager().loadObjectWithExternals("D:/SVN\_Local\_files/handy\_whole\_des\_big.xml",model.getDesign());

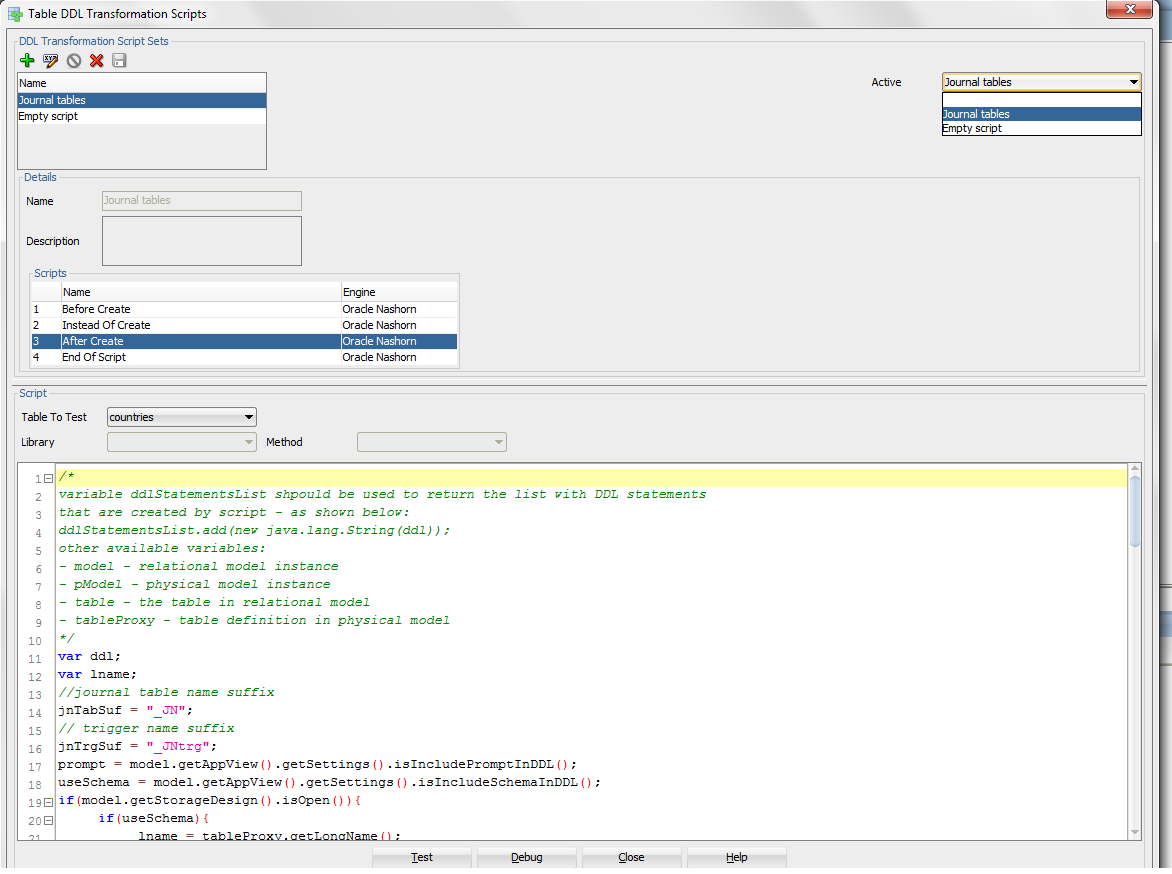
Note about physical models – only Oracle physical models are described in current revision of XML metadata thus non Oracle physical models cannot be saved into single file.

1. User defined DDL generation using transformation scripts

DM 4.1 allows transformation scripts to be used at specific points of DDL generation for table – “Before Create”, “Instead of Create”, “After Create” and “End of Script”.

If “Instead of Create” script is defined it’ll suppress built-in DDL generation for table and its components so DDL generation for non supported databases can be implemented.

Sets of such quadruples of scripts can be defined and one of them could be set as active – if no one is active then there is no custom DDL generation.

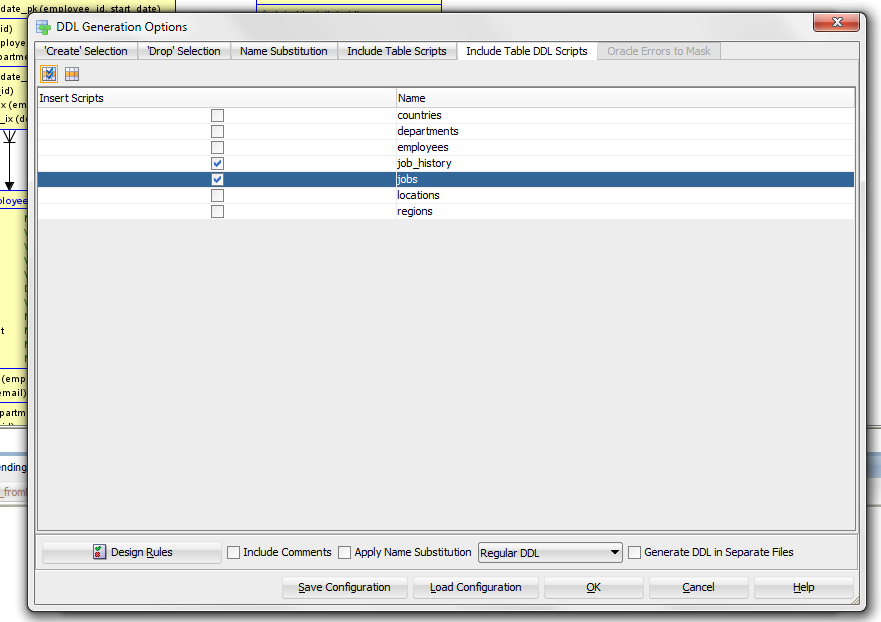


The dialog is activated from “Tools>Design rules and Transformations>Table DDL Transformations”. It allows writing, testing and debugging scripts. Table can be selected for “Test” and DDL preview window will pop-up and shows resulting DDL. There is no DDL preview in “Debug” mode.

Following variables are available to script during execution:

* + **model** - relational model instance;
  + **pModel** - physical model instance;
  + **table** - the table in relational model;
  + **tableProxy** - table definition in physical model;
  + **ddlStatementsList** - should be used to return the list with DDL statements that are created by script – each DDL statement should be added as is shown below:  
    *ddlStatementsList.add(new java.lang.String(ddl));*

There is a new property on table defined “Use DDL Transformation Script” and if it’s set then table will be included in custom DDL generation. That property can be set in table properties dialog or in “DDL generation options”



DM 4.1 is distributed with “After create” script that generates journal table and related supporting trigger providing 2 DDL statements for each processed table:

var ddl;

var lname;

//journal table name suffix

jnTabSuf = "\_JN";

// trigger name suffix

jnTrgSuf = "\_JNtrg";

prompt = model.getAppView().getSettings().isIncludePromptInDDL();

useSchema = model.getAppView().getSettings().isIncludeSchemaInDDL();

if(model.getStorageDesign().isOpen()){

if(useSchema){

lname = tableProxy.getLongName();

}else{

lname = tableProxy.getName();

}

}else{

if(useSchema){

lname = table.getLongName();

}else{

lname = table.getName();

}

}

if(prompt){

ddl= "PROMPT Creating Journal Table for '"+lname+"';\n";

}else{

ddl = "";

}

ddl = ddl + "CREATE TABLE "+lname+jnTabSuf+"\n"+

" (JN\_OPERATION CHAR(3) NOT NULL"+"\n"+

" ,JN\_ORACLE\_USER VARCHAR2(30) NOT NULL"+"\n"+

" ,JN\_DATETIME DATE NOT NULL"+"\n"+

" ,JN\_NOTES VARCHAR2(240)"+"\n"+

" ,JN\_APPLN VARCHAR2(35)"+"\n"+

" ,JN\_SESSION NUMBER(38)"+"\n";

cols = table.getElements();

for(var i=0;i<cols.length;i++){

col = cols[i];

ddl = ddl +" ,"+col.getName()+" "+col.getDatatypeString();

if(col.isMandatory()){

ddl = ddl + " NOT NULL\n";

}else{

ddl = ddl + "\n";

}

}

ddl = ddl + " );";

ddlStatementsList.add(new java.lang.String(ddl));

if(prompt){

ddl= "\nPROMPT Creating Journal Trigger for '"+lname+"';\n";

}else{

ddl = "\n";

}

ddl = ddl +

"CREATE OR REPLACE TRIGGER "+lname+jnTrgSuf +"\n"+

" AFTER \n" +

" INSERT OR \n" +

" UPDATE OR \n" +

" DELETE ON emp2 for each row \n" +

" Declare \n" +

" rec "+lname+"\_JN"+"%ROWTYPE; \n" +

" blank "+lname+"\_JN"+"%ROWTYPE; \n" +

" BEGIN \n" +

" rec := blank; \n" +

" IF INSERTING OR UPDATING THEN \n" ;

for(var i=0;i<cols.length;i++){

col = cols[i];

ddl = ddl + " rec."+col.getName()+" := :NEW."+col.getName()+"; \n";

}

ddl = ddl +

" rec.JN\_DATETIME := SYSDATE; \n" +

" rec.JN\_ORACLE\_USER := SYS\_CONTEXT ('USERENV', 'SESSION\_USER'); \n" +

" rec.JN\_APPLN := SYS\_CONTEXT ('USERENV', 'MODULE'); \n" +

" rec.JN\_SESSION := SYS\_CONTEXT ('USERENV', 'SESSIONID'); \n" +

" IF INSERTING THEN \n" +

" rec.JN\_OPERATION := 'INS'; \n" +

" ELSIF UPDATING THEN \n" +

" rec.JN\_OPERATION := 'UPD'; \n" +

" END IF; \n" +

" ELSIF DELETING THEN \n" ;

for(var i=0;i<cols.length;i++){

col = cols[i];

ddl = ddl + " rec."+col.getName()+" := :OLD."+col.getName()+"; \n";

}

ddl = ddl +

" rec.JN\_DATETIME := SYSDATE; \n" +

" rec.JN\_ORACLE\_USER := SYS\_CONTEXT ('USERENV', 'SESSION\_USER'); \n" +

" rec.JN\_APPLN := SYS\_CONTEXT ('USERENV', 'MODULE'); \n" +

" rec.JN\_SESSION := SYS\_CONTEXT ('USERENV', 'SESSIONID'); \n" +

" rec.JN\_OPERATION := 'DEL'; \n" +

" END IF; \n" +

" INSERT into "+lname+"\_JN"+" VALUES rec; \n" +

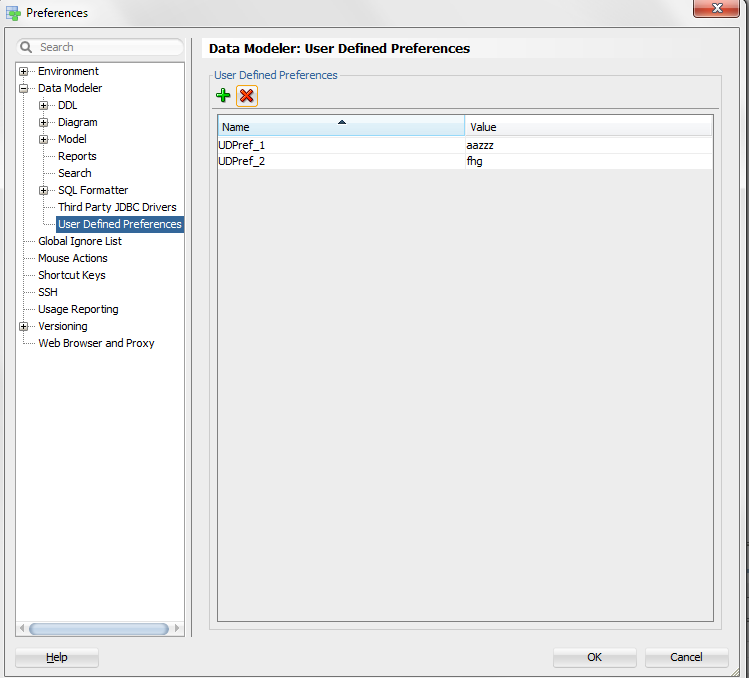
" END; \n" +

" /"

ddlStatementsList.add(new java.lang.String(ddl));

1. Usage of User Defined Preferences

DM 4.1.1 introduced possibility for user defined preferences:



They can be accessed in following way:

set = model.getAppView().getSettings();

pref1 = set.getUserDefinedPreferenceValue("UDPref\_1");