Talend Open Studio For Master Data Management: A Practical Starter Guide 2nd Edition

About the Author:
Diethard Steiner is working as an independent business intelligence consultant in London, UK and focuses mainly on open source business intelligence solutions. He is also author of the Diethard Steiner on Business Intelligence blog which offers regular tutorials on various open source tools and topics. You can find more info on LinkedIn. Follow him on Twitter or Google+.

Table of Contents:

Version used: Talend Open Studio for MDM v5.4.
Mission: Get you up and running quickly with Talend Open Studio for MDM! I will cover the basics and point you to further resources for the more advanced topics.
Prerequisite: You are already familiar with Talend Open Studio for Data Integration, Linux command line and have a SQL database installed.

Introduction

Master data management has gained a strong momentum in recent years. So what is it actually about?

Definition Master Data Management
In computing, master data management (MDM) comprises a set of processes and tools that consistently defines and manages the master data (i.e. non-transactional data entities) of an organization (which may include reference data). MDM has the objective of providing processes for collecting, aggregating, matching, consolidating, quality-assuring, persisting and distributing such data throughout an organization to ensure consistency and control in the ongoing maintenance and application use of this information. (Source: Wikipedia)

Just to highlight again: Talend MDM Server is supposed to hold **non transactional** data which is shared across various business departments.

The Talend MDM Server stores all the data in an XML database (eXist). All changes on the MDM data are versioned automatically.

For this tutorial we take on the role of a second hand camera franchise which has branches across various cities. Each store keeps their own local database. The initial aim is to have a central list of lens master data.

Installation

You can find detailed instructions online on Talend's website. Here I just want to offer a short summary.

In a nutshell (Linux/Ubuntu):

Download the TOS MDM package from the Talend website.

The download package will be named like this:
TOS_MDM-All-rReleaseNumber-VVersionNumber.zip.
Unzip the package and you will see that it contains two files:

- **MDM Studio**: TOS_MDM-Server-rReleaseNumber-VVersionNumber.jar
- **MDM Server**: TOS_MDM-Studio-rReleaseNumber-VVersionNumber.zip

Note: Amand TOS MDM version numbers in all commands listed below.

First install the server first using wizard:

```
cd ~/Downloads
unzip TOS_MDM-All-r84309-V5.1.1.zip -d ./TOS_MDM
cd ./TOS_MDM
java -jar TOS_MDM-Server-r84309-V5.1.1.jar
```

Follow the wizard.

Please note that the community edition only supports two databases:

- eXist: XML database
- H2: in-memory RDBMS

Once the server setup is complete, unzip the TOS MDM Studio zip file:

```
unzip TOS_MDM-Studio-r84309-V5.1.1.zip
```

Then copy it to a convenient location.

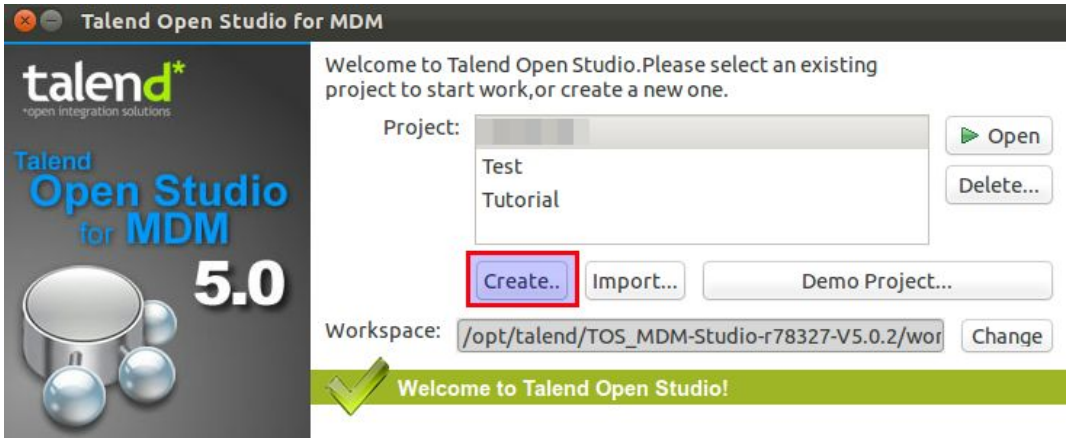
Navigate to the **TOS MDM Studio** root directory and run:

```
chmod 700 *.sh
```

Now you can start the **MDM Studio** like this:

```
sh ./TOS_MDM-linux-gtk-x86.sh
```

On startup you will be asked which project you want to access. We will create a dedicated project for this tutorial: Click on the **Create** button and name the project **Tutorial**.



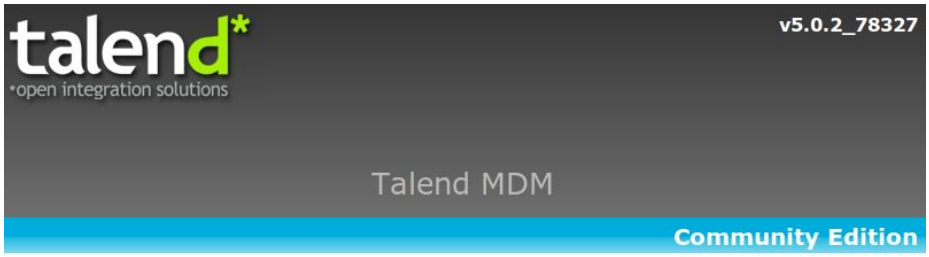
Click **OK**.
Then mark **Tutorial** and click on **Open**.

To start the MDM server, navigate to the **MDM server** root directory:

```
cd jboss-4.2.2.GA
chmod 700 *.sh
sh ./run.sh
```

Open your favourite internet browser and check if the server is properly running:

Web GUI
On successful installation, http://localhost:8080/talendmdm will show:



The open source version comes with only two user accounts (it is restricted to these two ones):

standard user
user: user
password: user

admin
user: administrator
password: administrator

Uninstalling the MDM Server

Should you have to uninstall the MDM server at some point, you can find an uninstall utility inside <jboss>/Uninstaller. Just run:

sudo java -jar uninstaller.jar

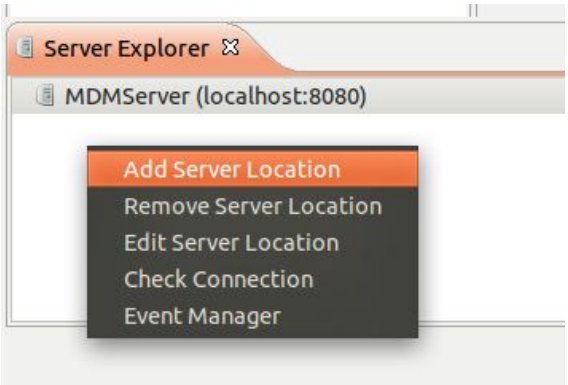
Add the server details to TOS MDM Studio

You should have **TOS MDM Studio** already running. You will realize that it comes with 3 **perspectives** (top right hand corner):

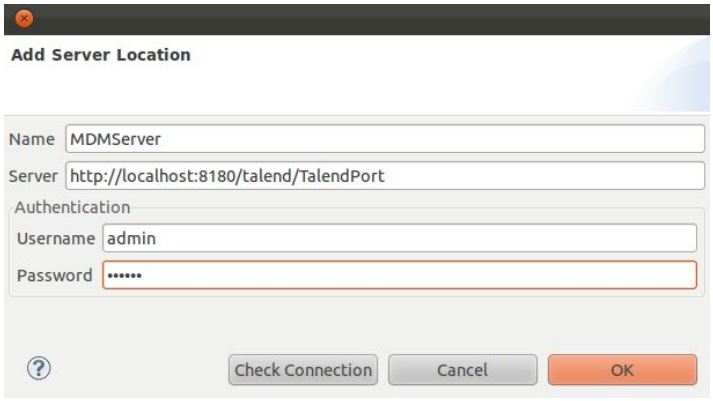
Make sure the **MDM perspective** is selected.

If the **Server Explorer** view is not visible, choose from the main menu **Window > Show View** and select **Server Explorer**.

Right click in the **Server Explorer** area and choose **Add Server Location**:



Then provide the MDM server details:



To connect to the server use:
server: http://localhost:8080/talend/TalendPort
user: admin
password: talend

These account details work as well, but you will not have admin rights:
user: administrator
password: administrator

Note: You might have to adjust the port number and domain in the server URL depending on your setup!

Click **OK**.

Preparation

Let's first create a sample database for our imaginary second hand camera store in London. This is just representation of one of many stores. We will use this database later on to synchronize the master data with the local store data. I will be working with PostgreSQL. In case you are working with another database, please amend the SQL respectively:

```
$ psql -Upostgres
CREATE DATABASE storelondon;
\c storelondon;
CREATE SCHEMA lenses;

SET search_path TO 'lenses';

CREATE TABLE
lens_metadata
(
lens_name VARCHAR(100)
```

```
, vendor VARCHAR(30)
, release_year INT4
, lens_category VARCHAR(20)
, min_focal_length INT4
, max_focal_length INT4
, min_aperture NUMERIC
, max_aperture NUMERIC
, min_focus_distance_mm INT4
, filter_size_mm INT2
, talend_mdm_key INT4
)
;

INSERT INTO
lens_metadata
VALUES
('Asahi Optical Co. Super Takumar 28mm f/3.5-22.0', 'Pentax', 1962, 'Wide', 28,
28, 3.5, 22, 40, 58, null),
('Carl Zeiss Jena Biotar 75mm f/1.5', 'Carl Zeiss Jena', 1951, 'Short
Telephoto', 75, 75, 1.5, 1.5, 80, 58, null)
;
```

Please note we that we decided to add a talend_mdm_key column to our raw data so that we can later on store the mdm key in here.

Modeling

Before we get started, let's first get a common understanding of the most important MDM terms:

Term	Description
(business) element	Also referred to as business attribute. The actual name of the data point.
(business) entity	Describes the actual data (the elements), its nature, its structure and its relationships.[1] An entity can have one or more business elements. The Talend MDM jargon for this concept is **data model entity**.
data model type	This is an element or collection of elements which is globally defined and can be used across various entities. This makes maintenance of common elements easier.
data model	Defines the attributes (elements), user access rights and relationships of entities mastered by the MDM Hub. The data model is the central component of Talend MDM. A data model maps to one or more (business) entities that can be explicitly

	defined. Any concept can be defined by a data model.[1] A data model can have multiple entities.
(business) domain	A collection of data models that define a particular concept. For instance, the customer domain may be defined by the organization, account, contact and opportunity data models. A product domain may be defined by a product, product family and price list. Ultimately, the domain is the collection of all data models that relate to a concept. Talend MDM can model any and many domains within a single hub. It is a generic multi-domain MDM solution.[1]
data container	Holds data of one or several business entities. Data containers are typically used to separate master data domains.[1]

[1] Source: Talend MDM Guide

You can think of these ones as a hierarchy: A **data model / domain** can have one or many **business entities** and a business entity can have one or many **elements**.

How to create a data model

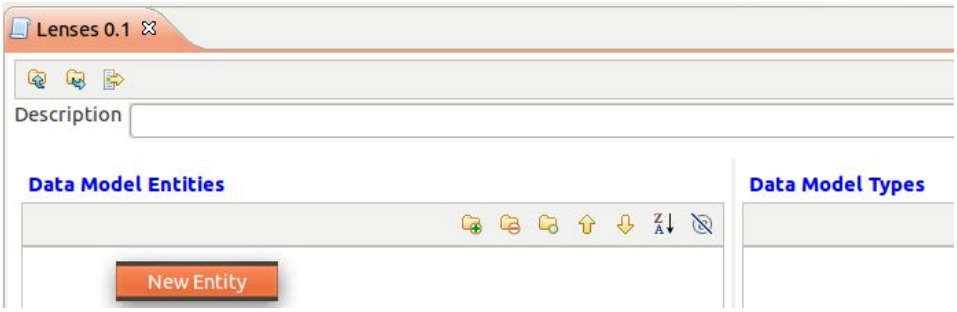
Right click on **Data Model** in the repository and choose **New**:

Name the data model **Lenses** and click **OK**. Theoretically, lenses could be part of a much bigger context, but let's keep it simple here.

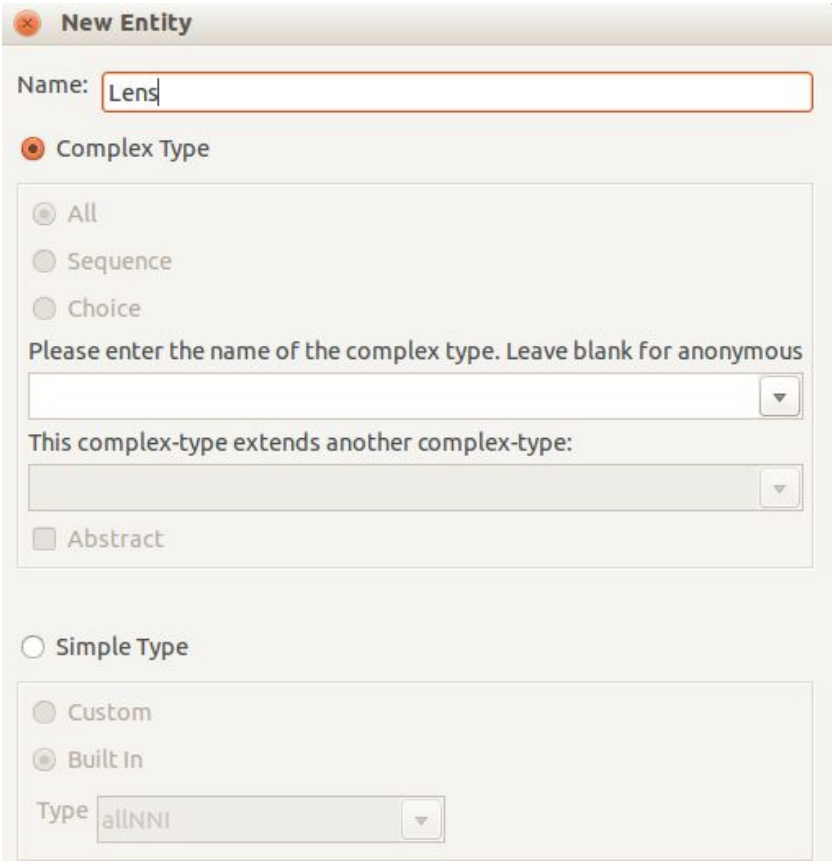
How to create data model entities

As already mentioned above, data model entities are the actual business entities. So in our case we will have a lens entity (For the purpose of this tutorial we keep it very simple here).

Right click on the main data model area and choose **New Entity**:



Name the new entity **Lens** and leave everything else empty:



What follows is a short description of the available options in this dialog window:

Let's have a look at the available **types**:

- **Simple type**: Used for single, self contained elements like email addresses.
- **Complex type**: Used for structures like address which consists of multiple elements. A complex type can also inherit elements from another complex type.

The **Complex type** has following options: They are grayed out/disabled by default

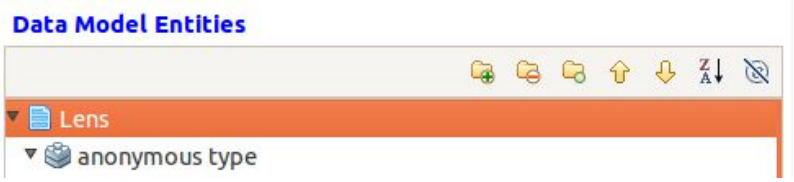
- **All**: lists the entity in any sequence. This is the **default** one.
- **Sequence**: lists the entities in the defined sequence
- **Choice**: to have a choice on the entities

Option: **Please enter name of the complex type. Leave blank for anonymous.**
If you specify a name here, you create a reusable type. This type will show up under **Data Model Types** in the **Data Model** design area. Once you define a name, you will also be able to select **All**, **Sequence** or **Choice**. If you don't define a name here, **All** will be default one.

Option: **This complex-type extends another complex-type**
Complex types can inherit elements from another complex type. This is very useful (and advanced) feature. Example: You can create an Address type which has first name and surname as elements, then you add US address and EU address type, which both inherit the elements from the Address type and add the area specific elements.

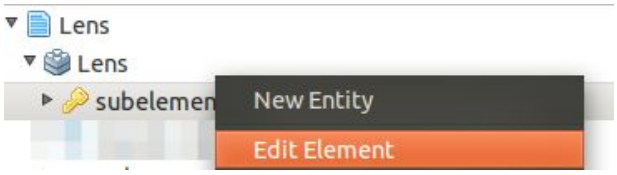
For our entity we leave everything on the default settings and click **OK**.

You will see that now you have a new entity under **Data Model Entities**:

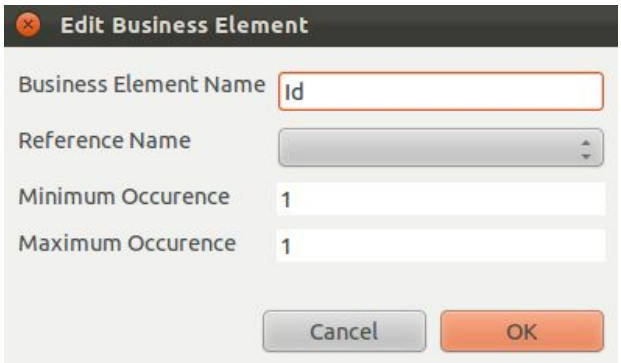


How to define an auto increment key

In our case the unique key will be an auto incremented identifier. Expand the **Lens** entity completely and then right click **subelement**, which is easy to identify as it has a **key** symbol next to it. Choose **Edit Element**:

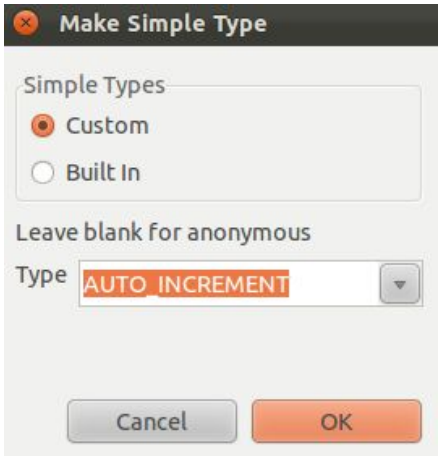


Change the name to **Id** and then click **OK**:

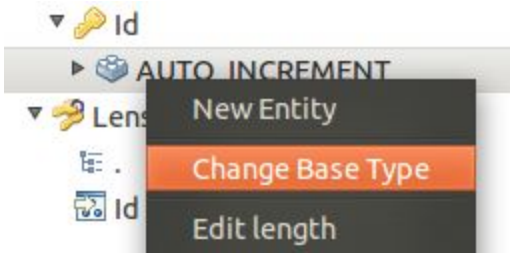


Using auto increment integer

Right click on the key element and choose **Change to Simple Type**. Then choose **Custom** and choose **AUTO_INCREMENT** as **Type**.



Next right click the **AUTO_INCREMENT** element and choose **Change Base Type**:

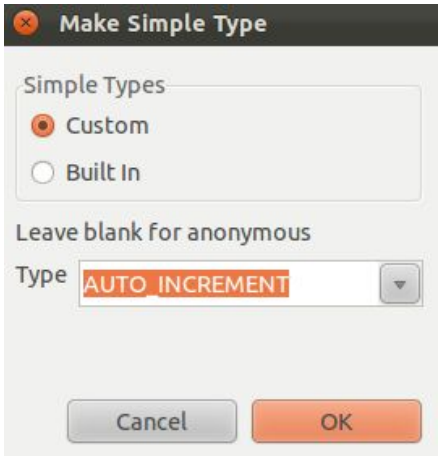


Set the **built-in Type** to **Integer**:

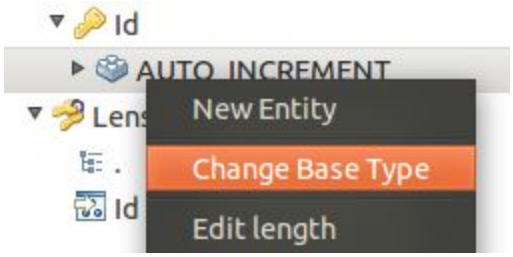
Using UUID

This section is for your information only, hence do only read this, do not follow steps practically:

Right click on the key element and choose **Change to Simple Type**. Then choose **Custom** and choose **AUTO_INCREMENT** as **Type**.



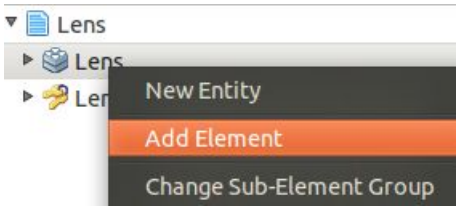
Next right click the **AUTO_INCREMENT** element and choose **Change Base Type**:



Then choose **Custom** and choose **UUID** as **Type**.

How to create elements

We have to define all the lens attributes now, so let's start with the lens name. Right click on **Lens** and choose **Add Element**:

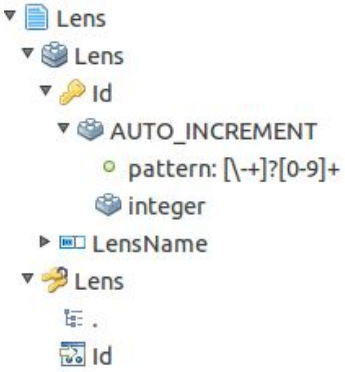


Define the name **LensName** and set **Minimum Occurrence** to **1**. This way the business

element has to be defined by the end user:

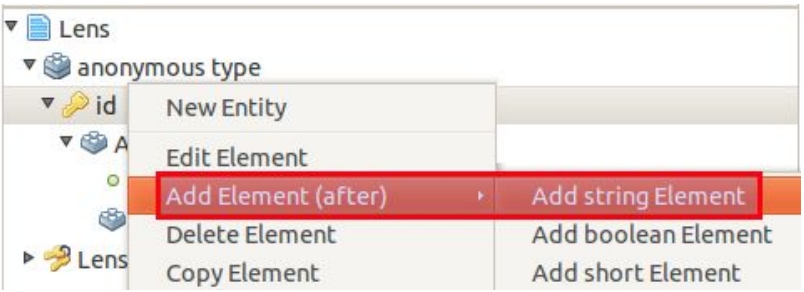
Then click **OK**.
In this case the default type **String** is just what we need.

The entity overview should look like this now:



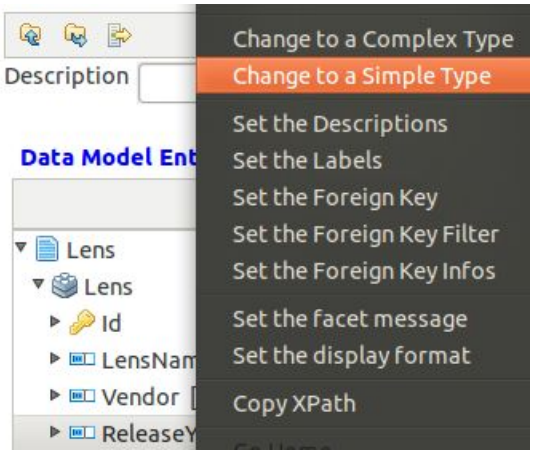
Press **CTRL+S** to save the model.

Note: An alternative way to add a new element is to right click on an existing element, like the Id (our key) and choose **Add Element (after) > Add string Element**. This is a bit more convenient as you can choose the data type straight away [This function is not available in Talend Studio v5.0]:

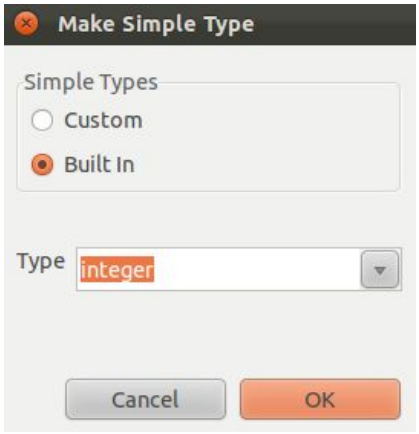


Next create a new element called **Vendor** which will be of type String as well. This element is optional, so the minimum occurrence has to be 0.

Right click on the **Lens entity** and choose **Add Element**: name it **ReleaseYear**. This one has to be of type **Integer**, hence right click on **ReleaseYear** and choose **Change to Simple Type**:



Now choose **Integer** from the built-in types:



Click **OK**.

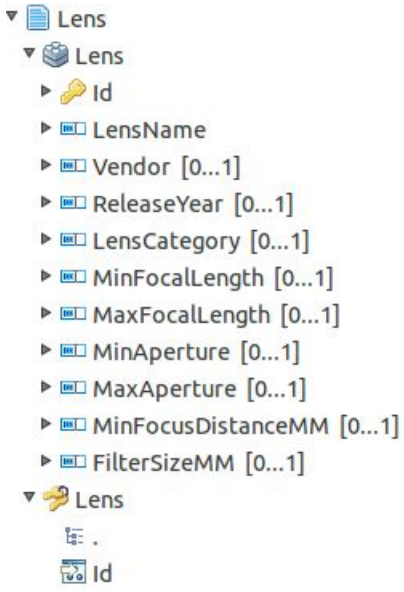
So far you familiar with adding elements in the following ways:

- From the entity node, which doesn't allow you to specify the data type straight away. You learnt how to change the data type later on.
- From an existing element, which is the most convenient method, as you can choose the data type straight away. I recommend using this method.

Finally add all the other business elements:

- LensCategory: String
- MinFocalLength: Integer
- MaxFocalLength: Integer
- MinAperture: Double
- MaxAperture: Double
- MinFocusDistanceMM: Integer
- FilterSizeMM: Integer

When completed, your entity should look like this:



Save your model. As we are quite curious people, we would like to understand what Talend MDM Studio actually did "under the hood" for us. Click on **Schema Source** tab and try to read through the schema:

```
1  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
2    <xsd:import namespace="http://www.w3.org/2001/XMLSchema" />
3    <xsd:element name="Lens">
4      <xsd:complexType>
5        <xsd:all>
6          <xsd:element maxOccurs="1" minOccurs="1" name="id" type="AUTO_INCREMENT" />
7          <xsd:element maxOccurs="1" minOccurs="1" name="LensName" type="xsd:string" />
8          <xsd:element maxOccurs="1" minOccurs="0" name="Vendor" type="xsd:string" />
9          <xsd:element maxOccurs="1" minOccurs="0" name="ReleaseYear" type="xsd:integer" />
10         <xsd:element maxOccurs="1" minOccurs="0" name="LensCategory" type="xsd:string" />
11         <xsd:element maxOccurs="1" minOccurs="0" name="MinFocalLength" type="xsd:int" />
12         <xsd:element maxOccurs="1" minOccurs="0" name="MaxFocalLength" type="xsd:int" />
13         <xsd:element maxOccurs="1" minOccurs="0" name="MinAperture" type="xsd:double" />
14         <xsd:element maxOccurs="1" minOccurs="0" name="MaxAperture" type="xsd:double" />
15         <xsd:element maxOccurs="1" minOccurs="0" name="MinFocusDistanceMM" type="xsd:int" />
16         <xsd:element maxOccurs="1" minOccurs="0" name="FilterSizeMM" type="xsd:integer" />
17       </xsd:all>
18     </xsd:complexType>
19     <xsd:unique name="Lens">
20       <xsd:selector xpath="." />
21       <xsd:field xpath="id" />
22     </xsd:unique>
23   </xsd:element>
```

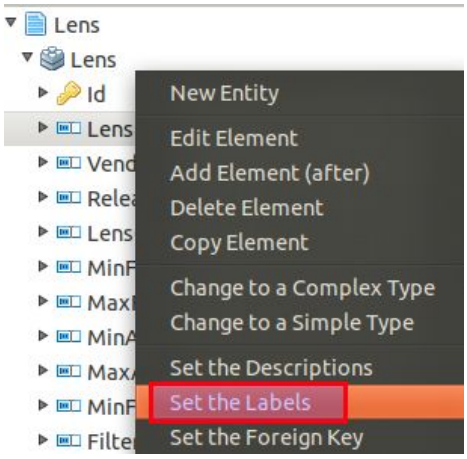
Remember that earlier on I mentioned that the default complex Type is **All**, if you leave the type set an anonymous (if you don't create a dedicated type which can be inherited). As you can see from the schema, this is what happened in our case.

You just created your first business entity! We don't want to make it more complex at this point in terms of modelling. Just reflect once again what you have been doing so far and then you should be ready for the next step.

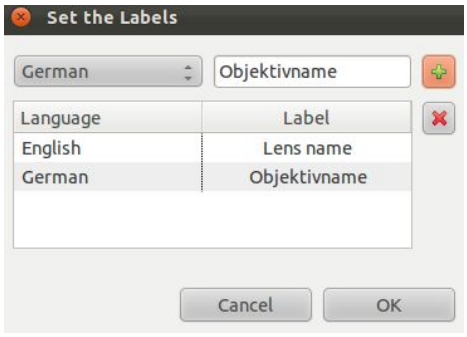
Set labels and descriptions

Each entity and element should have human readable / pretty print labels and a proper description. The great things about Talend MDM Studio is that it includes internationalization, meaning you can add labels and descriptions in various languages/locals.
Our imaginary second hand camera chain happens to have a store in Austria as well, hence we will not only add English labels and descriptions, but German ones as well.

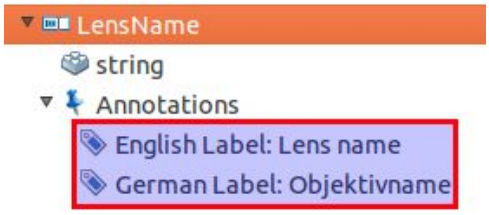
Right click on the **LensName** business element and choose **Set Labels**:



Choose the **English** language and type **Lens** into the input field, then click the press the **+** icon. Now change the language to **German** and write **Objektivname**, click **+** again:



In the entity overview these labels will now show up under the **annotations** element:



Now let's add a description, so that end users know exactly what information they have to provide: Right click **LensName** again and choose **Set the descriptions**. The approach is similar to the labels, so I will not go through it again. Please insert following text:

English:
The full name of the lens as engraved on the lens.
German:

Der vollstaendige Name wie am Objektiv ablesbar.

Save your model again.

All this internationalization will come into play when the end users log on to the Talend MDM web interface. If in example an Austrian user logs on, all the labels and descriptions will be displayed in German.

Define display format for date and numbers

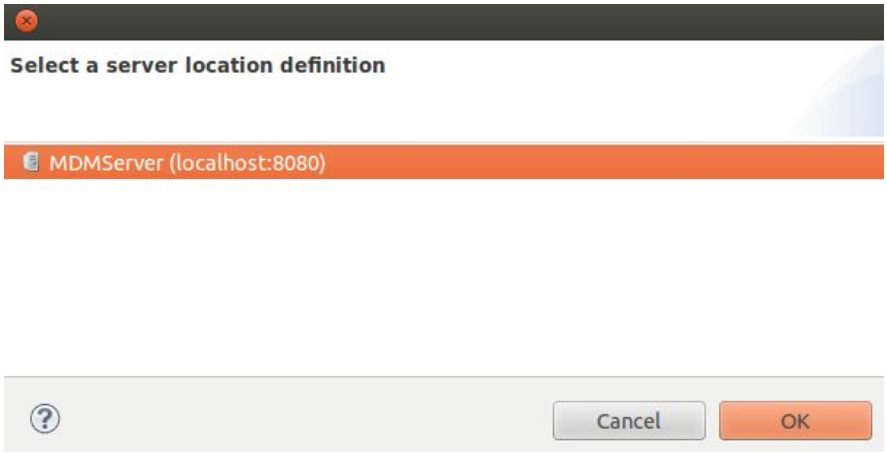
Just some short info: Talend Open Studio for MDM makes use of the java.util.Formatter class. Have a look here for a detailed description. Right click on a element and choose **Set display format**.

Note that Java automatically substitutes the local specific decimal separator.

How to deploy a data model

Once you have finalized your data model, you can deploy it to the MDM server. Right click the data model in the repository and and choose **Deploy To …**.

NOTE: If you get an error message that your data model cannot be deployed because it is locked, make sure that you close your data model in case it is open in the design area.

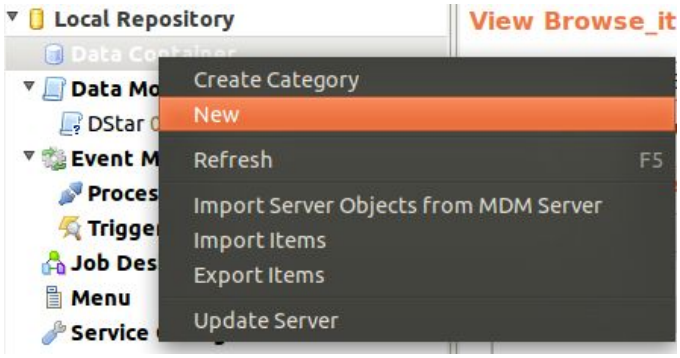


Choose the server you want the data model to be deployed to. You will get a success message if deployment process completes without problems. Note, that the icon next to the data model in the repository will change slightly and display a green play symbol. Also, next to the data model name you will find the server name.

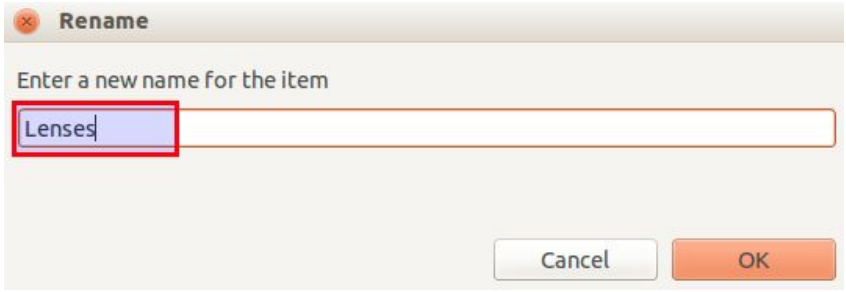
Data Container

All the master data is stored in a **Data Container**. A data container can hold the data of various

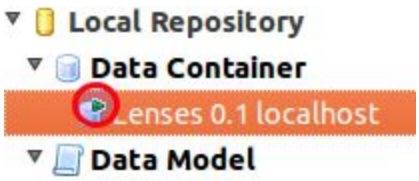
business entities. Note that a business entity stored in one data container is not visible from another data container.
To create a data container, simple right click on the **Data Container** in the repository tree and choose **New**:



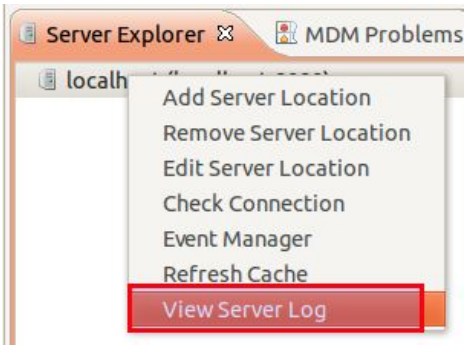
Define a name:



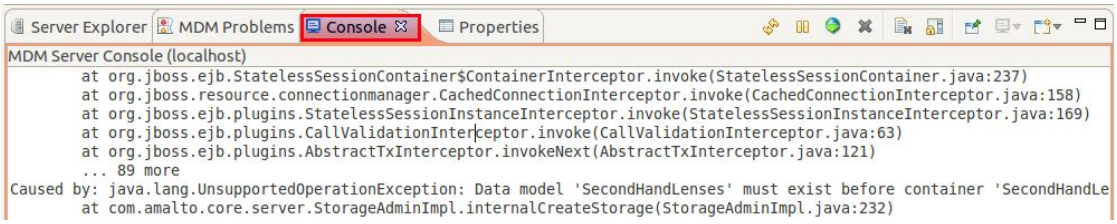
Important: Since version 5.2 the **data container and the data model must have the same name!** (see also here)
Once the container is defined, right click on it and choose **Deploy to …** to deploy your data container to the **MDM server**.
If everything goes fine, you will get a success message and see a play icon next to the data model:



Note: It is worth keeping an eye on the server log. The Talend MDM Studio offers a function to show the server log: In the **Server Explorer** tab, right click on a registered server and choose **View Server Log**:



A new tab called **Console** will be activated:



If you see following error (or similar) in the log:

```
Caused by: java.lang.UnsupportedOperationException: Data model
'SecondHandLenses' must exist before container 'SecondHandLenses' can be
created.
```

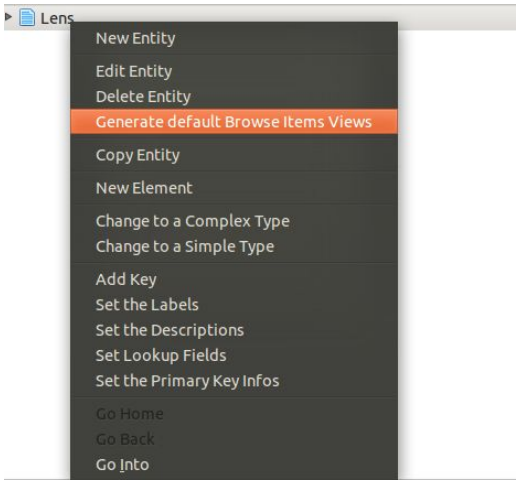
This error was raised because your **data container** does not have the same name as the **data model**. To fix this rename your data container to exactly the same name as your data model and then deploy it.

Please note that in v5.0 this 1:1 relationship data container - data model was not enforced. This might have to do with the recent addition of RDBMS. The last time I tested the example project discussed in this book was with Talend MDM v5.3 using the embedded H2 database and I got the exception shown above. Some of the content in this book (especially screenshots) are still showing a data container name *SecondHandLenses*. I didn't have the time to update those screenshots, so please keep this info in mind.

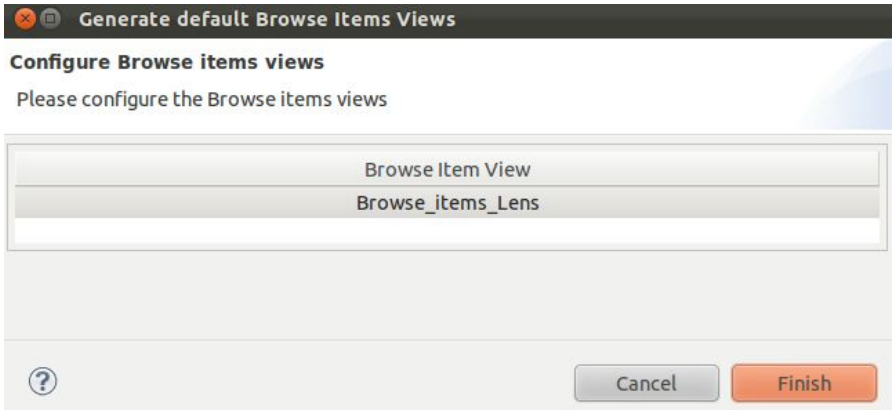
How to create a view

A view is basically what an end user can see via the web interface, which includes the form and search functionality. There are various views that you can create. We will only have a look at the most simple view here, which basically will allow end users to create the business entity online and search for values within certain attributes.

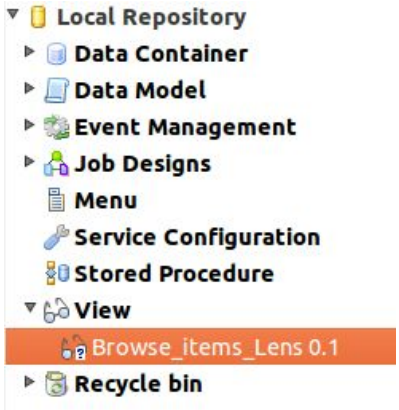
To auto-generate a view, open the **business model** and right click on the entity for which you want to create a view. Choose **Generate default Browser Items Views**:



In the following dialog click on **Finish**:



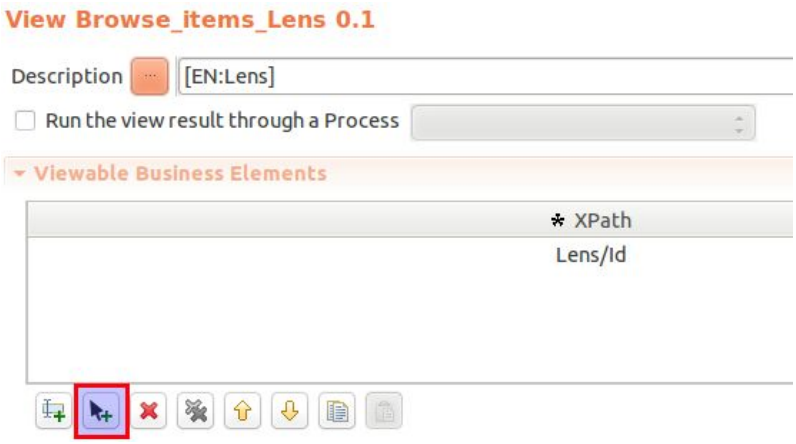
The auto-generated view will show up in the **View** node of the repository browser:



Double click on the view element to inspect it in the design view. The design section is divided into **viewable** and **searchable** business elements. The auto-generated view might not include all the elements you require. You can simply add additional elements by specifying the XPath. To create the XPath, you can use a visual navigator, so even users not familiar with XPath can easily create the views.

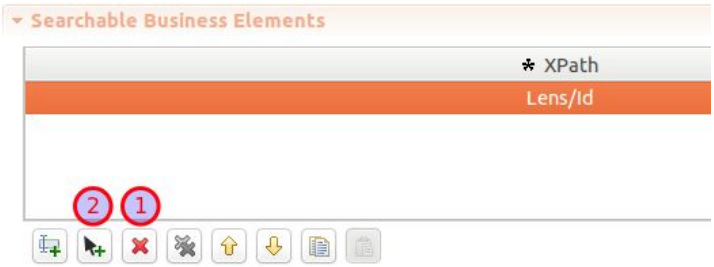
In our case we can see that only **Id** was added automatically. To speed this process up, click on

the **Add Multiple** button:



Mark the elements which you want to add to the view (In our case everything except the Id) and click **Add**:

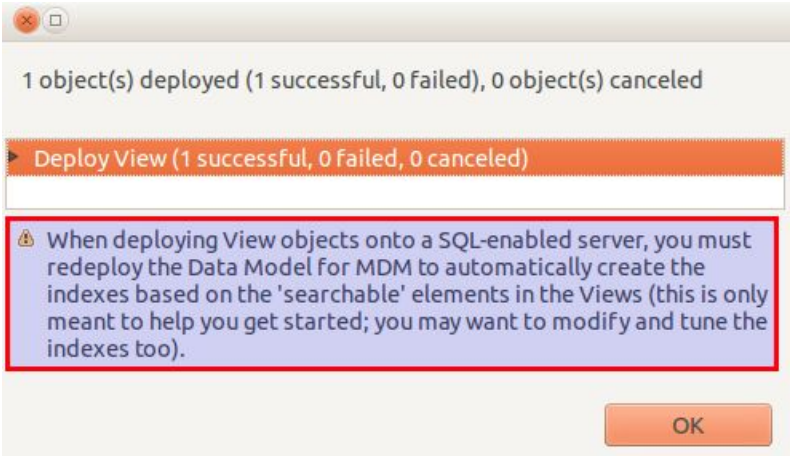
In the **Searchable Business Elements** section delete the **Id** element and add the **LensName**:



This will allow the end users to search for records by **LensName**.

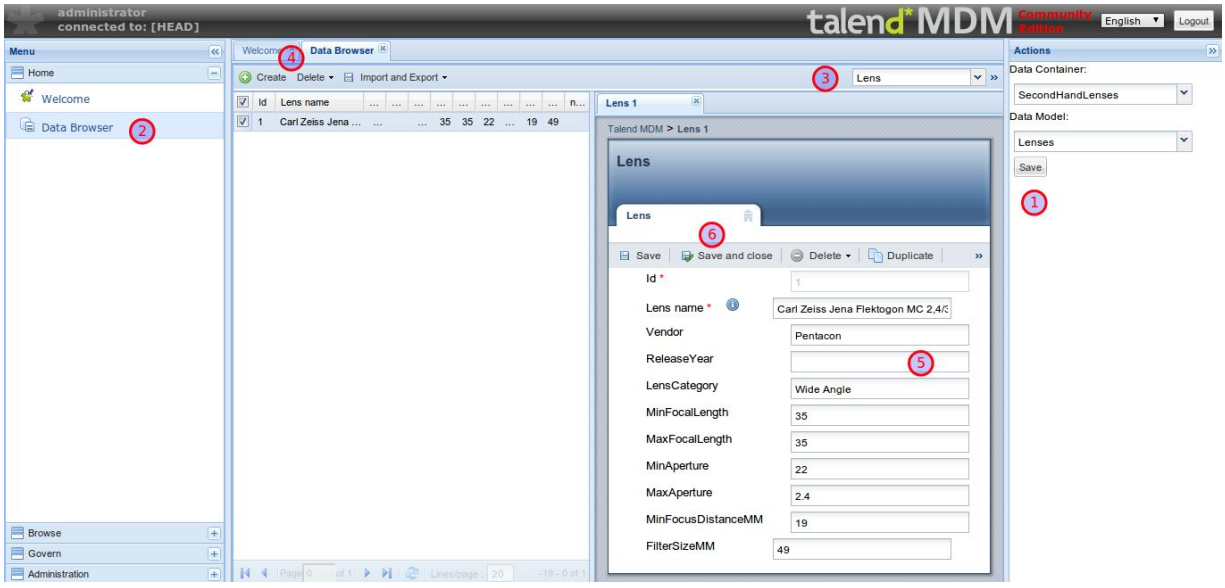
Once done, **save** the view and **close** it (the design view only, not Talend Open Studio). Then right click on the view in the MDM **repository browser** and choose **Update Server**.

Note: Later versions of Talend MDM support storing the data in relational databases (RDBMS). Pay special attention to the status message that is shown after publishing the view on the MDM server:



As you can understand from the info text, redeploy the data model to the MDM server so that the appropriate indexes on the RDBMS can be created automatically for the searchable elements. This is quite a convenient feature!

In order to better understand what you have created so far, open your favourite web browser and navigate to http://localhost:8080/talendmdm (Adjust URL accordingly to your setup). Log on with user *user* and password *user*.



Follow these steps:

1. On the left hand side, set the **Data Container** to *Lenses* and **Data Model** to *Lenses*.
2. On the right hand side, double click on on **Data Browser**.
3. In the menu (in the top centre of the screen) choose **Lens** from the entities pull down menu.

4. Click on **Create** to add some data.
5. Fill out the form.
6. Click on **Save**.
7. The new record will show up in the data area.

Congratulations! You have just created you first Master Data model and provided end users with a web interface to manage the data of this model.

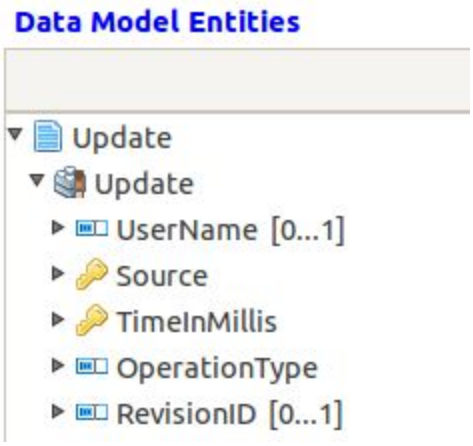
The Update Report

The Talend MDM server keeps a change log called **Update Report**. The data is stored in XML format as well.

The Update Report
Whenever a record is created/updated/deleted (CRUD), the *MDM Server* logs this activity in the Update Report - this is similar to an audit report. This log holds information about who added/updated/deleted a record, when, for which data container, data model, business entity etc; so in a nutshell some metadata - but it does not contain the data record itself. In case of updates, the old and new values of a business attribute are part of the Update Report. This Update Report is then sent to the *Event Manager*, which evaluates all *Trigger* conditions against this report. If a condition is met, the *Event Manager* will route the report to the specified process or job.

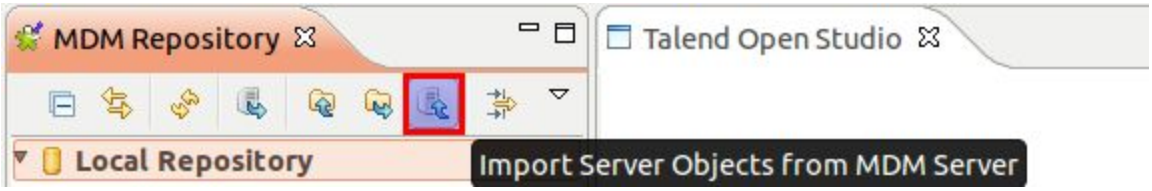
You will be pleased to hear that there is a data model for the **Update Report** as well for you to inspect. Under **Data Model > System** double click on **UpdateReport**:

Now you can analyse the structure of this data model in the main design area:

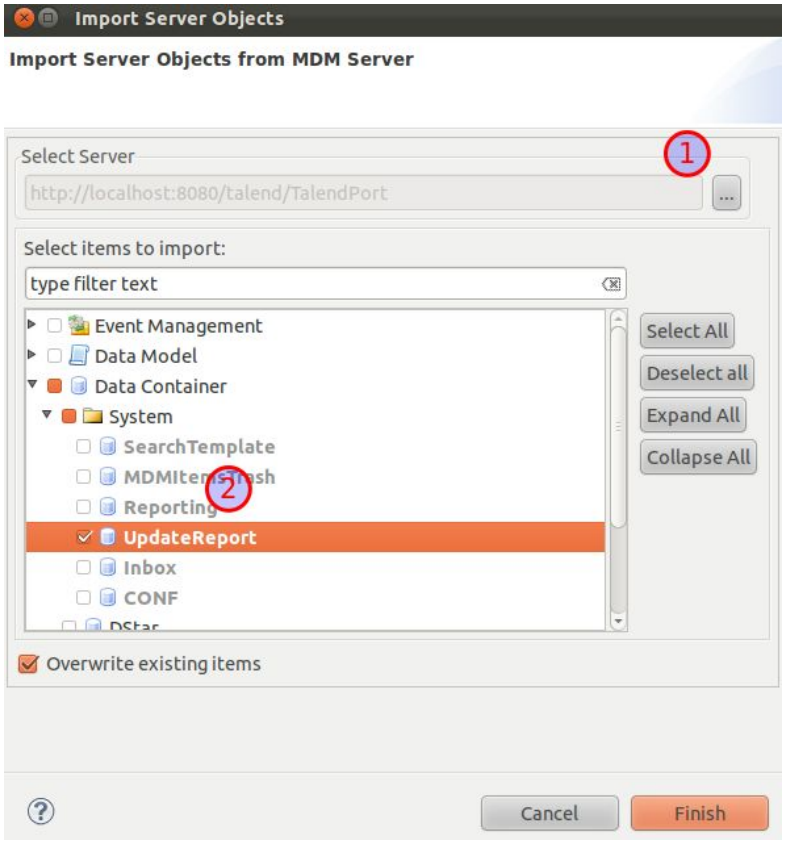


Let's import the **Update Report** data container from the MDM server so that we have some data to play with:

Click the **Import Server Objects from MDM Server** icon:



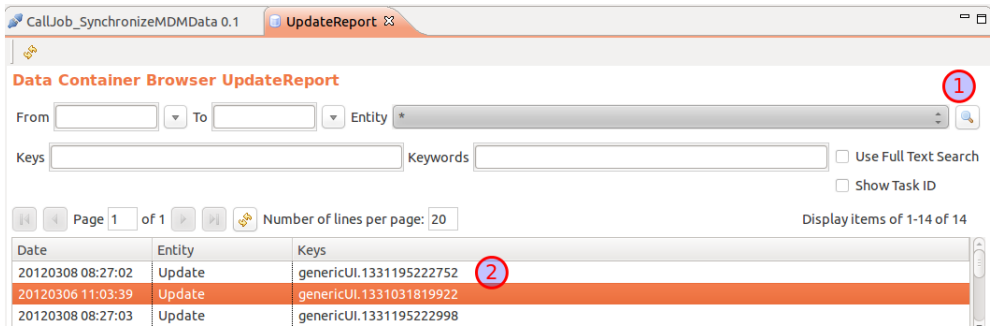
Choose (1) the respective server and (2) untick all elements in the import section and then tick the **Data Container > System > UpdateReport** element:



Click **Finish**.

The **UpdateReport** will now show up in your local repository under **Data Container > System > UpdateReport**. Double click on it:

Specify the search criteria (if any - in our case you can just leave everything empty), run a search and **double click** on the record for which you want to retrieve the XML fragment:



In the XML viewer click the **Source** tab to see the XML fragment:

Tree Source

```
<Update xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <UserName>administrator</UserName>
      <Source>genericUI</Source>
      <TimeInMillis>1334780602406</TimeInMillis>
      <OperationType>CREATE</OperationType>
      <RevisionID>null</RevisionID>
      <DataCluster>SecondHandLenses</DataCluster>
      <DataModel>Lenses</DataModel>
      <Concept>Lens</Concept>
      <Key>1</Key>
    </Update>
```

Just to highlight it once more, the **Update Report** does not contain the actual data (the full record). Let's look at some examples:

Update Report for a newly created record:

Tree Source

```
<Update>
  <UserName>user</UserName>
  <Source>genericUI</Source>
  <TimeInMillis>1381695757541</TimeInMillis>
  <OperationType>CREATE</OperationType>
  <RevisionID>null</RevisionID>
  <DataCluster>Lenses</DataCluster>
  <DataModel>Lenses</DataModel>
  <Concept>Lens</Concept>
  <Key>4</Key>
</Update>
```

As you can see the XML contains all the Metadata on the record (Who, Where, When, …). The Key points to the actual data record.

Update Report for an updated record:

Tree Source

```
<Update>
  <UserName>user</UserName>
  <Source>genericUI</Source>
  <TimeInMillis>1381695682972</TimeInMillis>
  <OperationType>UPDATE</OperationType>
  <RevisionID>null</RevisionID>
  <DataCluster>Lenses</DataCluster>
  <DataModel>Lenses</DataModel>
  <Concept>Lens</Concept>
  <Key>1</Key>
  <Item>
    <path>Vendor</path>
    <oldValue>Pentacon</oldValue>
    <newValue>Carl Zeiss Jena Pentacon</newValue>
  </Item>
</Update>
```

In case of an update, the Update Report contains the old and new values of the affected fields.

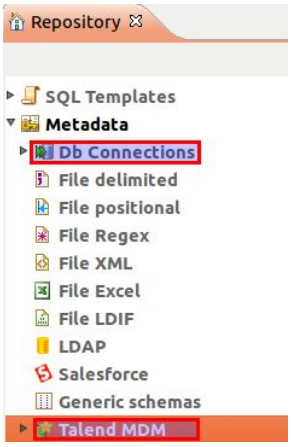
Data Integration

After creating the model and the view(s), the next task is to load existing data into the MDM data container and set up processes which in example synchronize new, deleted and updated records with remote databases. In our case we want to load a list of existing lens metadata and then make sure that each new and updated record gets synchronized with all our stores, which each hold an individual database.

It is best if you change the TOS perspective to **Integration** now:

Getting Started

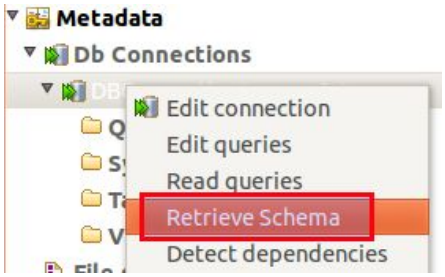
It's good practice to store the connection details in the **Talend Metadata**:



Create a connection for your database and retrieve the **lens_metadata** schema. I assume you already know how to set up a database connection, hence I will not discuss this in too much detail.

Create the source database connection:

1. Right click on **Db connections** and choose **Create new connection**. Name it **DBConnectionLenses**.
2. On the next screen provide the connection details. The database is called **storelondon** and the **schema** lenses. Click **Check** to make sure that the connection details are correct.
3. Right click on **DbConnections > DBConnectionLenses** and choose **Retrieve Schema**:



4. Click **Next** and on the second screen expand the nodes and then tick **lens_metadata**:

Click **Next**.

5. On the next screen you can see all the columns of the table. You can map the **DB column names** to Talend Data Integration **column names**. For the **database schema definition** set the **column names** to the same names as you defined them in the **business entity**. Doing it this way saves us creating an additional mapping step in the DI jobs. **Note**: Currently, when you change the **Column** name, also the **Db Column** name gets automatically set to the same value. My suggestion is that you first copy the original **Db Column** name before changing the **Column** name. Once you changed the **Column** name, simply paste the old value back to the **Db Column**.
   5.1. Also tick **Key** for the **talend_mdm_key**, so that this field is used as a key for updates and deletions. Note that here the **Column** name has to be **Id**, the same as we defined in our data model. The reason why we set this field as a key is that we want to use it later on to identify updates and deletions.
   5.2. Move the **Id** (talend_mdm_key) column to the top (so that it is the very first row) by using the arrow up button. This will facilitate the mapping later on to the Talend MDM dataset.
   5.3. Make sure to set the **Type** correctly for each column.
   5.4. Your database schema definition should look like this:

Column	Db Column	Key	DB Type	Type
Id	talend_mdm_key	☑	INT4	Integer
LensName	lens_name	☐	VARCHAR	String
Vendor	vendor	☐	VARCHAR	String
ReleaseYear	release_year	☐	INT4	Integer
LensCategory	LensCategory	☐	VARCHAR	String
MinFocalLength	min_focal_length	☐	INT4	Integer

MinFocalLength	min_focal_length	☐	INT4	Integer
MaxFocalLength	MaxFocalLength	☐	INT4	Integer
MinAperture	min_aperture	☐	NUMERIC	Double
MaxAperture	max_aperture	☐	NUMERIC	Double
MinFocusDistanceMM	min_focus_distance_mm	☐	INT4	Integer
FilterSizeMM	filter_size_mm	☐	INT4	Integer

Click **Finish**.

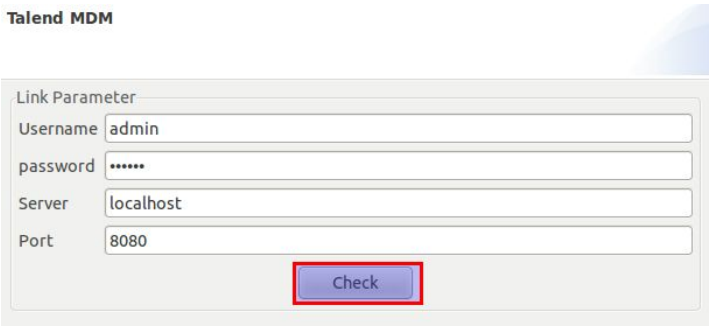
Now let's focus on the MDM connection:

Right click on **Talend MDM** and choose **Create MDM connection**. Specify **MDMConnectionLenses** as the name for the connection, click **Next**. Then fill in the connection details as shown below:

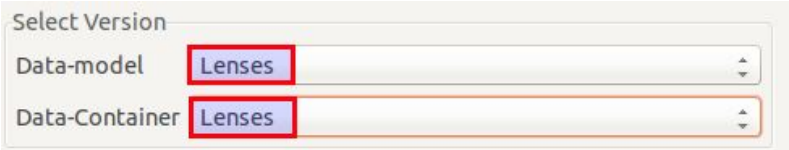
server: localhost
port: 8080
username: admin
password: talend

Alternatively:
username: administrator
password: administrator

Adjust server and port details according to your setup.

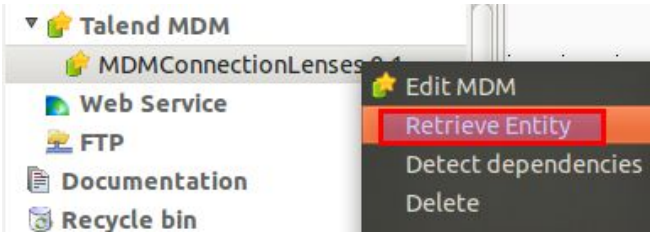


Click on **Check** to make sure that you provided the correct connection details. Then click **Next**. On the following dialog choose **Lenses** for the data model and **Lenses** for the data container:

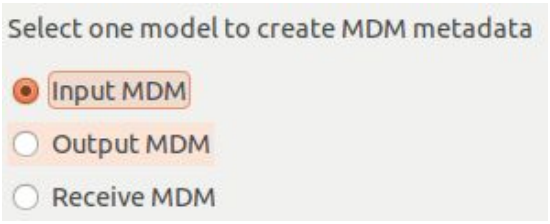


Click **Finish**.

Then right click **MDMConnectionLenses** and choose **Retrieve Entity**:

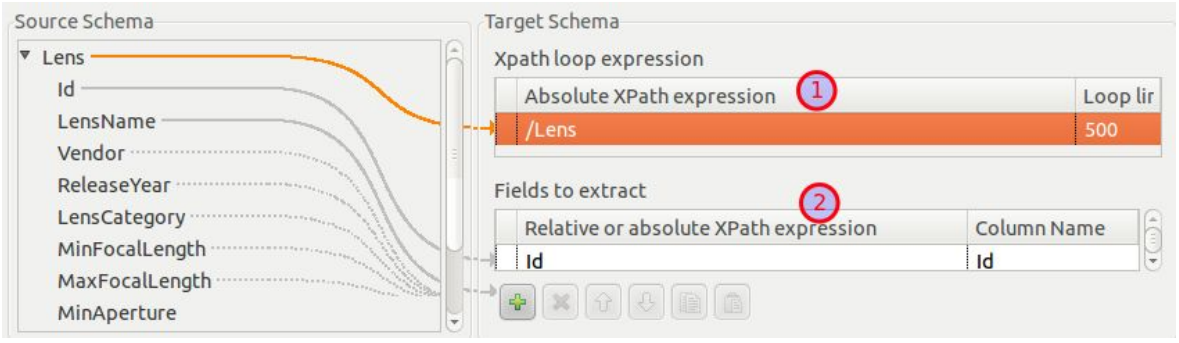


First we create the entity definition for the input to Talend DI:



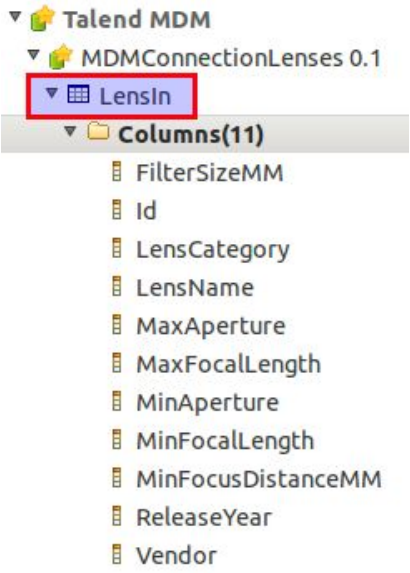
Click **Next** twice, then the wizard allows you to define an XML mapping:

1. Drag and drop **Lens** into the **Absolute XPath expression** section.
2. Mark all the other fields and drag and drop them into the **Fields to extract** section.

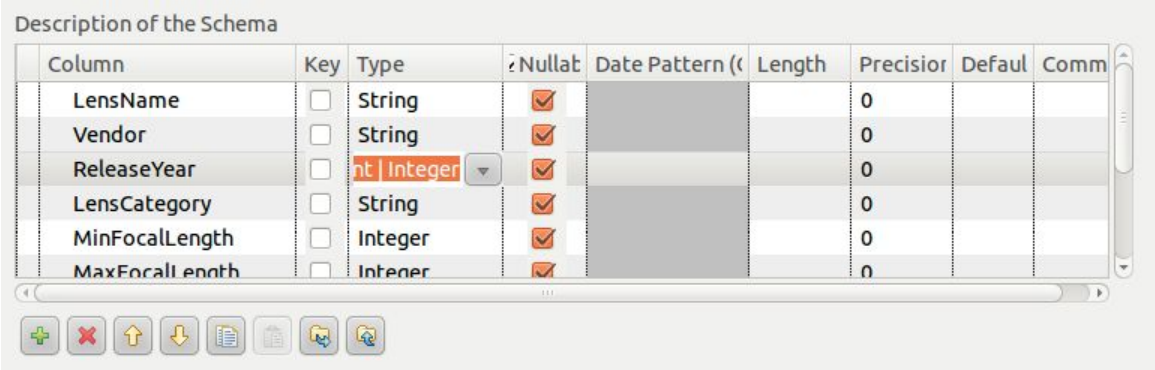


Click **Next**. Now you have a chance to check the schema and amend it if necessary. Once you are happy with it, click **Finish**.

Right click on the **LensIn** node in the repository tree and choose **Read Entity**:

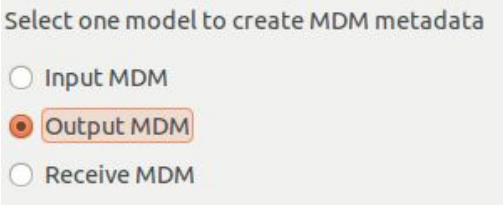


Click **Next** several times until you arrive on the **schema** definition dialog. Adjust the column types where necessary:



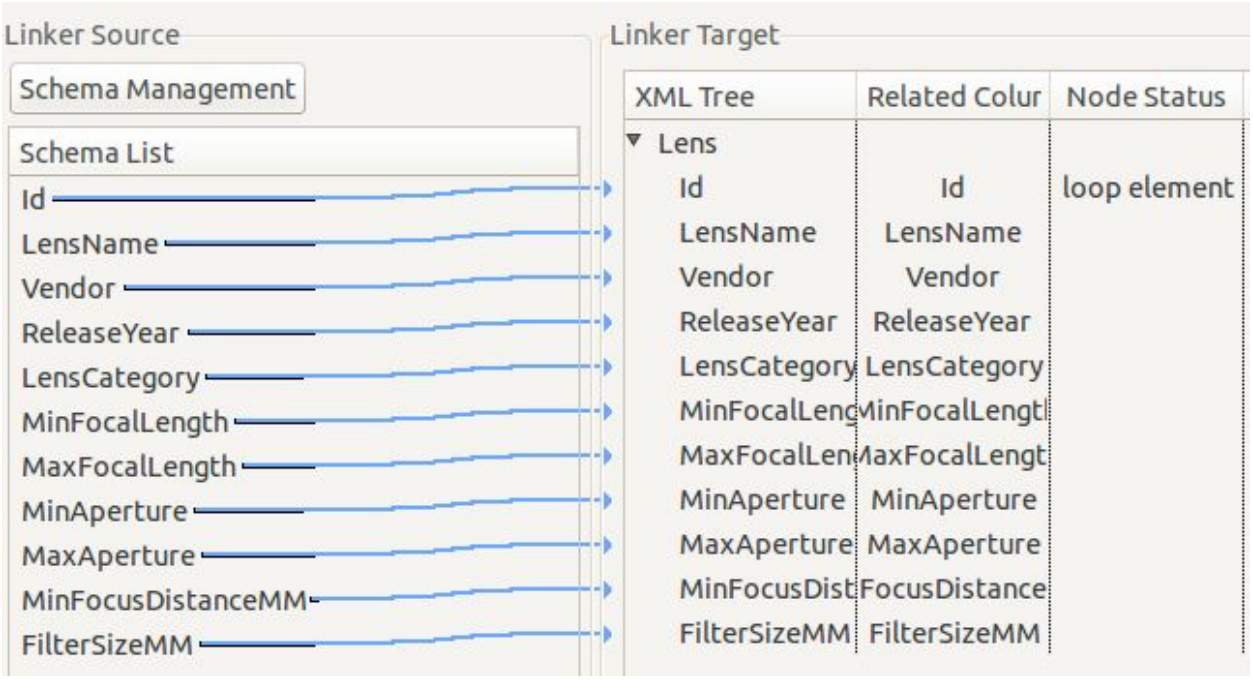
Then click on **Finish** and confirm to propagate the changes. (Note: The very first time you set this up, you might not get to the schema screen. Simply right click on the **LensIn** node and choose **Edit Entity** then).

Next we will create the definition for the **MDM output**. Right click again on **MDMConnectionLenses** and choose **Retrieve Entity**. This time choose **Output MDM** and click **Next** twice.



The wizard will then allow you to set up the mapping. Please note that the mapping shown is based on what Talend MDM Studio knows about the data model (as you can see both sides look identical). The **Schema List** represents the fields in our data integration **job**, the **Linker Target** list represents the fields/elements in our data model.

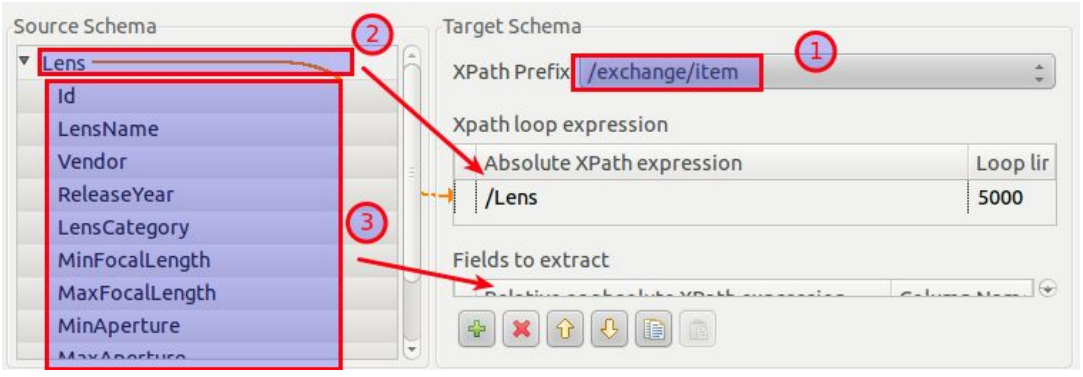
In our case the mapping is simple, because we will make use of all the fields. If you have to adjust the schema for some reason, you can do so via the **Schema Management** button.



Click on **Finish**.

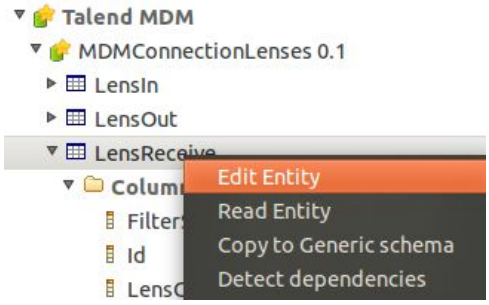
Finally, we will set up the Metadata for the MDM receive option:

1. Right click again on **MDMConnectionLenses** and choose **Retrieve Entity**. This time choose **Receive MDM** and click **Next**.
2. Set **XPath Prefix** to **/exchange/item**. Then drag and drop the **Lens** element from the **Source Schema** on the first row of the **Absolute XPath loop expression**. Next mark all the remaining source schema fields and drag and drop them on **Fields to extract**.

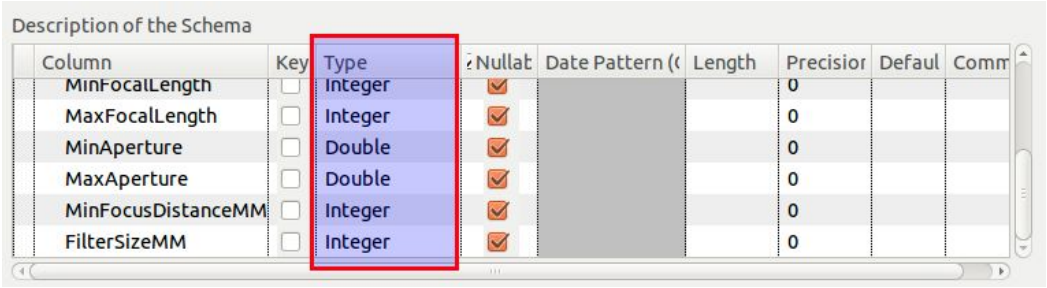


3. Click on **Finish**.

4. Right click on **LensReceive** in the **Talend DI Metadata** tree and choose **Edit Entity**:



5. Click several times **Next** twice. Now you have to opportunity to correct the automatically generated **Schema**. In our case, all is fine as it is … no changes required:



6. Click **Finish** and agree to propagate the changes.

Load data into the MDM hub

All the examples assume that you are familiar with Talend DI. I will not discuss in detail on how to create the jobs, but mainly provide a brief description.

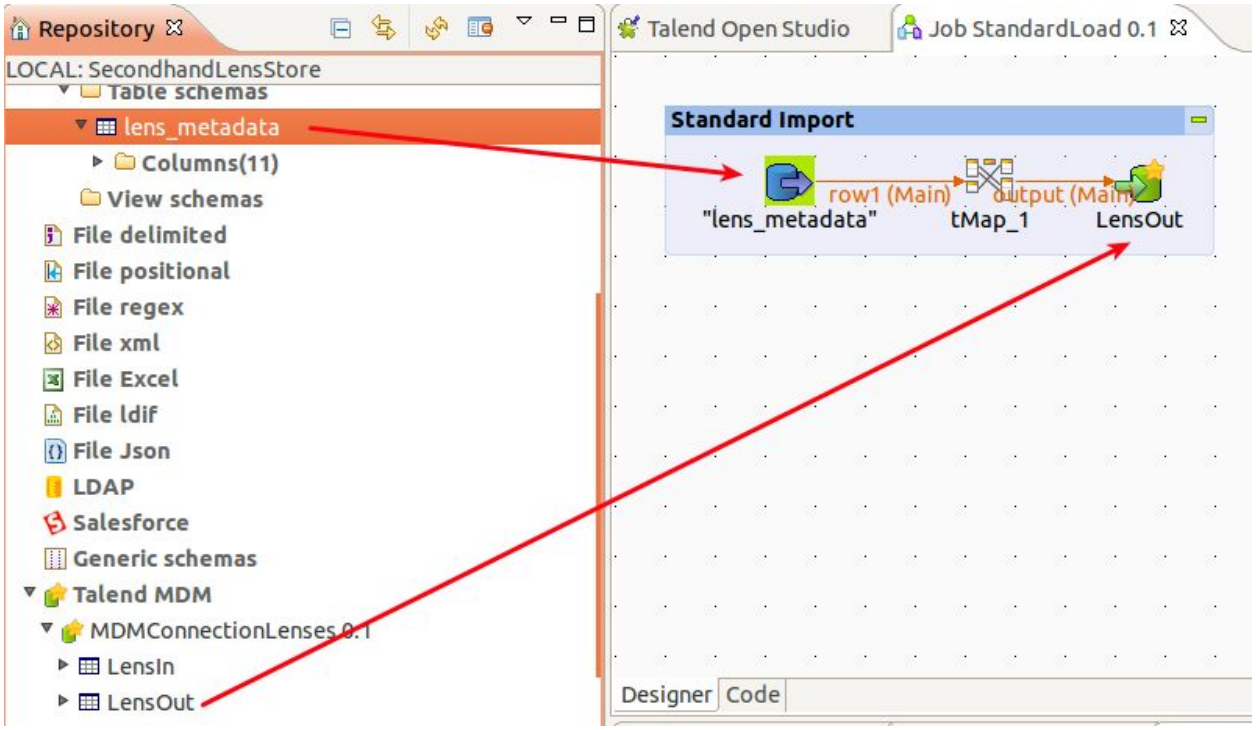
NOTE: All data loaded into the MDM server has to be in XML format!

IMPORTANT: The attribute names of the MDM **data model entity** are **case sensitive**! So id and Id are two different attributes. Make sure that you pay attention to this as this will avoid errors.

IMPORTANT: If your business entity has an auto increment key, then this key will be generated automatically, hence do not attempt to import it.

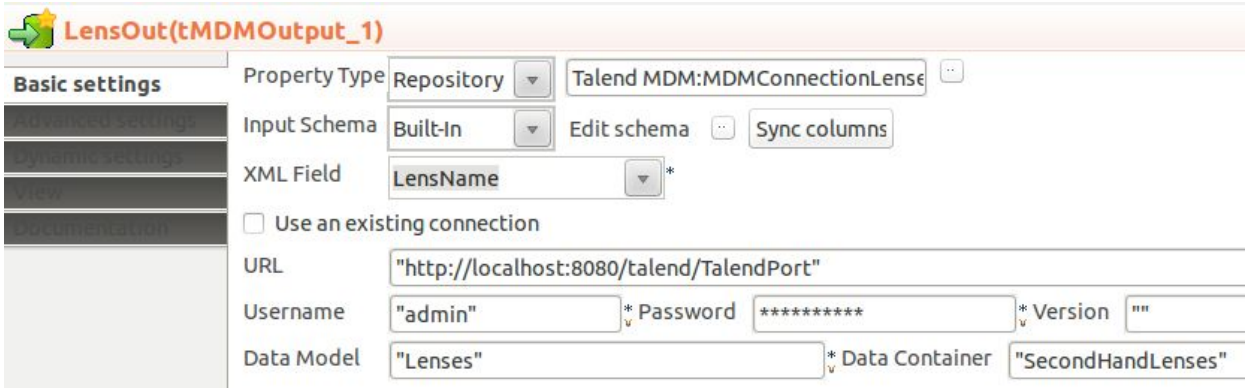
Standard load

Create a new job called **StandardLoad**. This very simple example will illustrate loading a small data set from a PostgreSQL database:

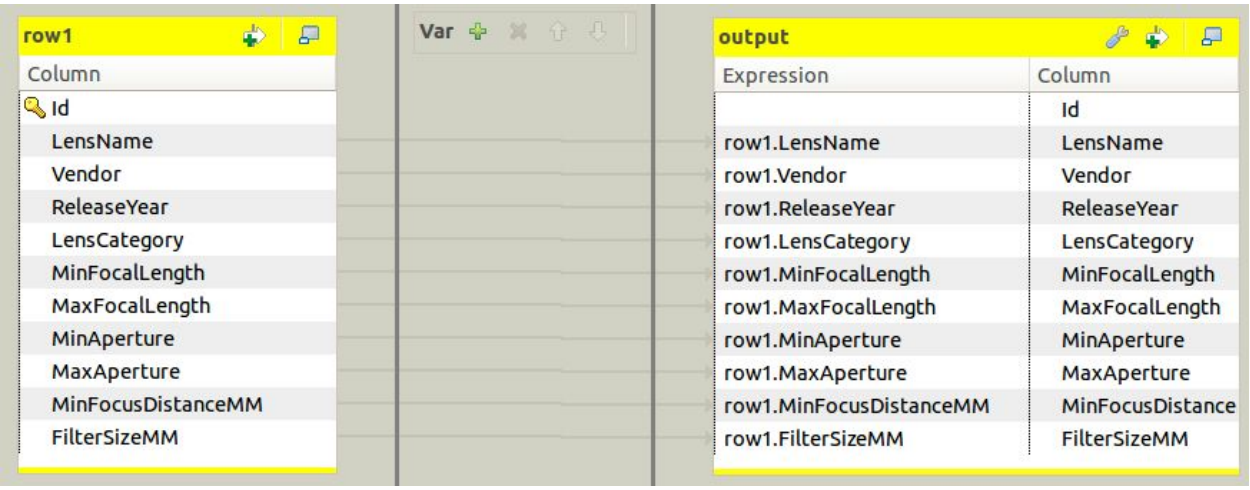


This example can be created extremely quickly:

1. Just drag and drop the **lens_metadata Metadata** connections onto the canvas. When prompted for the component type, choose a suitable input component.
2. Add a **tMap** component
3. From the **Metadata** repository drag and drop the **LensOut** entity onto the canvas. Connect all three components as shown on the screenshot above. The component settings are all prepopulated already, so it is all set up for us!



4. Double click on the **tMap** component and map the fields as shown below. We do not import the Id (first of all it is not populated any ways [we will update this field later on with the Talend MDM Ids] and second of all Talend MDM will create the Ids for us):



Execute the job now.

The next step only applies if you did not follow the MDM metadata connection setup in this tutorial or are not using a MDM metadata connection:
In the **tMDMOutput** component settings provide all the connection details for the MDM server, the data model and the data container.

This component has a built-in feature to create XML based on the input row. To access this feature, double click on the component in the design area.
Create the XML structure:

1. Double click on the component in the design area.
2. The XML root tag has to have to same name as the **data model entity/business entity** you want to populate! So rename it accordingly. Note: This is **case sensitive**!
3. Drag and drop the source columns onto the XML root tag:
   Specify how you want them to be added to the XML structure. The sub-elements here represent the **entity attributes**. They are **case sensitive**!
4. Define one element as the **loop element**. In our case, it is the id element. Right click on the element and choose **Set loop element**.
   Click **OK**.
5. Next we have to change the schema. Basically, now that we have our XML structure defined, we end up having only one column left to handle. In the **Component settings** click on the edit button ("...") and delete all columns apart from the first one. Change the name of the first column to something like xmlRecord.
6. In the **component settings** specify **xmlRecord** as the **XMLField**.

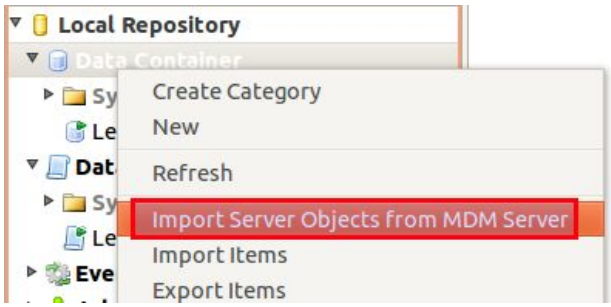
Checking the data container records via the MDM Server web interface

Now let's check if our job worked properly. Login to the MDM website and search for the lens records:

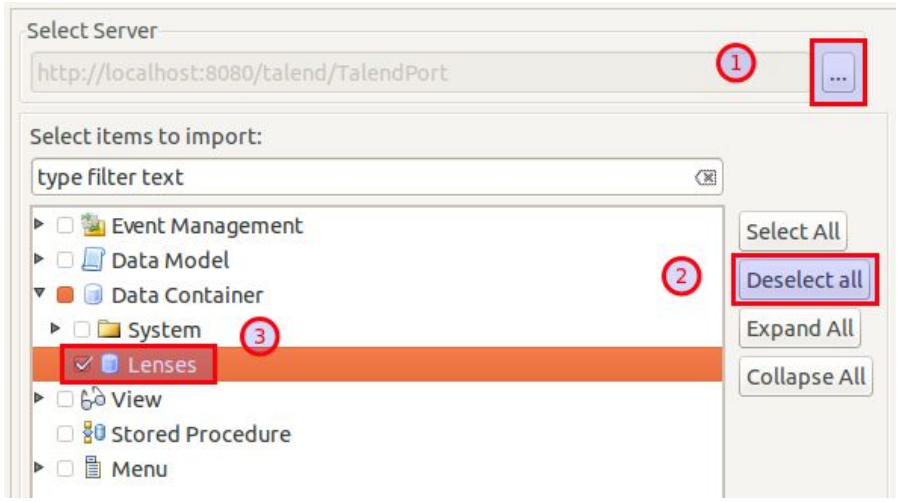
As you can see our database records show up on the MDM server. Note that your ids might be different from the ones shown on the screenshot.

Checking the data container records via Talend MDM Studio

You can also check the records from within **Talend MDM Studio**. To do so, **right click** on

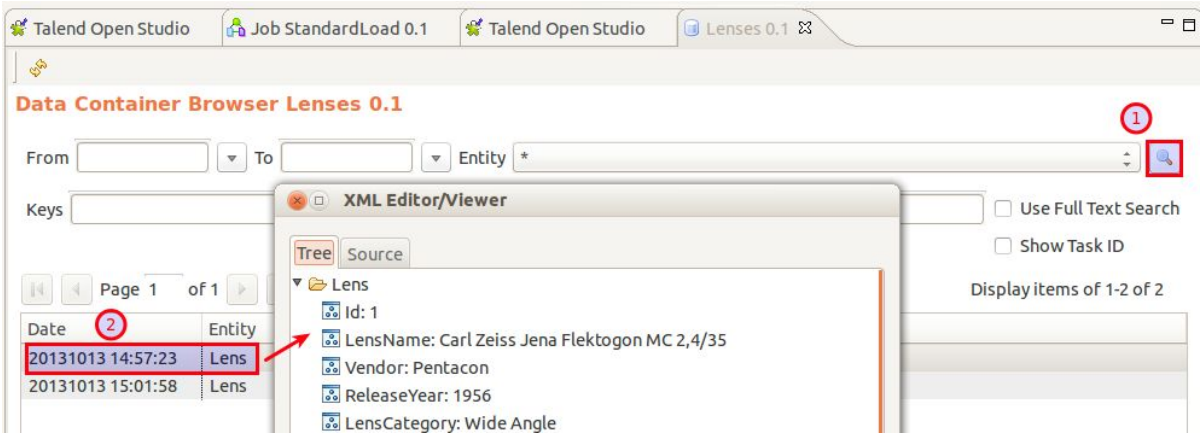


On the next screen click the ellipsis button [...] and choose your **MDM Server**. Once the server repository is loaded, click **Deselect All** and then expand the **Data Container** node and tick **Lenses**:



Click **Finish**.
Now you have the same dataset available in the **Talend MDM Studio** as on the **Talend MDM Server**.

Next double click on **Lenses** in **Local Repository > Data Container**. Confirm the Server selection. Then the **Data Container Browser** will open. As we have only a few records loaded, hit the **Search** button straight away without specifying any search criteria.
Entity records will be listed below. Double click on one row to retrieve the details, which will be shown in the **XML Viewer**:



Checking the data container records via your database's query client

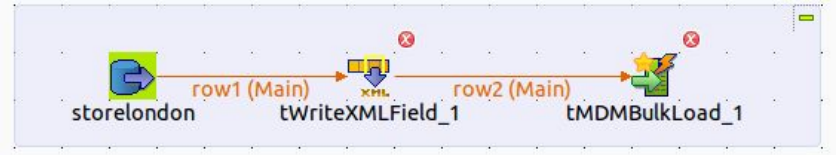
The third option of course is to check the records directly in the query client that is supplied with your database package.
The embedded Talend MDM Server database eXist comes with a web interface which allows you to conveniently analyze the records without having to set up an additional query client.
In case you are storing your records in a standalone RDBMS you are quite likely familiar with the dedicated query client, so I will not cover this here.

Bulk load

Note: **tMDMBulkLoad** has some limitations when bulk loading an XML database - see Component Guide for details.

Create a new job called **BulkLoad**. I will only provide a short description here, as the process is similar to the standard load.

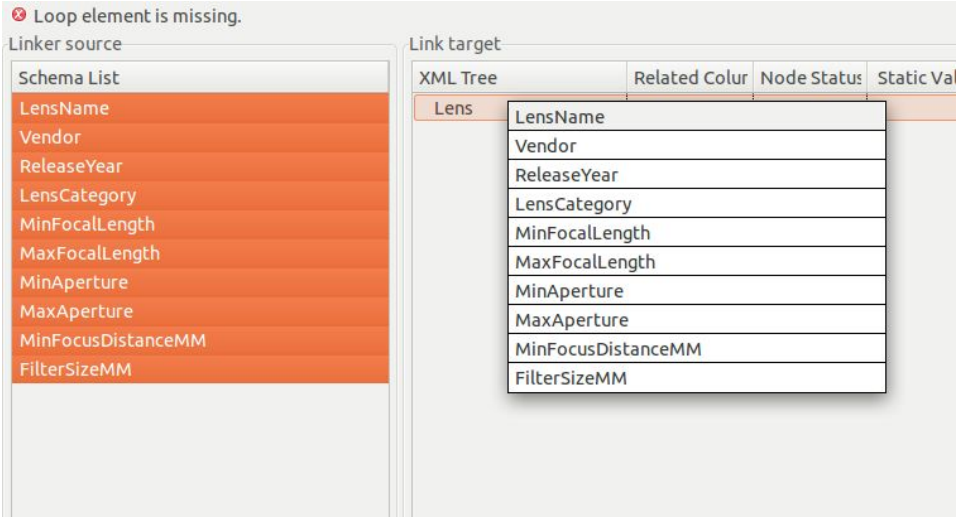
The main difference to the standard load is that the **tMDMBulkLoad** component does not come with a built-in XML transformation feature, hence we have to use a dedicated XML component (**tWriteXMLField**) beforehand.
You can drag and drop the **lens_metadata** MDM definition as a database input component onto the design canvas. Add the **tWriteXMLField** and **tMDMBulkLoad** components via the **Palette**. Line them up and connect them as shown in the screenshot below:



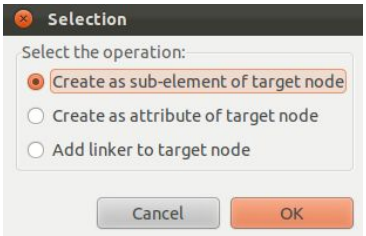
Let's talk about some details:

1. In the **tWriteXMLField** component tab click on **Configure XML Tree**:

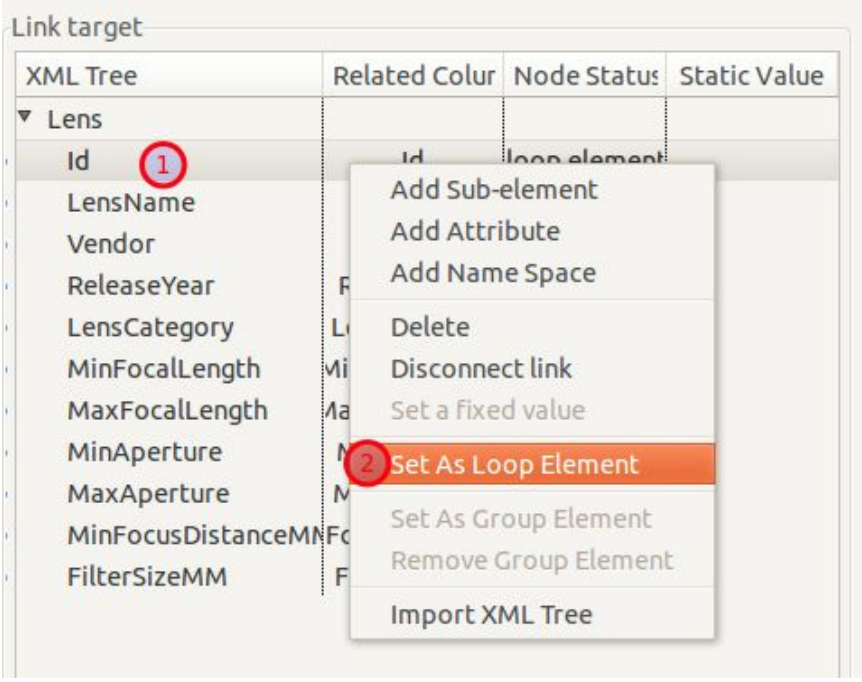
2. In the new dialog rename the XML **rootTag** in the **Link target** (**on the right hand side**) to **Lens** (our business entity name).
3. Mark all attributes on the left and drag and drop them onto the **Lens** row on the right hand side:



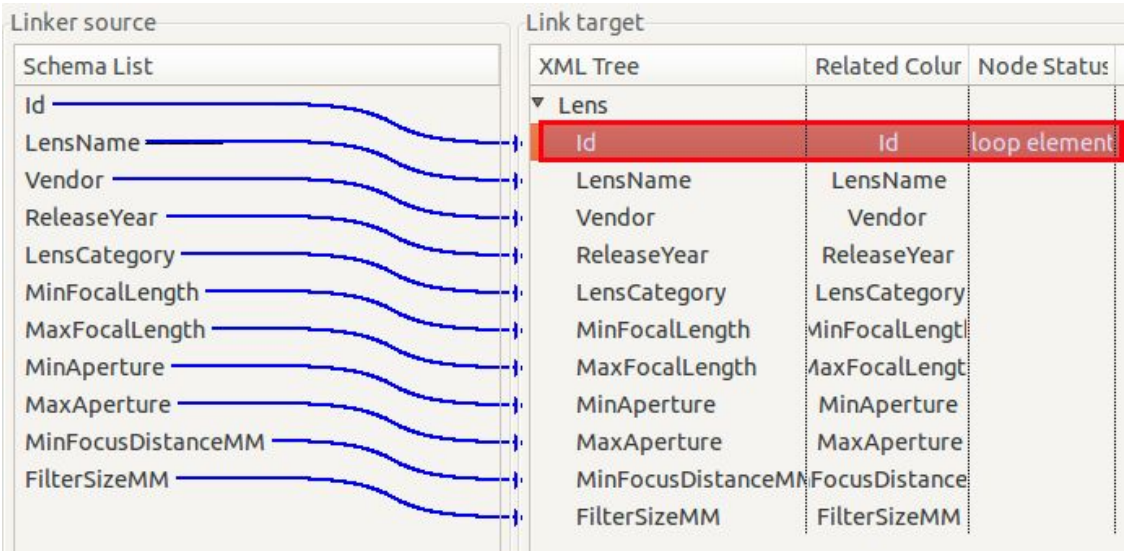
4. Tick **Create as sub-element of target node** and click **OK**:



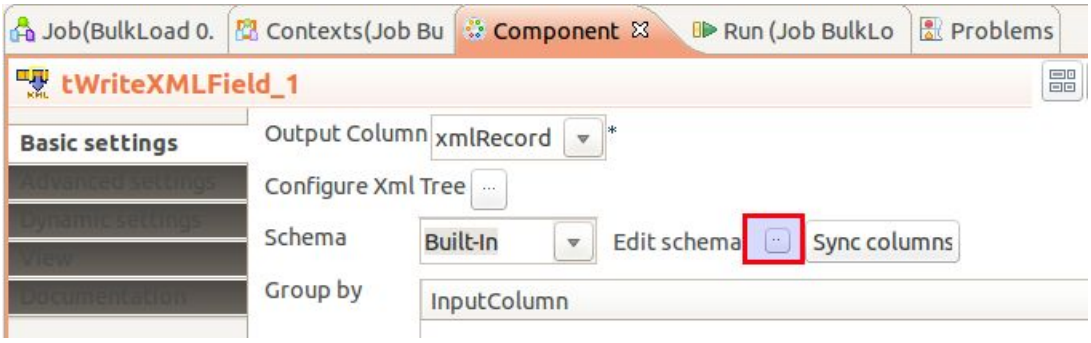
5. Mark the LensName row by clicking on the very left side. Then right click and choose **Set As Loop Element**:



6. The mapping should now look like this:

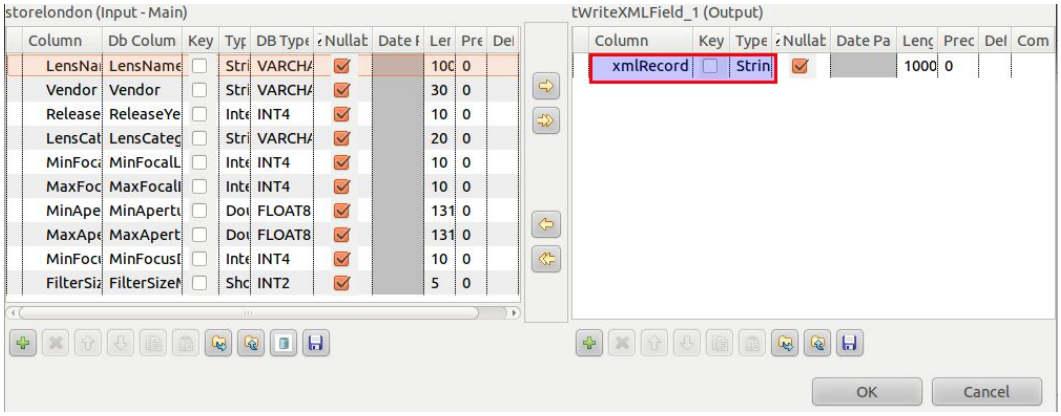


7. Click **OK** to close the dialog. Next click on **Edit Schema** in the **Component** tab:



8. All our input columns get basically folded into one column which holds the XML. Hence

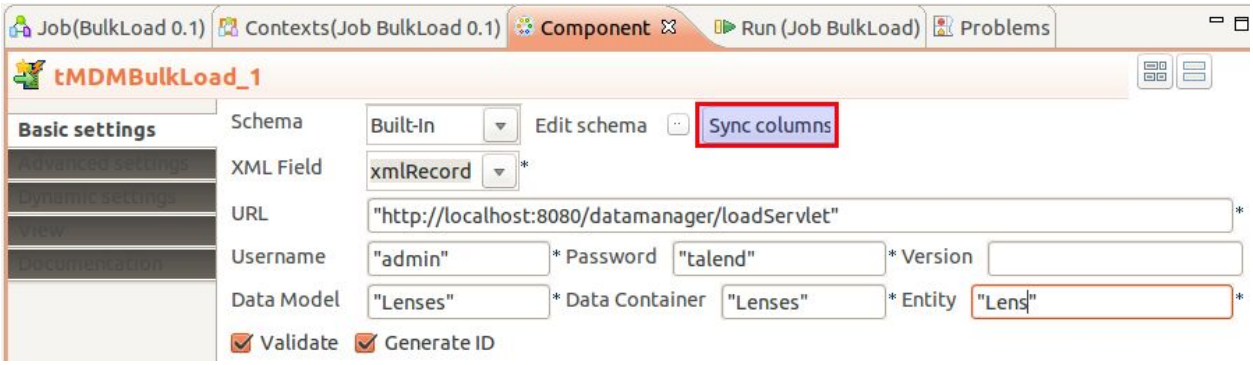
we have to change the schema accordingly. We will only have one output column, therefore delete all output columns (if there are any) and add a new one called **xmlRecord** of type **String** and change the **Length** to a larger value (or just leave it empty):



Once finished click **OK**. You will see the the **Output Column** is now set to **xmlRecord** in the component settings.

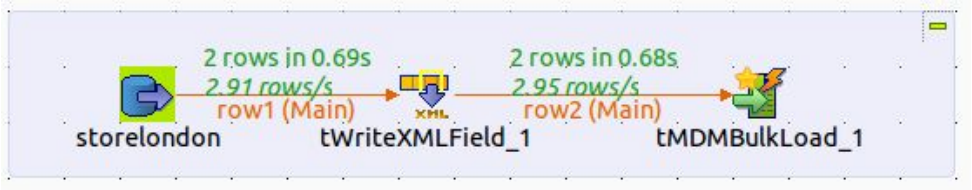
9. Click **OK**. A pop-up window will come up … confirm that you want to propagate the changes.

Instruction for the **tMDMBulkLoad** component:

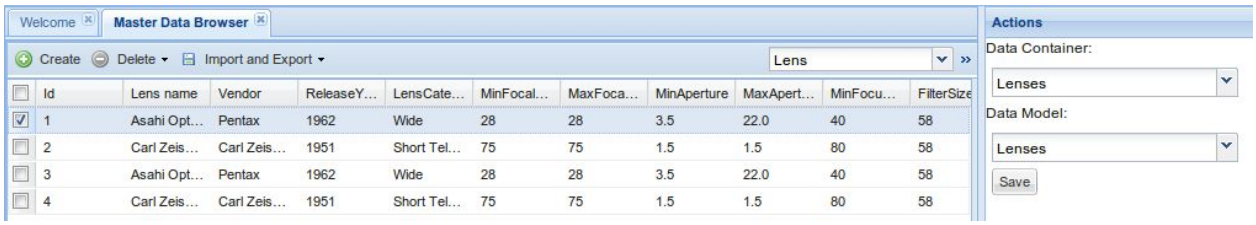


1. Click **Sync columns** to source the schema definition from the row.
2. Provide all the MDM server details. Currently this component does not support sourcing the connection details from the **Talend DI Metadata**, hence you have to fill out all the required configuration fields.
3. Make sure that you tick **Validate** so that the data is validated against the MDM data model.
4. If your business entity has an auto-incremented key (which is true for our example), tick **Generate ID**.

Once you are finished, run the job:



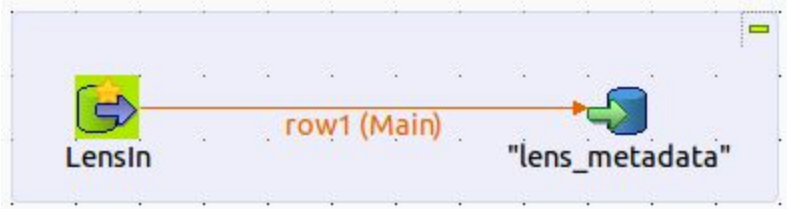
All should execute successfully. If you do a search via the web interface for all **Lens** records, you should now see these new records:



For the purpose of this tutorial we imported the same records twice, which should never happen in a normal setup. Also, we see a clear problem with our business entity: The lenses should be unique. Normally we would set a unique key on the lens name, but as the lens name is a longer string and can contain special characters, it is not possible to do this in Talend MDM.
Theoretically you can set a unique key on it and not use a dedicated Identifier, but you will soon release that Talend MDM is not happy with enforcing unique keys on strings with special characters.
There is a solution though: You can create a dedicated DI job which checks each time the uniqueness of the lens names before a new record gets inserted using Talend MDM triggers - more about triggers later on.
Delete the duplicated records via the web interface for now.

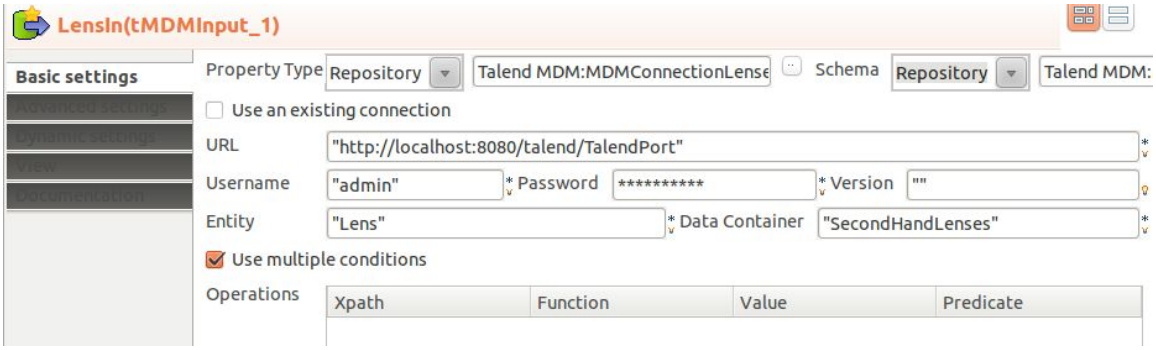
Export data from the MDM server

Create a new job called **Export**. The purpose of our export job is to basically update the existing records in the external database with the keys that the Talend MDM hub created for them. In our case, this is a one-off operation. After this, we only intend to change records via the Talend MDM web interface and have another DI job update the external DB.

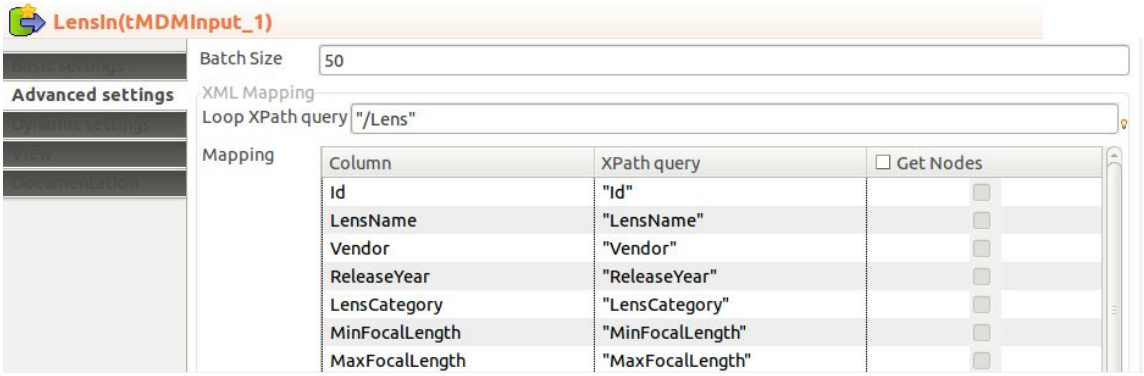


1. Drag and drop the **LensIn** metadata schema on the design canvas. This will automatically create a **tMDMInput** prefilled with all the Talend DI Metadata details:

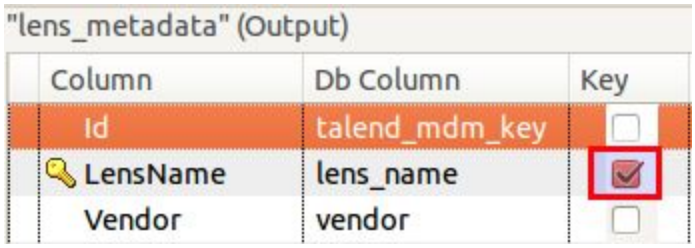
2. Leave **Use multiple conditions** ticked. Delete the default entry if one shows up in the operations table. The **Operations** table can remain empty!



3. Click on **Advanced Settings**. Note that the **Loop XPath query** is already specified. This will normally be your **business entity** name. If you do not use the **Talend DI** Metadata definition, make sure you always add the preceding **/** when specifying the XPath. As we use **Talend DI** Metadata, we don't have to worry about this. You can even see below that attributes have been already defined (Note: This is **case sensitive**!). We do not want to retrieve the XML tags, hence we leave **Get Notes** unticked:



4. From the **Repository > Metadata** drag and drop **lens_metadata** on the design area and choose a suitable DB output component. Create a main row from **LensIn** to this component.
5. Click on the **DB output** component and then on the **Component** tab. Set **Action on data** to **Update**.
6. Click on **Edit Schema**, then choose **Change to built-in property**. For the **Output** dataset change the key to **LensName**:



As you are aware, currently we can only update the records in the external DB based on the lens name. After we ran this job, we will be able to use the Talend MDM key (Id) in other data integration jobs.
Click **OK**.

7. Tick **Die on error**.
8. **Run** the job.
9. In your favourite query client, check if the records in the external DB show the Talend MDM keys:

```
storelondon=# SELECT lens_name, talend_mdm_key FROM lenses.lens_metadata;
                    lens_name                     | talend_mdm_key
--------------------------------------------------+----------------
 Asahi Optical Co. Super Takumar 28mm f/3.5-22.0 |              1
 Carl Zeiss Jena Biotar 75mm f/1.5               |              2
(2 rows)
```

Real-time data integration / propagation

Talend MDM comes with powerful process and trigger features, which allow in example instant propagation of changes to other systems. We will create one job which propagates updates and new records and another job for the deleted records. The final section will try to combine these jobs into one.

Processes

A process consists of various steps (plugins) which are executed in the specified order - so it behaves pretty much like a workflow.
You can quickly create simple processes and triggers by right clicking on the data integration jobs in the MDM perspective and choosing the auto generate feature.

Before-Saving and Before-Deleting Processes are special type of processes which can be used to perform an action before the data records gets saved or deleted.

Triggers

You are probably familiar with triggers from a database context. The concept of triggers on the **Talend MDM server** is quite similar: A certain event can trigger a certain process.
Following events exist:

- CREATE
- UPDATE
- LOGIC_DELETE
- PHYSICAL_DELETE

You can define various trigger conditions which can be based on various fields from the Update Report (You will learn more about the Update Report a bit later on) :

Trigger XPath Expressions Overview

XPath	Description	Possible Values
Update/Source	what initiated this event	genericUI, ...
Update/OperationType		PHYSICAL_DELETE, LOGIC_DELETE, UPDATE, CREATE
Update/DataCluster	Data Container	
Update/DataModel	Business Model	
Update/Concept	Business Entity	
Update/Item/path	Business attribute which got changed in case of an update	

The Exchange Document

In most cases, you will send an XML document (let's call it Exchange Document) from the trigger or process to your data integration job, which consists of the the **Update Report** and the **actual data record** (in XML format). The root element of this document is **exchange**.

Triggers send this Exchange Document by default (no additional configuration required), non-integrated processes however not. For non-integrated processes you have to combine 2 steps to achieve the same outcome (more about this later on).

Some sections of the Exchange Document look different depending on the action (create, update, delete). If you want to learn more about the XML structure, take a look here.

CRU: New records and updates

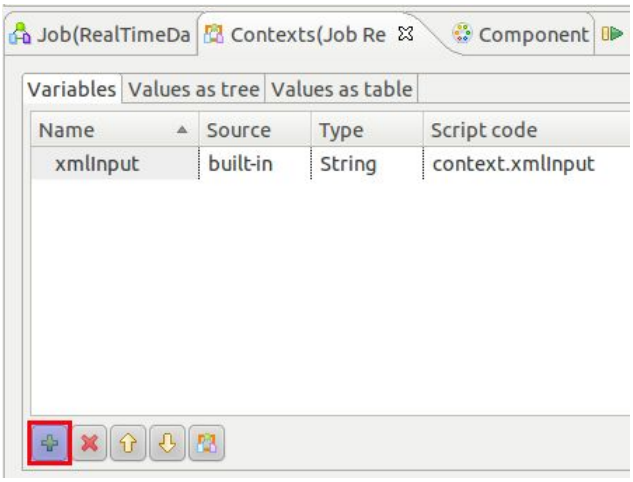
We will create the next two DI jobs "the old way", so that you learn this approach as well. The third example (CRUD Integrated) will be based on "the new way" … the *integrated* way … but more about this later.

Creating the data integration job

Create a new job called **RealTimeDI_CRU**. We will create this extremely simple example:

Let's get started:

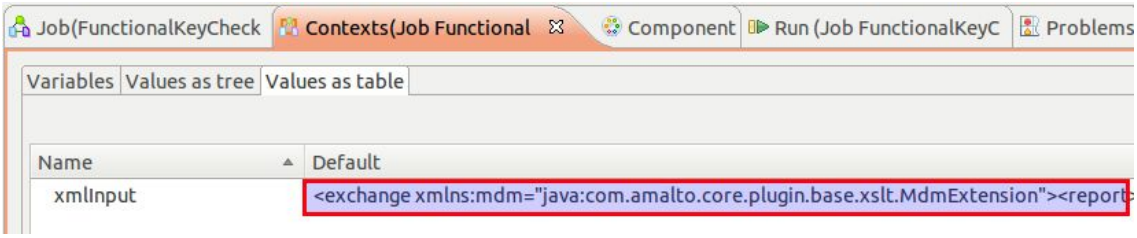
1. From the **Talend DI Metadata** drag and drop the MDM **LensReceive** schema on the design canvas. This will create a **tMDMReceive** component which is designed to receive MDM XML records from TOS MDM triggers and processes. [* If you are not using **Talend DI Metadata** please find some instruction at the end of this section].
2. From the **Talend DI Metadata** drag and drop the DB connection **lens_metadata** schema on the design canvas and choose the **database output** component.
3. Connect the two components as shown in the screenshot above.
4. The **tMDMReceive** component will receive an XML document from the MDM server. This XML document - the Exchange Document - consists of the Update Report and the actual data record. As we do not know the name of this file beforehand, we will create a context variable so that it can be supplied at runtime. Create a **context variable** called **xmlInput**:



5. Click on the **Values as table** tab and insert the following **XML fragment** into the **Default** value field. This is actually a specially prepared XML document which contains the **Update Report** AND the actual **data record**. It is the same XML fragment which will be output by the MDM trigger or process which we will create later on:

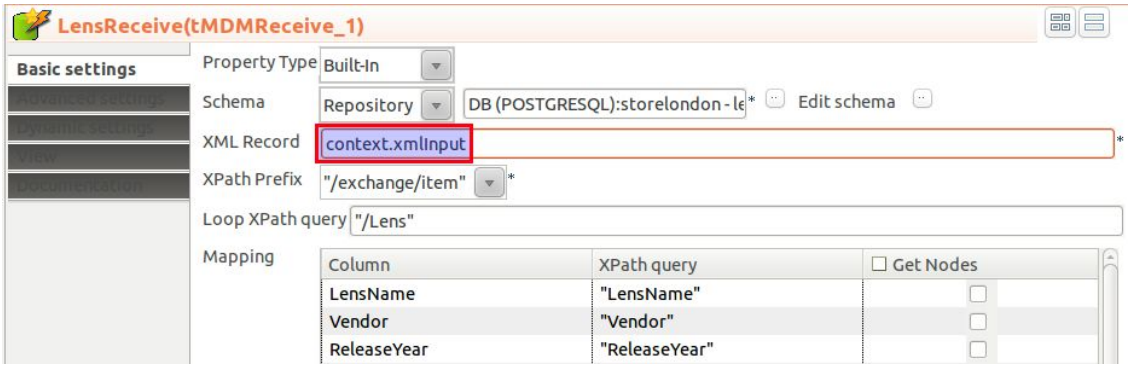
```
<exchange
xmlns:mdm="java:com.amalto.core.plugin.base.xslt.MdmExtension"><report><Update
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><UserName>administrator</
```

```
UserName><Source>genericUI</Source><TimeInMillis>1336311970956</TimeInMillis>
<OperationType>CREATE</OperationType><RevisionID>null</RevisionID><DataCluster>
SecondHandLenses</DataCluster><DataModel>Lenses</DataModel><Concept>Lens</Conce
pt><Key>39</Key></Update></report><item><Lens
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><Id>39</Id><LensName>Asah
i Super Takumar 55
1.8</LensName><Vendor>Pentax</Vendor><ReleaseYear/><LensCategory/><MinFocalLeng
th/><MaxFocalLength/><MinAperture/><MaxAperture/><MinFocusDistanceMM/><FilterSi
zeMM/></Lens></item></exchange>
```

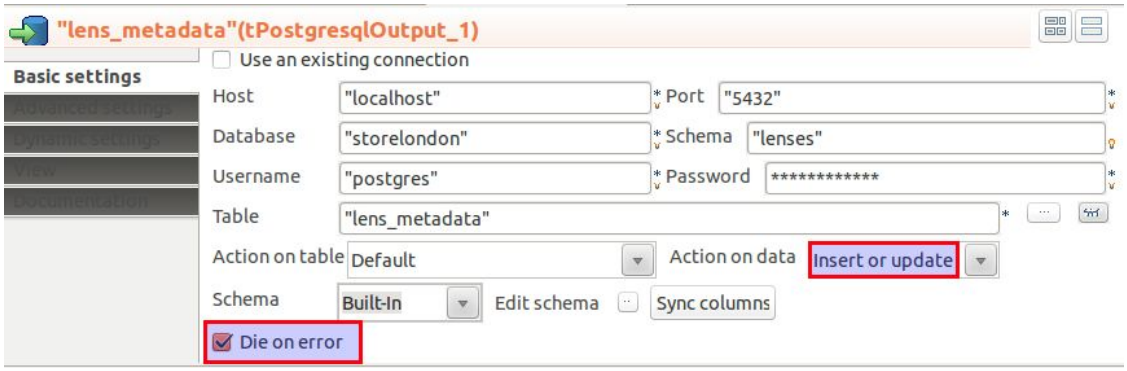


This will allow us to test our DI job independently!

6. In the **LensReceive Component** settings specify **context.xmlInput** for **XML Record**. All the other settings are already provided by the **Talend DI Metadata** which we set up before:



7. Next, click on the **database output component**, which is already mostly prepopulated. Just set **Action on data** to **Insert or update** and tick **Die on error**:



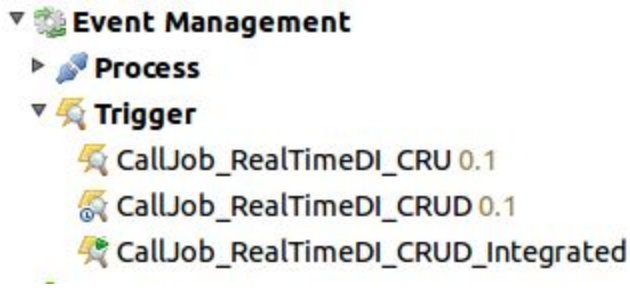
8. **Save** the job and run it.

* In case you have not been following this tutorial all the way through or you are not using the **Talend DI Metadata**, just drag and drop the **tMDMReceive** component from the **Palette** and specify the following:

- Define the **Schema**
- **XML Record**: Set up a job context variable with the same name as the parameter name of the **MDM process** which calls this job.
- **XPath Prefix**: Choose /item in case the XML record is coming from a process and /exchange/item in case the XML record is coming from a trigger.
- **Loop XPath query**: The business entity. **IMPORTANT**: This has to be prefixed with a forward slash!
- **Mapping**: Define how the attributes of the business entity match to the **schema**.

Creating the trigger

1. In the MDM perspective, right click on the job in the repository view and choose **Generate Talend Job Caller Trigger**.
2. In the **Event Management > Trigger** section you will find the recently created trigger. Double click on it to open it (if it is not already open).



3. Most settings are already preconfigured, we only have to make a few changes: In the **Service** section change the name of the parameter in the XML fragment to **xmlInput** (which happens to be the same name as you assigned to the context variable in the data integration job):

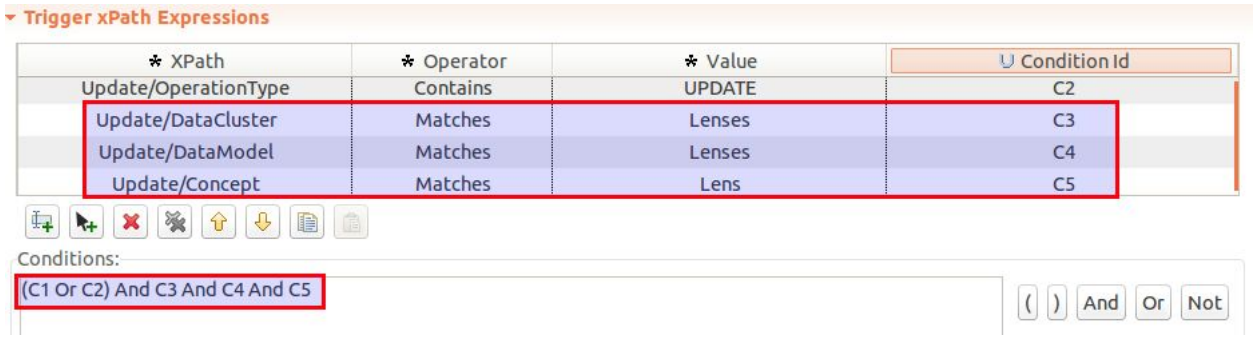
```
<configuration>
  <url>ltj://RealTimeDI_CRU/0.1</url>
  <contextParam>
    <name>xmlInput</name>
    <value>{exchange_data}</value>
  </contextParam>
</configuration>
```

4. Remove the Delete condition from the **Trigger XPath Expressions** and the following conditions, so that we link the trigger to the right data container, data model and entity:

XPath	Operator	Value	Condition
Update/DataCluster	Matches	SecondHandLenses	C3

Update/DataModel	Matches	Lenses	C4
Update/Concept	Matches	Lens	C5

5. Set the conditions to: **(C1 Or C2) AND C3 AND C4 AND C5**



6. **Save** the trigger and deploy it to the MDM server.

Testing

Via the Talend MDM web interface add and update a few records and check if these changes are reflected in the external DB. Keep an eye on the JBoss log.

CRUD: Routing in the DI job

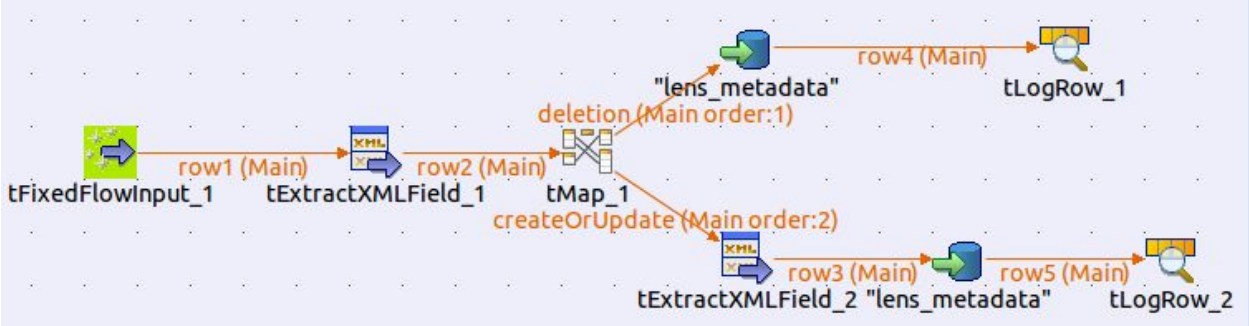
Normally you would use processes and/or triggers to route updates, new records, deletions to certain jobs.

But there is also a way to achieve the same functionality within your data integration job, and it is indeed a very elegant solution!

This is very useful in example for synchronizing the data as you can implement the whole logic within one job.

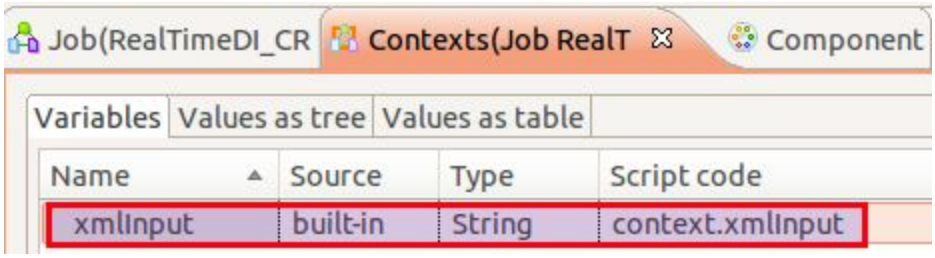
The way to achieve this is to send the **Exchange Document** to your DI job. This one checks then what kind of action (*update, creation, deletion*) was performed and takes actions accordingly.

Data integration

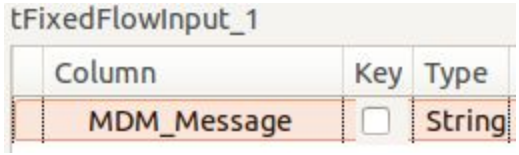


Let's get started:

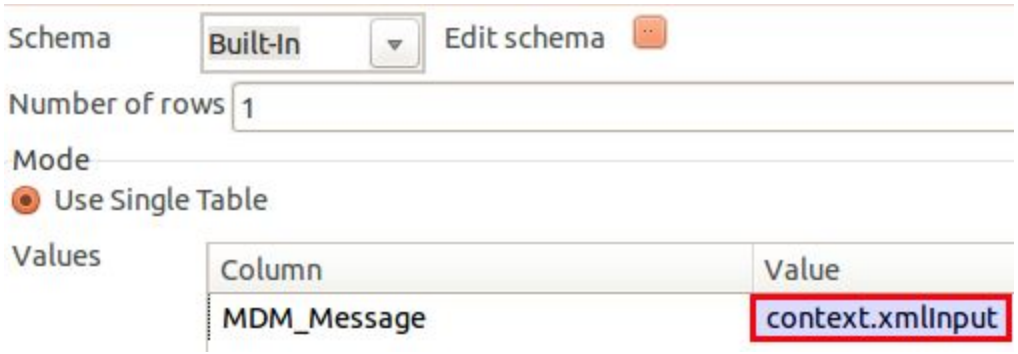
1. Create a new job called **RealTimeDI_CRUD**.
2. Add a context variable called **xmlInput** of type String:



   This context variable will be used to store the **Exchange Document** which originates from a process or trigger.
   You can provide a value as well. Click on tba **Values as table** and copy one of the Exchange Documents listed in *The Integrated version of CRUD > Exchange Document* in there, which makes it then easy to test the DI job.
3. Add a **tFixedFlowInput** component to the canvas. Click on **Edit Schema** in the **Component** settings. Create a new column called **MDM_Message** of type String:



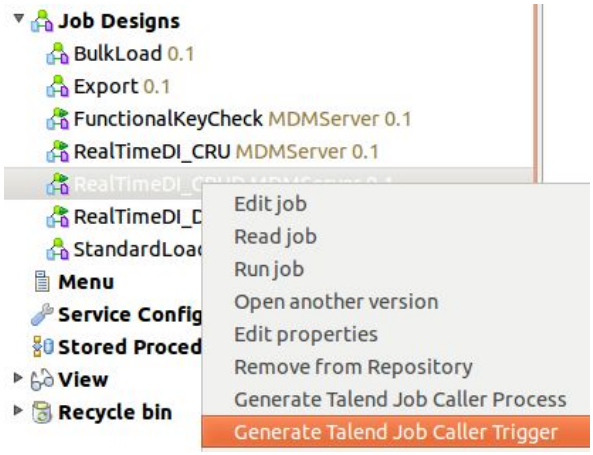
4. In the **Values** table click in the **Value** field next to **MDM_Message**: Press **CTRL+SPACE** and choose the context variable we defined earlier on: **context.xmlInput**.



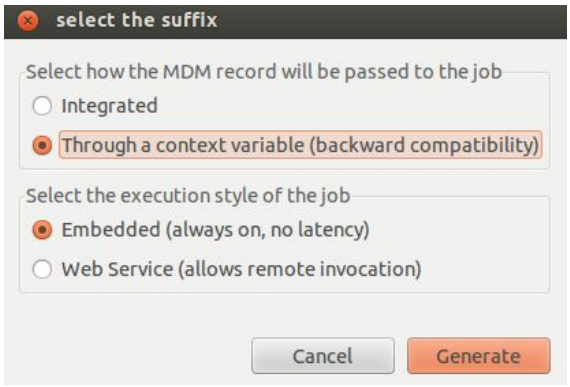
5. The next steps are exactly the same as described in *The Integrated version of CRUD* (We are just using here the tFixedFlowInput instead of the tMDMInput component).

Creating the trigger

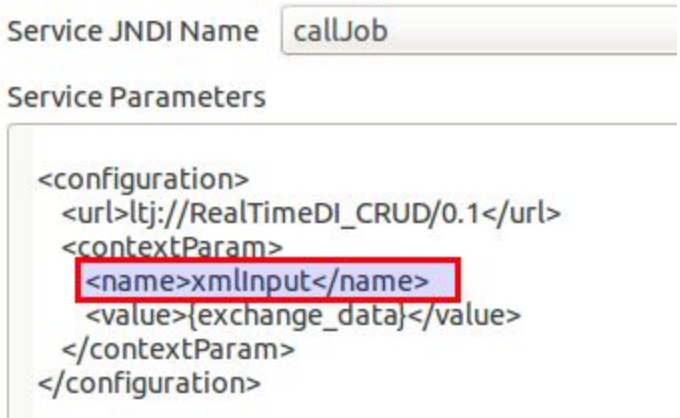
1. Right click on **RealTimeDI_CRUD** and choose **Generate Talend Job Caller Trigger**:



2. In the next dialog choose **Through a context variable (backward compatibility)** and **Embedded (always on, no latency)**:



3. Under **Event Management > Trigger** you will now find a trigger called **CallJob_RealTimeDI_CRUD**. Double click on it.
4. In the **Service** section change the name of the parameter in the XML fragment to **xmlInput** (which happens to be the same name as you assigned to the context variable in the data integration job):

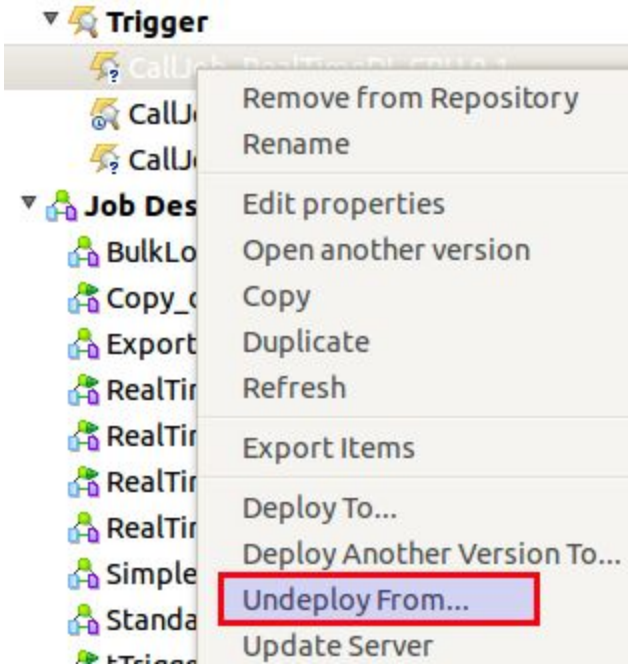


5. Add the following to the **Trigger XPath Expressions**, so that we link the trigger to the right data container, data model and entity:

XPath	Operator	Value	Condition
Update/DataCluster	Matches	SecondHandLenses	C4
Update/DataModel	Matches	Lenses	C5
Update/Concept	Matches	Lens	C6

6. Set the conditions to: **(C1 Or C2 Or C3) AND C4 AND C5 AND C6**

7. **Save** the trigger.
8. Undeploy the trigger we uploaded in the previous exercise: Right click on **CallJob_RealTimeDI_CRU** and choose **Undeploy From...**



9. Now deploy **CallJob_RealTimeDI_CRUD** it to the MDM server.

Testing

Via the Talend MDM web interface add, update and delete a few records and check if these changes are reflected in the external DB. Keep an eye on the JBoss log.

The Integrated version of CRUD

Integrated VS non integrated - what it is all about

The tMDMTrigger* components were only introduced in Talend MDM Studio v5.0. Do you remember that the trigger and process auto-generate function asks you if the process/trigger should be **Integrated** or **Through a context variable (backwards compatible)**? Did you ever wonder what **Integrated** really means and how to get this working?
There are two Talend DI components which do not require any special configuration to have access to the data changes on the MDM hub and are hence **Integrated**:

tMDMTriggerInput: "*Once executed, tMDMTriggerInput reads the XML message (Document type) sent by MDM and passes them to the component that follows. Every time when you save a change in your MDM, the corresponding change record is generated in XML format. At runtime, this component reads this record and sends the relative information to the following component. With this component, you do not need to configure your Job any more in order to communicate the data changes from MDM to your Job.*"**

"*No more context variable is required with the 'Integrated' method. MDM passes the exchange message transparently to the job. This is passed as a DOM Document so it's possible to use tXMLMap to extract elements.*"***

tMDMTriggerOutput: "*tMDMTriggerOutput receives an XML flow (Document type) from its preceding component. This component receives an XML flow to set the MDM message so that MDM retrieves this message at runtime. With this component, you do not need to configure your Job any more in order to communicate the data changes from MDM to your Job.*"**

"*This component works alongside the new trigger service and process plug-in in MDM version 5.0 and higher. The MDM Jobs, triggers and processes developed in previous MDM versions remain supported. However, we recommend using this component when designing new MDM Jobs.*"**

** Source: Talend Open Studio Components - Reference Guide
*** Source: Talend Forum

These components have the word Trigger in their name, which makes it a bit confusing … so you remember that there are Triggers as well in the MDM Studio? Hm, now what does this really mean? Are they the same? No.
As mentioned above, they are part of the *MDM Trigger Service*. The **tMDMTriggerInput** component fetches the whole **Exchange Document** XML document and is fully aware about the related basic XML structure (apart from the data record/entity elements inside the <item>

node), so no further configuration is required. This component doesn't allow you to set any triggers as such (on create, on update, on delete), so it is better to think of this component as a **tMDMReceive component on steroids** than a trigger.
You will still need a simple integrated trigger with your DI job. Basically each time there is a change to the data, the trigger will run and start the job. Just this time you don't have to define a variable to exchange the XML document plus the input component is aware of the main structure of this XML document.

Earlier on we create the **RealTimeDI_CRUD** data integration job. Let's see how we can implement the same logic using the **tMDMTriggerInput** component. The advantage of this approach is that we do not have to define the variable any more which gets exchanged between a trigger or process and the data integration job.

Let's have a detailed look again at the structure of the Exchange Document for each action:

Exchange Document Structure

CREATE

```
<exchange>
  <report>
    <Update>
      <UserName>user</UserName>
      <Source>genericUI</Source>
      <TimeInMillis>1383156369929</TimeInMillis>
      <RevisionID>null</RevisionID>
      <DataCluster>Lenses</DataCluster>
      <DataModel>Lenses</DataModel>
      <Concept>Lens</Concept>
      <Key>1</Key>
      <OperationType>CREATE</OperationType>
    </Update>
  </report>
  <item>
    <Lens>
      <Id>1</Id>
      <LensName>Asahi Optical Co. Super Takumar 28mm f/3.5-22.0</LensName>
      <Vendor>Pentax</Vendor>
      <ReleaseYear>1962</ReleaseYear>
      <LensCategory>Wide</LensCategory>
      <MinFocalLength>28</MinFocalLength>
      <MaxFocalLength>28</MaxFocalLength>
      <MinAperture>3.5</MinAperture>
      <MaxAperture>22.0</MaxAperture>
      <MinFocusDistanceMM>40</MinFocusDistanceMM>
```

```
      <FilterSizeMM>58</FilterSizeMM>
    </Lens>
  </item>
</exchange>
```

UPDATE

```
<exchange>
  <report>
    <Update>
      <UserName>user</UserName>
      <Source>genericUI</Source>
      <TimeInMillis>1383157466851</TimeInMillis>
      <RevisionID>null</RevisionID>
      <DataCluster>Lenses</DataCluster>
      <DataModel>Lenses</DataModel>
      <Concept>Lens</Concept>
      <Key>18</Key>
      <Item>
        <path>Vendor</path>
        <oldValue>Pentax</oldValue>
        <newValue>Pentax Asahi</newValue>
      </Item>
      <OperationType>UPDATE</OperationType>
    </Update>
  </report>
  <item>
    <Lens>
      <Id>1</Id>
      <LensName>Asahi Optical Co. Super Takumar 28mm f/3.5-22.0</LensName>
      <Vendor>Pentax Asahi</Vendor>
      <ReleaseYear>1962</ReleaseYear>
      <LensCategory>Wide</LensCategory>
      <MinFocalLength>28</MinFocalLength>
      <MaxFocalLength>28</MaxFocalLength>
      <MinAperture>3.5</MinAperture>
      <MaxAperture>22.0</MaxAperture>
      <MinFocusDistanceMM>40</MinFocusDistanceMM>
      <FilterSizeMM>58</FilterSizeMM>
    </Lens>
  </item>
</exchange>
```

LOGIC_DELETE

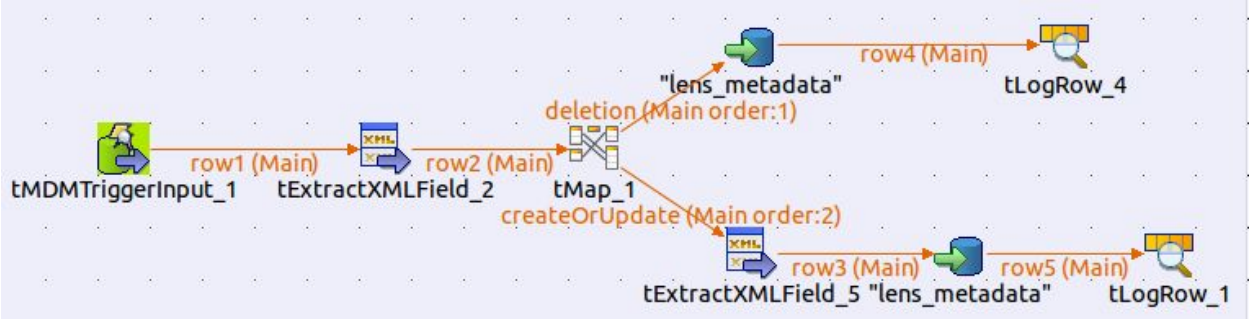
```
<exchange>
  <report>
    <Update>
      <UserName>user</UserName>
      <Source>genericUI</Source>
      <TimeInMillis>1383167579885</TimeInMillis>
      <OperationType>LOGIC_DELETE</OperationType>
      <RevisionID>null</RevisionID>
      <DataCluster>Lenses</DataCluster>
      <DataModel>Lenses</DataModel>
      <Concept>Lens</Concept>
      <Key>1</Key>
    </Update>
  </report>
  <item />
</exchange>
```

PHYSICAL_DELETE

```
<exchange>
  <report>
    <Update>
      <UserName>user</UserName>
      <Source>genericUI</Source>
      <TimeInMillis>1383156401776</TimeInMillis>
      <OperationType>PHYSICAL_DELETE</OperationType>
      <RevisionID>null</RevisionID>
      <DataCluster>Lenses</DataCluster>
      <DataModel>Lenses</DataModel>
      <Concept>Lens</Concept>
      <Key>1</Key>
    </Update>
  </report>
  <item/>
</exchange>
```

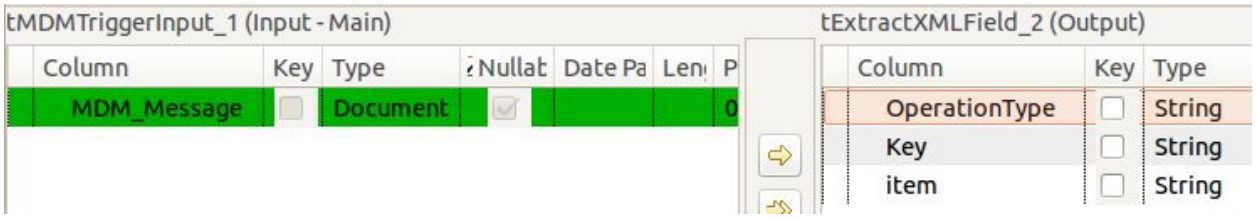
IMPORTANT: The data record does not exist any more at this time, hence it is not included!

Data Integration



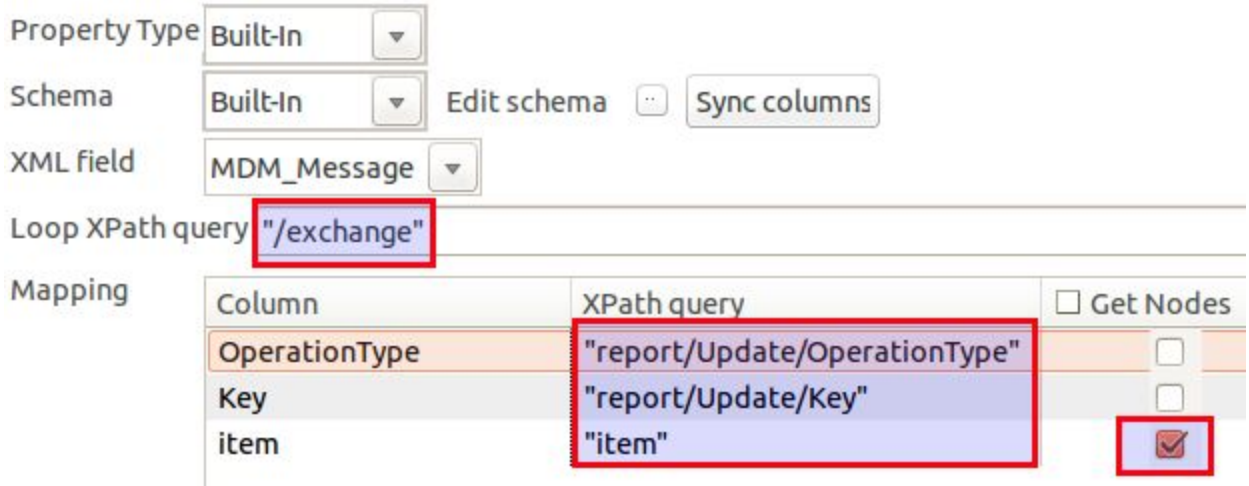
Let's get started:

1. Let's create a new job called **RealTimeDI_CRUD_Integrated**.
2. Add a **tMDMTriggerInput** component. There is nothing to configure here. Maybe take a look at the schema … you will see that there is only one column called **MDM_Message** and it is of type **Document**.
3. Next add a **tExtractXMLField** component. Connect it. Set it up so it looks like shown on the screenshot below. Click on **Edit schema** and define the fields that we want to have in our output dataset:



So here we only want to extract the **OperationType** (CREATE, UPDATE, PHYSICAL_DELETE), the **Key** and the **item** (the actual data record).

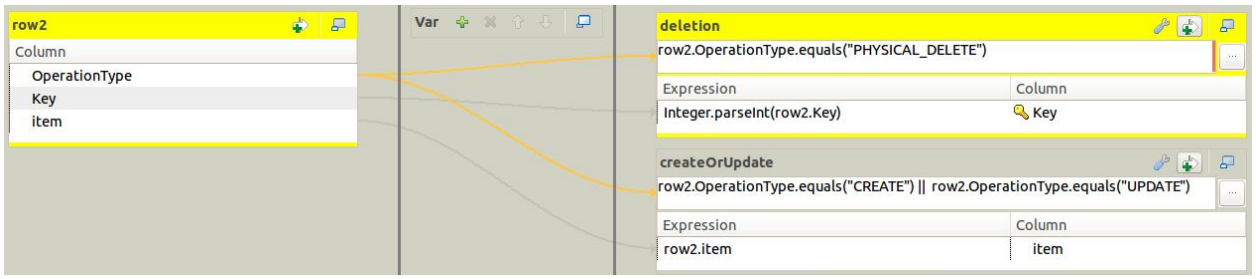
4. Set **XML field** to **MDM_Message**, the **Loop XPath query** to "/exchange", the reason being that we want to get both details from the **Update Report** as well as the **actual data** item. Next provide the **XPath query** values in the **Mapping** table, so in example for OperationType "report/Update/OperationType", for the Key "report/Update/Key" and for the actual data record "item". For "item" tick **Get Nodes**. In this case we want to retrieve the XML fragment to dissect it later on:



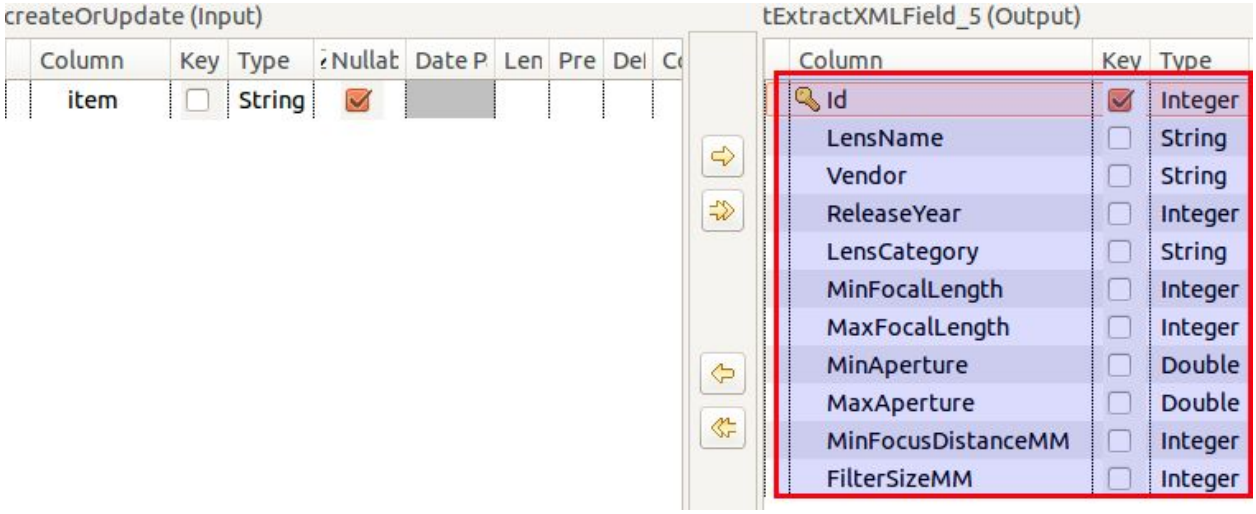
5. Add a **tMap** component. Connect it. Then double click on the **tMap** component and create two output datatsets: **createOrUpdate** and **deletion**.
6. For the **createOrUpdate** dataset create one output field called **item** of type *String*.

7. For the **deletion** dataset create one output field called **Key** of type *Integer*. Make sure you tick **Key**:

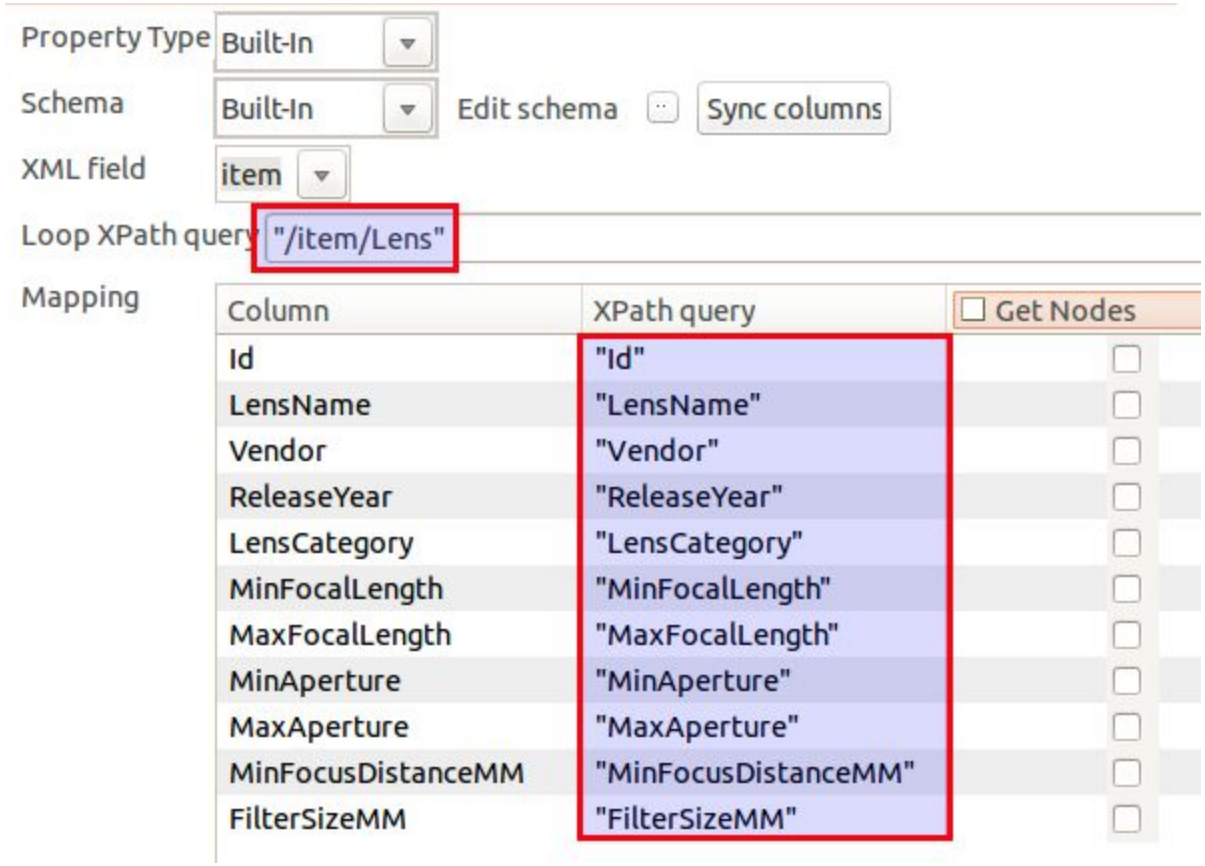
8. For **createOrUpdate** activate the **Expression filter** and define following expression:
```
row2.OperationType.equals("CREATE") ||
row2.OperationType.equals("UPDATE")
```
We only want this output dataset to be populated if there is a CREATE or UPDATE operation taking place.
9. For **deletion** activate the **Expression filter** and define following expression:
```
row2.OperationType.equals("PHYSICAL_DELETE")
```
10. Map the input **Key** field to the output **Key** field for the **deletion** dataset. Alter the expression so that it converts the String value to an Integer:
```
Integer.parseInt(row2.Key)
```
Next map the **item** input field to the **item** output field (dataset **createOrUpdate**)
Now your setup should look like this:



11. Add **tExtractXMLField** component and connect it. It should use the **createOrUpdate** dataset as input.
12. Click on **Edit schema**. With this component we basically want to retrieve the actual data record. So define all the fields as you know them from the entity. Make sure you tick **Key** for the **Id** field:



13. Set the **Loop XPath query** to "/item/Lens". Now we want to extract the fields from the actual data record part of the XML fragment. Set the **XPath query** for the columns to the same name as the columns itself, just enclose them in double quotation marks:



14. From the repository, **Metadata > DB Connections** choose **lens_metadata** and drag and drop it onto the canvas. When prompted, choose the output component (in my case **tPostgreSQLOutput**).
15. For this output component, set **Action** to **Insert and Update** and tick **Die on error**.
16. Finally add a **tLogRow** component and connect it. This will allow us to see in the log the values of the dataset passed on to the DB. So we are finished with the branch for

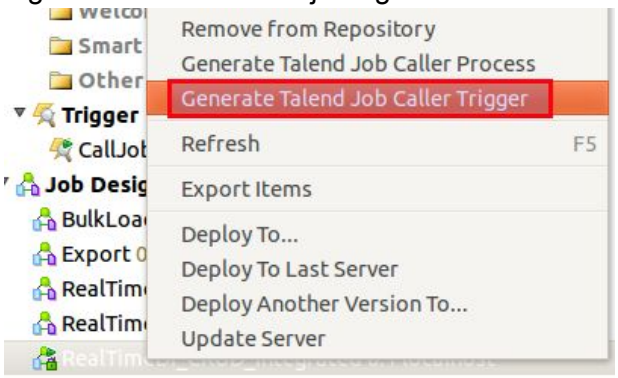
digesting the new and updated records, next we will focus on the deletions.

17. From the repository, **Metadata > DB Connections** choose **lens_metadata** and drag and drop it onto the canvas. When prompted, choose the output component (in my case **tPostgreSQLOutput**). Connect the **tMap** component with **tPostgreSQLOutput** by choosing the **deletion** row/output. Make sure the schema gets propagated.
18. For the DB output component, change the **Action** to **Delete**. Tick **Die on error**.
19. Add a **tLogRow** component and connect it. This will allow us to see the Key in the log once a record gets deleted.
20. Deploy the job to the MDM server.

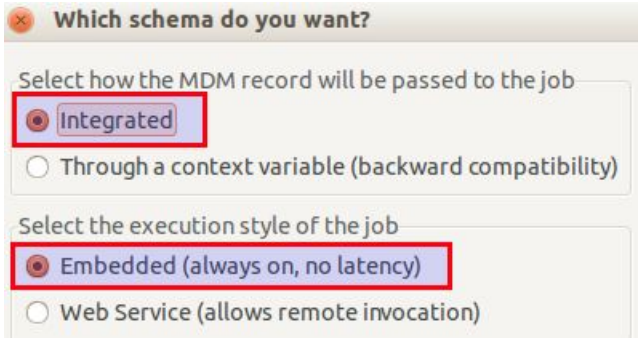
Generating the Job Caller Trigger

We are now finished with the job design:

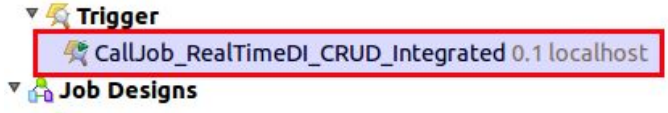
1. Change to the **MDM** perspective.
2. Right click on the job **RealTimeDI_CRUD_Integrated** and choose **Deploy To**. Accept the default settings.
3. Right click on the same job again and choose **Generate Talend Job Caller Trigger**:



4. Make sure that **Integrated** and **Embedded** is selected on the next screen. Click **Generate**:



5. You can now find the process under **Event Management > Trigger**:



Double click on it to open it.

6. Note that the **Entity** value was automatically set to **Update**:

7. Add the following to the **Trigger XPath Expressions**, so that we link the trigger to the right data container, data model and entity:

XPath	Operator	Value	Condition
Update/DataCluster	Matches	Lenses	C4
Update/DataModel	Matches	Lenses	C5
Update/Concept	Matches	Lens	C6

8. Set the conditions to:
   ```
   (C1 Or C2 Or C3)  And C4  And C5  And C6
   ```
9. Undeploy the trigger we uploaded in the previous exercise: Right click on **CallJob_RealTimeDI_CRU** and choose **Undeploy From...**
10. Deploy the trigger to the MDM server.
11. Next create or alter some records via the Talend MDM web interface. Also, watch the log … you should see either the data record or the key displayed.

Checking uniqueness of a functional key before saving

Creating the Before-Saving process

There is quite a nasty problem in our **Lens** data, in the sense that for now the only way to check for uniqueness of the records is based on the lens name. This is rather inconvenient as it is a long string.
There is a solution though to this problem:

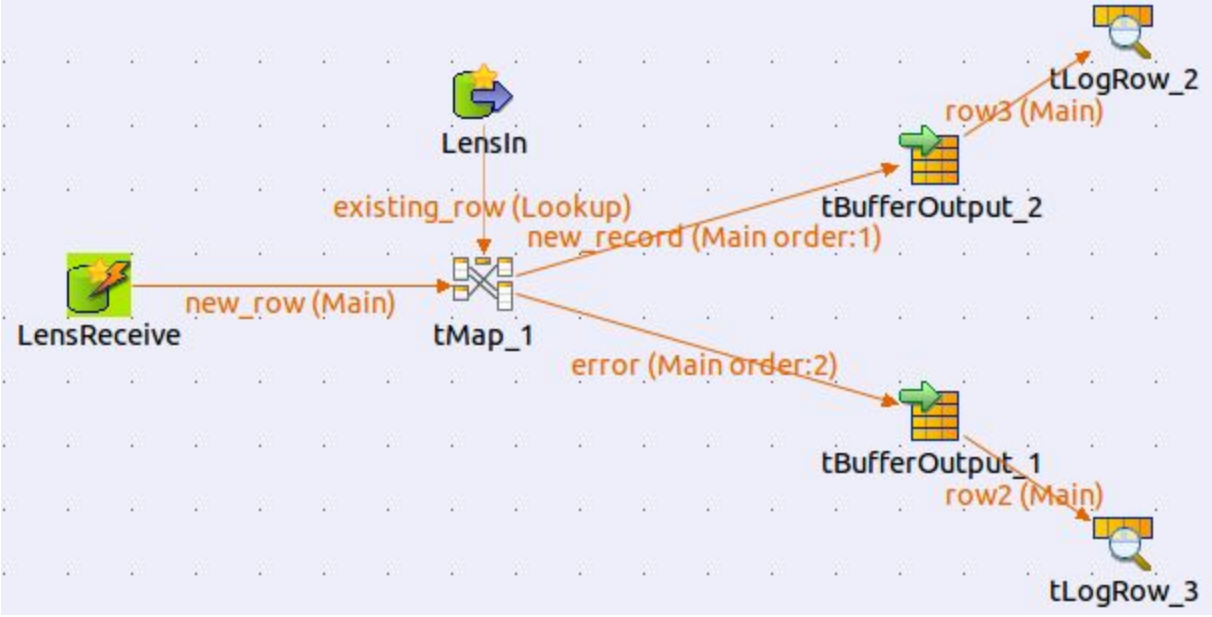
Before-Saving and Before-Deleting Processes
Talend features a *Before-Saving* process as well as *Before-Deleting* process. They are handled differently: These processes do not follow the standard Trigger - Process mechanism; they are directly called by *naming convention*. Furthermore, these processes receive a different XML document than the standard processes: a combination of the Update Report and the record to be added/deleted (Exchange Document). Finally, these processes are also expected to return a variable called *output_report* which contains a *status report* or an *error message* which the web interface can display.

IMPORTANT POINTS

- **Before-Saving** and **Before-Deleting** processes will automatically send the Update Report and the data record (the Exchange Document). To achieve exactly the same with a standard process we would have to use two additional steps/plugins. So you must not use the same approach here. It is already taken care of. Usually the only plugin you will need to use is the **calljob** plugin.
- The **Data Integration job** is supposed to return either a success or error message - but no data! Since TOS MDM version 5 these messages do have to look like this:
  ```
  <report><message type="info"> ... success message ... </message></report>
  <report><message type="error"> ... error message ... </message></report>
  ```
  Prior to TOS MDM version 5 they looked like this:
  ```
  <error code="0">... success message ...</error>
  <error code="1">... error message ...</error>
  ```

Data Integration

Create a new DI job called **FunctionalKeyCheck**. The screenshot below gives you an idea what we are up to:

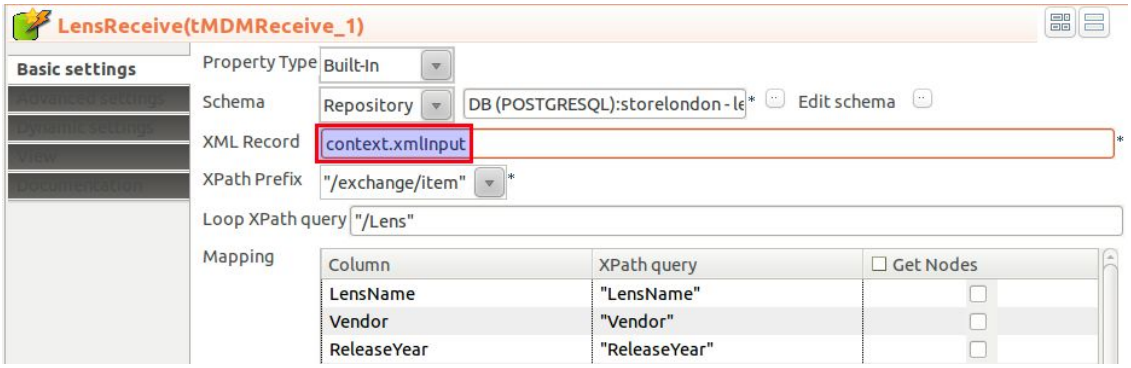


1. Click on the **Context** tab and then on the **+** button to add a new context variable. Assign the name **xmlInput** and the type **String**.
2. Click on the **Values as table** tab within the Context tab and assign following **Default** value:
   ```
   <exchange
   ```

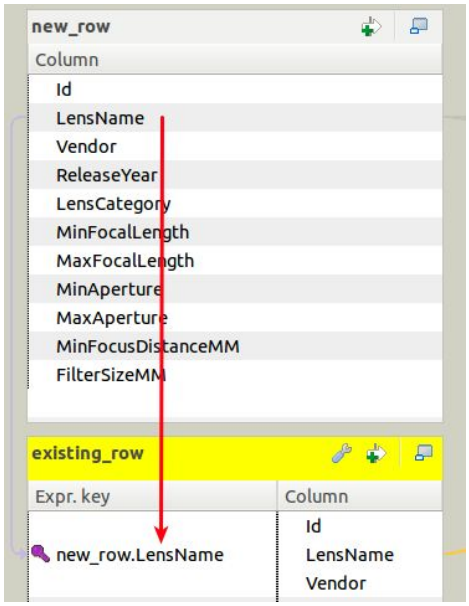
```
xmlns:mdm="java:com.amalto.core.plugin.base.xslt.MdmExtension"><report><Update
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><UserName>administrator</
UserName><Source>genericUI</Source><TimeInMillis>1336311970956</TimeInMillis>
      <OperationType>CREATE</OperationType> <RevisionID>null</RevisionID>
<DataCluster>SecondHandLenses</DataCluster><DataModel>Lenses</DataModel><Concep
t>Lens</Concept><Key>39</Key></Update></report><item><Lens
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><Id>39</Id><LensName>Asah
i Super Takumar 55
1.8</LensName><Vendor>Pentax</Vendor><ReleaseYear/><LensCategory/><MinFocalLeng
th/><MaxFocalLength/><MinAperture/><MaxAperture/><MinFocusDistanceMM/><FilterSi
zeMM/></Lens></item></exchange>
```

This will allow us to test the DI job independently.

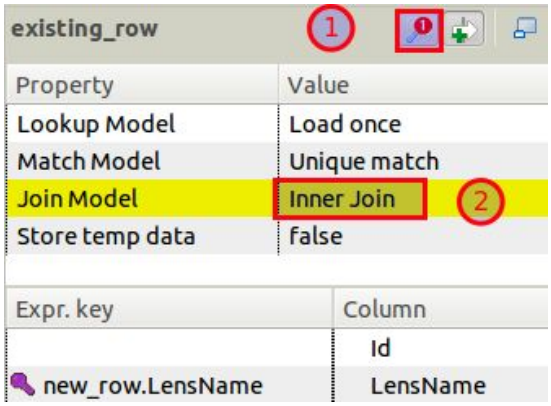
9. From the **Talend DI Metadata** drag and drop the MDM **LensReceive** schema on the design canvas. This will create a **tMDMReceive** component which is designed to receive MDM XML records from TOS MDM triggers and processes.
10. In the **LensReceive Component** settings specify **context.xmlInput** for **XML Record**. All the other settings are already provided by the **Talend DI Metadata** which we set up earlier on:



11. From the **MDM Metadata** add the **LensIn** schema.
12. From the **Palette** add a **tMap** component.
13. Create a **Main row** from the **LensReceive** to the **tMap** component. Name this row **new_row**.
14. Create a **Main row** from **LensIn** to the **tMap** component. Name this row **existing_row**.
15. Double click on the **tMap** component. We will now join the new record with the existing records to find out if the new record is a duplicate. Drag and drop the **LensName** from the **new_row** table on the **LensName** of the **existing_row** table to create a **Join**:

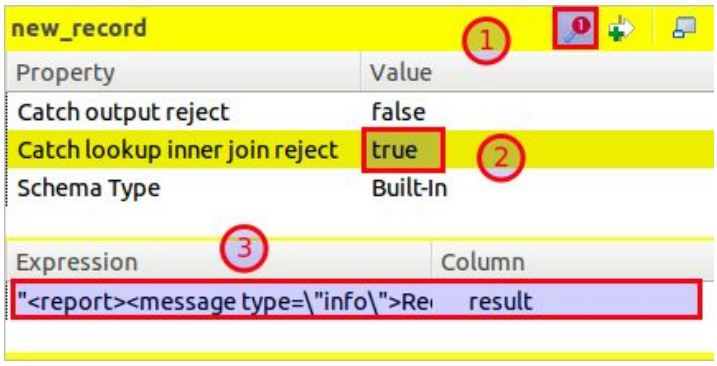


16. Click on the **tMap settings** icon in the existing_row table. Set **Join Model** to **Inner Join**:



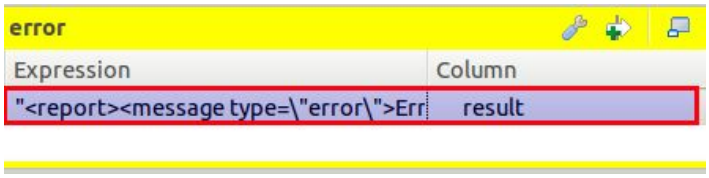
17. Add a new **output table** and name it **new_record**. Click on the **tMap settings** icon and set **Catch lookup inner join reject** to **true**. Add a new **column** named **result** and insert following **Expression**:

```
"<report><message type=\"info\">Record for "+new_row.LensName+"
successfully saved</message></report>"
```

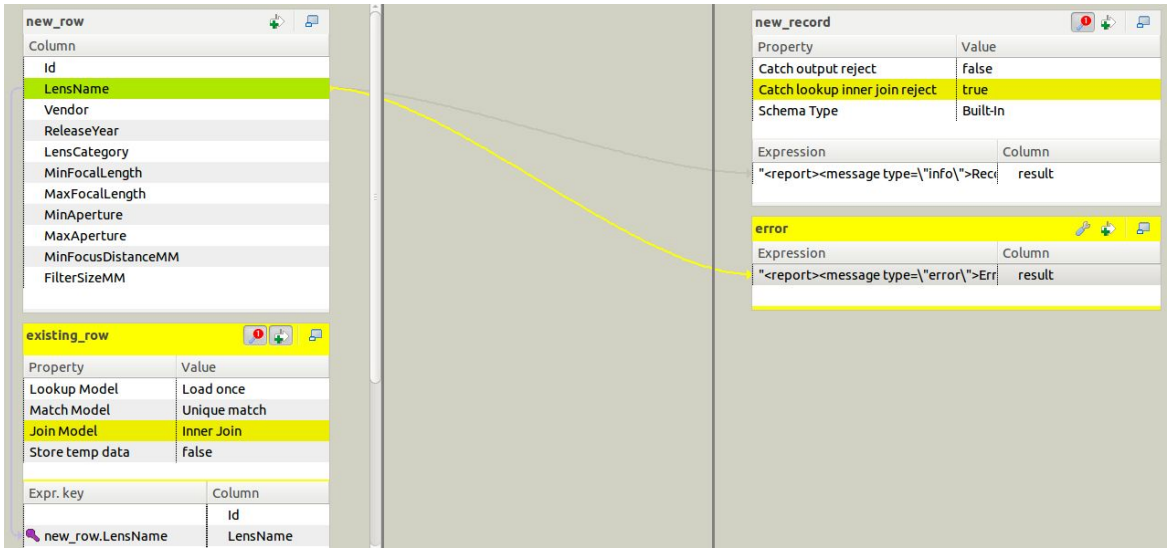


18. Add a new output table and name it **error**. This one will return a row in case there is a match in the join.
    Add a new **column** named **result** and insert following **Expression**:

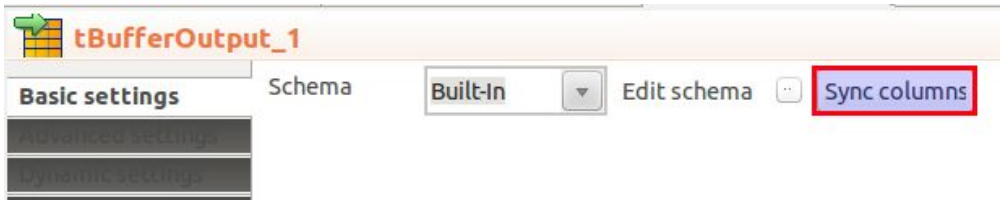
```
"<report><message type=\"error\">Error: Record for "+new_row.LensName+"
is a duplicate</message></report>"
```



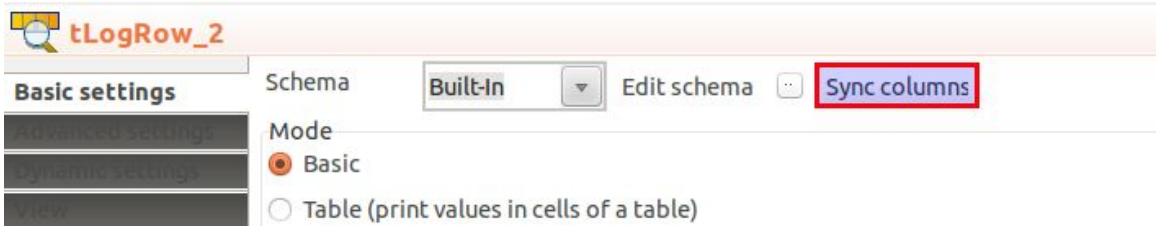
19. Your **tMap** configuration should now look like this:



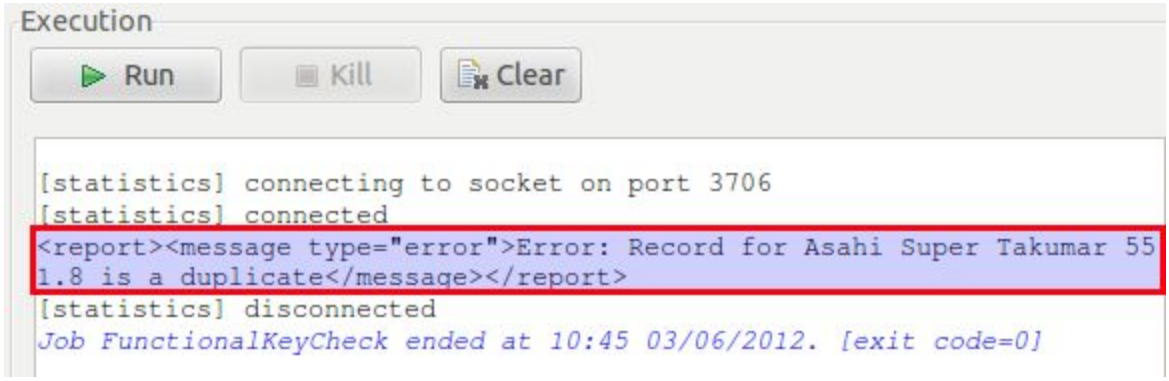
    Click **OK**.
20. From the **Palette** add two **tBufferOuput** components. Create rows from the **tMap** component to them, each one representing one state (success VS error).
21. Do the following for each of the **tBufferOutput** components: Click on **Sync columns**:



22. Although the DI job is functional complete now, it is difficult to test it in isolation. To overcome this shortcoming, we add to **tLogRow** components from the **Palette**. For each of them create a row from the **tBufferOutput** component. Click on each of them and then press **Sync columns**:



23. **Save** the DI job and **run** it. Observe the log to see which message gets returned (Note: Make sure your MDM server is running):



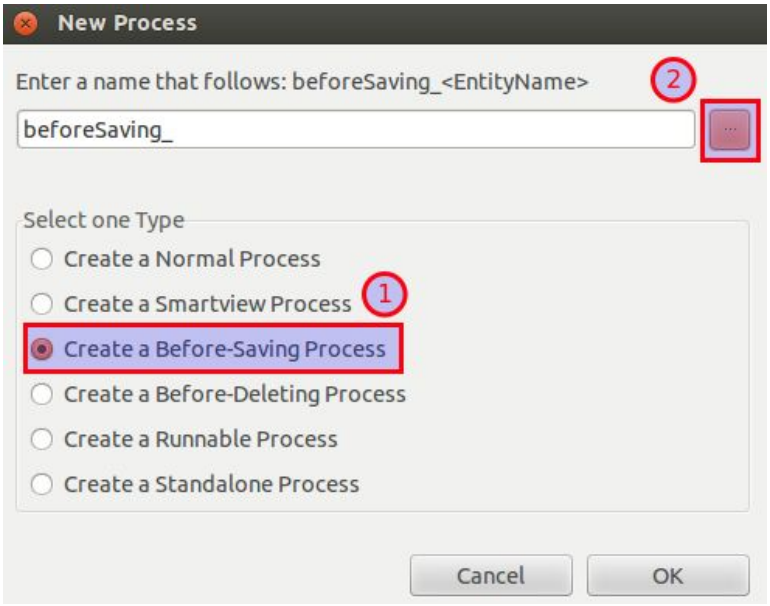
Make sure you test for both scenarios: Error and Info/Success. Simply change the default value XML fragment of the **xmlInput** context variable. If you want to test the error message, make sure the XML fragment holds a lens name which is already available on your MDM server. If you want to test the info/success message, just use a lens name which is not yet stored on your MDM server.

Note: Please note that this setup works fine for small data volumes. If you are handling large data volumes, you do not want to import all the data just to check if the new record is a duplicate or not. To improve this job, you could just first set the lens name as a global variable and then in a subjob fetch records from the MDM hub which have the same lens name.

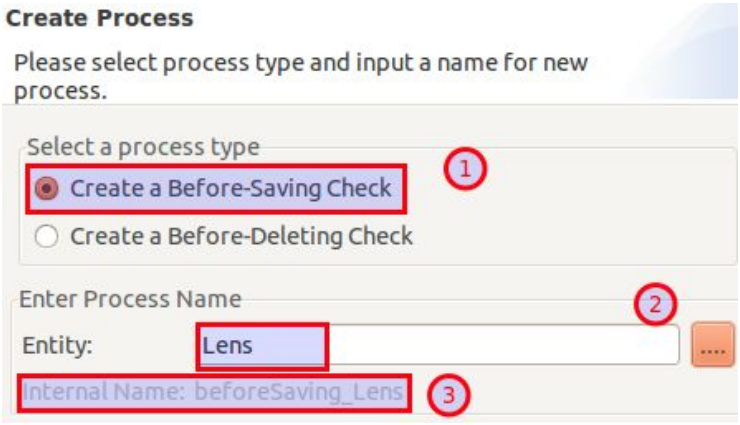
Creating the process

Let's get started:

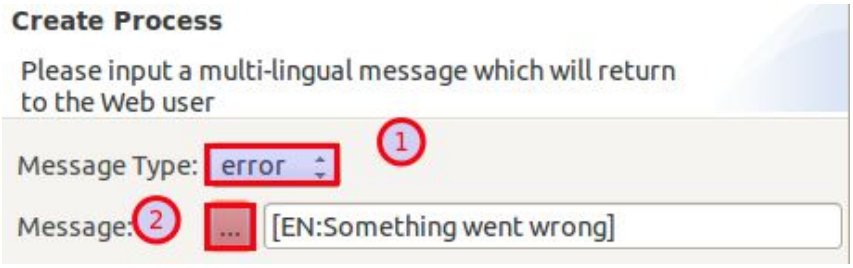
1. Right click on **Process**. Choose **New**.
2. Talend MDM v5.0 only: Choose **Create a Before-Saving Process**.
   You will see that Talend automatically adds the **beforeSaving_** prefix for the process name. Note: Do not change or delete this prefex! The process name has to be prefixed like this as this is the only way for Talend to identify this process as a **Before-Saving Process**. Next just click on the **[…]** ellipsis button and choose the entity - in our case it is **Lens**.



Talend MDM v5.3 and later only: Choose **Create a Before Saving/Deleting Check**. On the next screen choose **Create a Before Saving Check**. Then choose the **entity** to which this process should be linked to: **Lens**. Note that below the system internal process name will be displayed (grayed out):

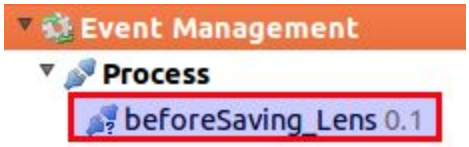


Click **Next**. On the next screen provide a info and error message. Choose the **Message Type** and then click on the ellipse [...] button to specify the message in various languages.

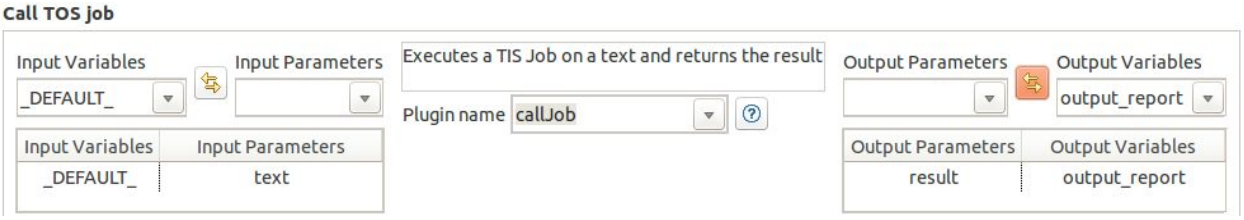


Click **Finish**.

3. Click **OK**. Your process should be now displayed like this in the repository tree:
Talend MDM v5.0 only:



Talend MDM v5.3 and later only:

4. In the **Process Editor** provide following **Description**: *Check uniqueness of LensName*.
5. Talend MDM v5.3 and later only: Delete *Retrieve Message* step … we already implemented a return message in our DI job.
6. Next write the following in the **Step Description field**: *Call TOS job* and click the **+** button.
7. Choose *callJob* as **Plugin name**.
8. Choose *_DEFAULT_* as **Input Variable** and *text* as **Input Parameter** and then click the **Add Link** button  .
9. Choose *result* as **Output Parameter** and set *output_report* as **Output variable** and then click the **Add Link** button  .
10. 

11. Replace the XML in the **Parameters** area with this one:

```
<configuration>
        <url>ltj://FunctionalKeyCheck/0.1</url>
        <contextParam>
                <name>xmlInput</name>
                <value>{_DEFAULT_}</value>
        </contextParam>
</configuration>
```

This basically specifies the job that should be executed (url tag) and the name and the value of the context parameter which is passed on to the job.

12. Deploy the process to the MDM server.

Testing

In the **MDM perspective**, right click on the DI job and choose **Deploy To** … . Choose your MDM Server and click **OK**. Do the same for the **Before Saving** process (If you haven't done so already).

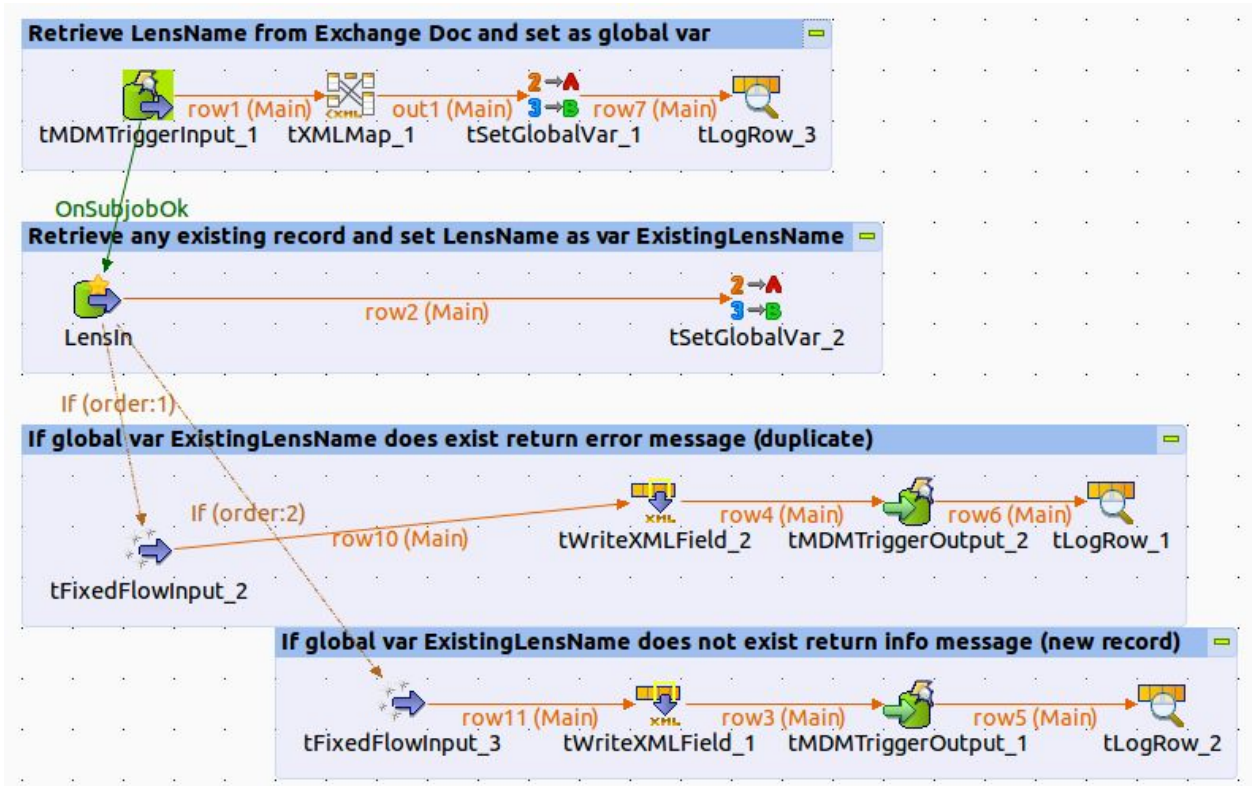
Log on to the web interface and try to insert the same **Lens Name** twice. You should get follow error message similar to this one:

Checking uniqueness of a functional key before saving (Integrated version)

The aim of this exercise is exactly the same as with the previous one, just this time we will use the MDM Trigger service (and add a few other improvements along the way).

Data Integration

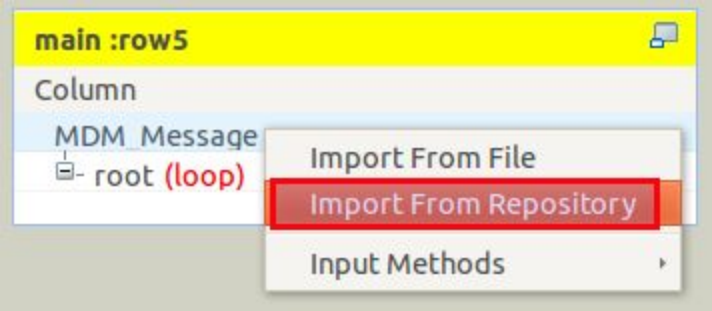
Here is a quick preview of what we are up to:



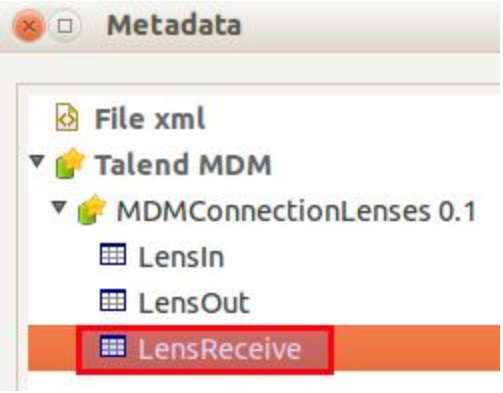
Let's get started:

1. Create a new job called **FunctionalKeyCheck_Integrated**.
2. Add a **tMDMTriggerInput** to the canvas.
3. Add a **tXMLMap** to the canvas and connect it.
4. Double click on the tXMLMap component.

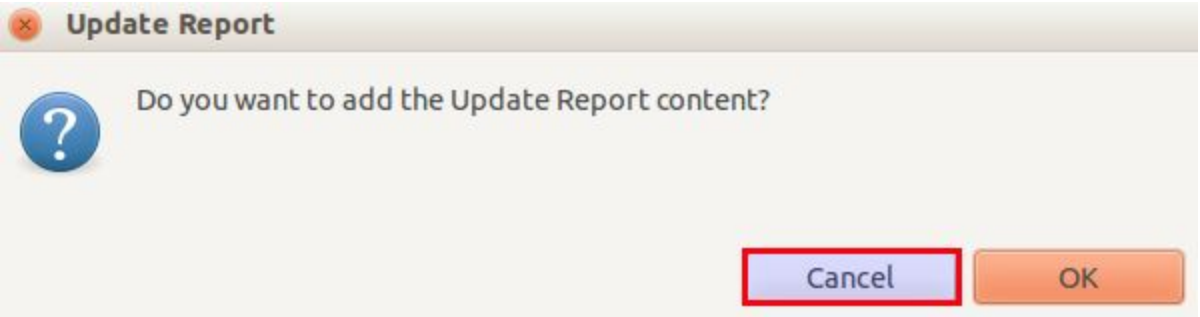
5. On the left hand side, right click on the **MDM Message** node and choose **Import From Repository**



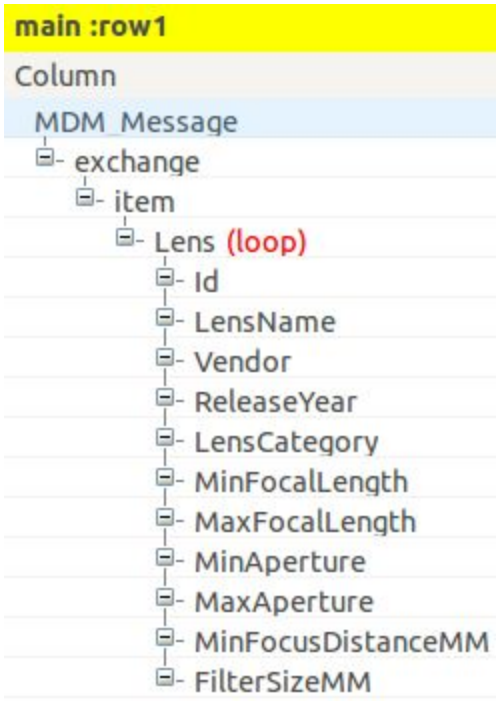
6. Select the respective MDM metadata entity, in our case **LensReceive**:



7. On the next screen click **Cancel** so that the **Update Report** is not added to the tree. We do not require any data from the Update Report for the purpose of this exercise:



8. The tree should look like this now:

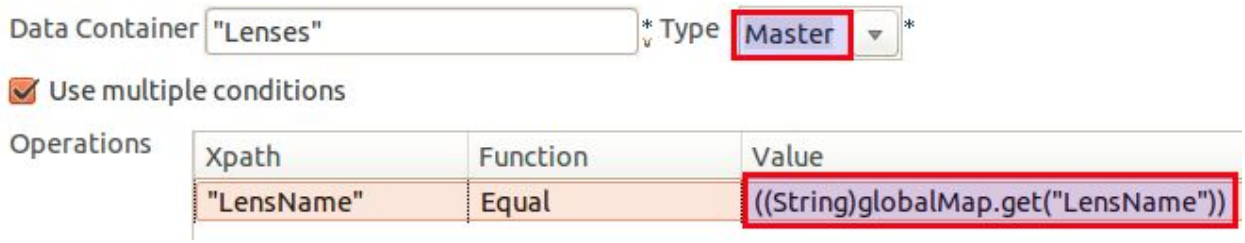


9. Create an output dataset and add a column called **LensName** of type String. Map it to the LensName field from the input dataset:

10. Add a **tSetGlobalVar** component and connect it. Create a new global variable called **LensName** and map it to the LensName from the incoming row (Note: your row might have a different name, so replace out1 with your row name):

11. Drag and drop **LensIn** from the repository onto the canvas just below the current subjob.
12. Right click on **tMDMTriggerInput** and choose **Trigger > On Subjob OK** and point it to **LensIn**.
13. In the **LensIn** component settings specify **Master** as **Type** and add a new condition (as shown in the screenshot below) which will only fetch a record from the MDM hub which has LensName equal to the global LensName variable. To reference the global variable use the following:

```
((String)globalMap.get("LensName"))
```



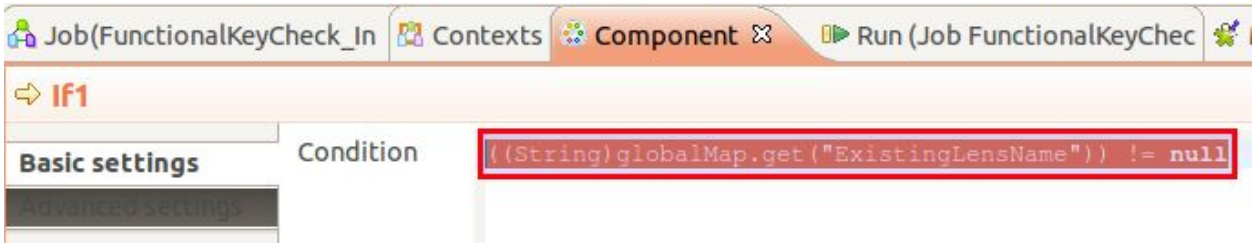
14. Add **tSetGlobalVar** component and connect it. Add a new Key called **ExistingLensName** and specify as value **row2.LensName** (Note: Your incoming row might be named differently, so replace row2 respectively):

The important point here to keep in mind is that this global variable will be only set if a record is retrieved from the MDM hub. We can use this condition then to prepare the return messages.

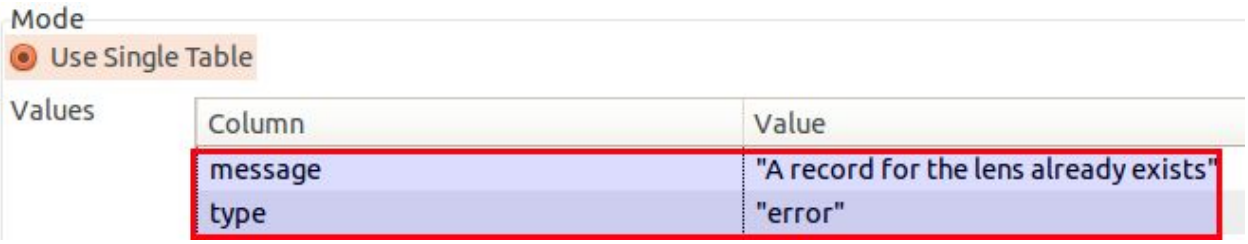
15. Add a **tFixedFlowInput** component. Right click on **LensIn** and choose **Trigger > Run if** and point it to the **tFixedFlowInput** component. Click on the Trigger row name

and specify following condition in the **Component** tab:

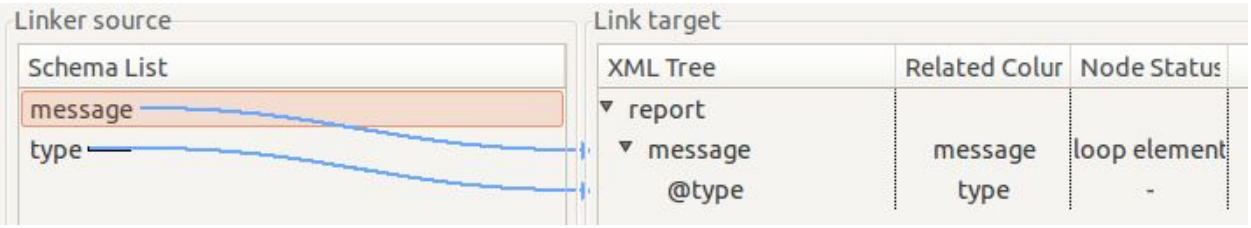
```
((String)globalMap.get("ExistingLensName")) != null
```



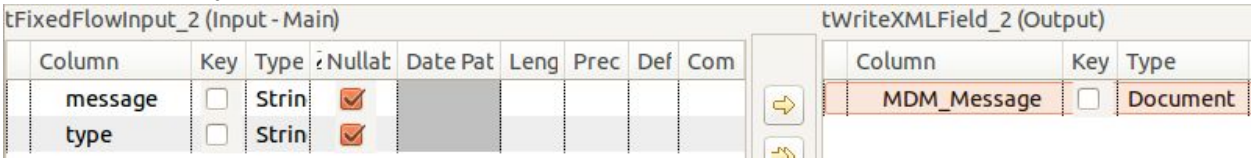
16. Click on the **tFixedFlowInput** component and in the **Component** settings on **Edit Schema**. Add two columns called **message** and **type**, both of type String. In the **Values** table set **message** to "A record for this lens already exists" and **type** to "error":



17. Add a **tWriteXMLField** component and connect it. Click on **Configure XML tree** and set it up as shown in the screenshot below:

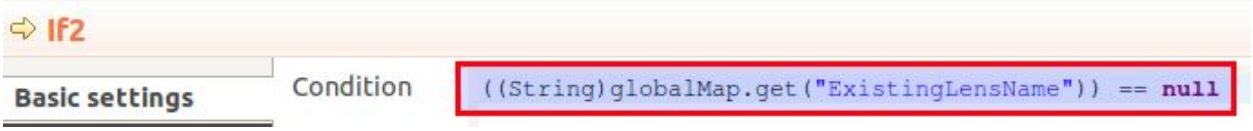


18. Click on **Edit Schema** and set the output column name to **MDM_Message** and make sure that it is of type **Document**:



19. Add a **tMDMTriggerOutput** component and connect it.
20. Finally add a **tLogRow** component so that we can see the output in the console as well.
21. Add another **tFixedFlowInput** component just below the current subjob. Right click on **LensIn** and choose **Trigger > Run if** and point it to the **tFixedFlowInput** component. Click on the Trigger row name and specify following condition in the **Component** tab:

```
((String)globalMap.get("ExistingLensName")) == null
```



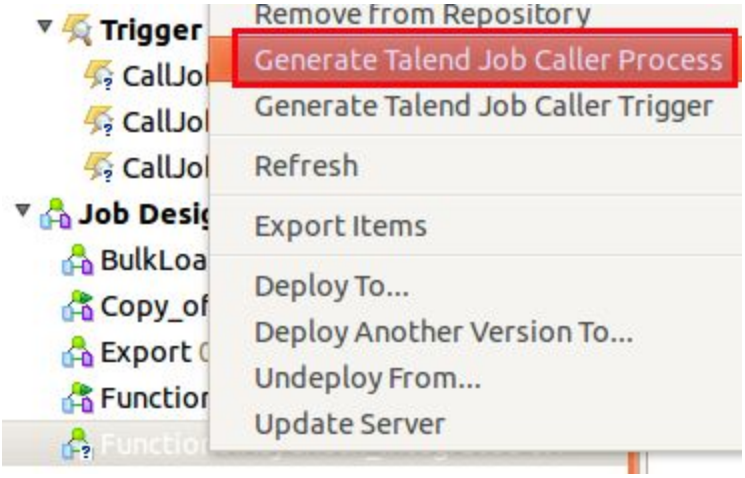
22. Specify the same columns as for the other **tFixedFlowInput** component. This time set the values for **message** to "All good ... this is a new record!" and for **type** to "info":

Column	Value
message	"All good ... this is a new record!"
type	"info"

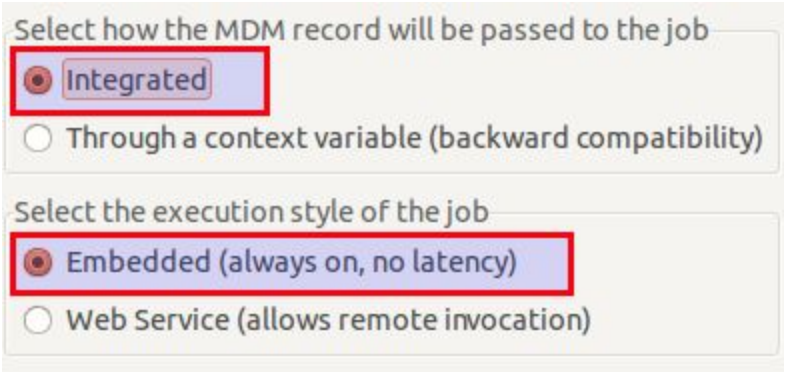
23. Add a **tWriteXMLField**, **tMDMTriggerOutput** and **tLogRow** component and set them up the same way as we did for the error message.
24. Save the job and deploy it.

Before-Saving Process

1. Back in the MDM perspective, right click on the DI job and choose **Create Talend Job Caller Process**:

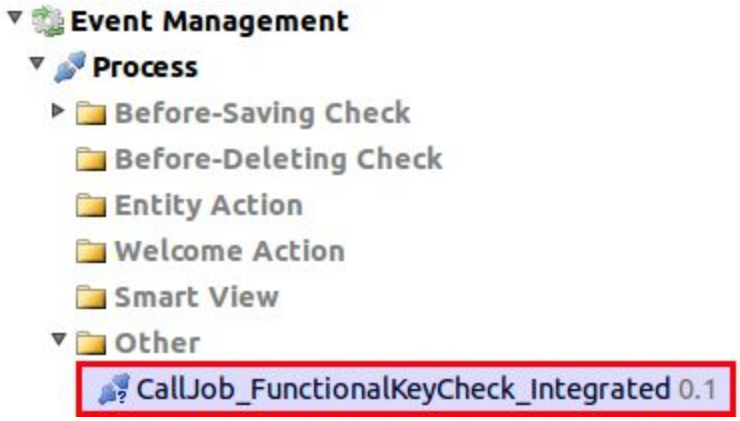


2. Next choose **Embedded** and **Integrated**:

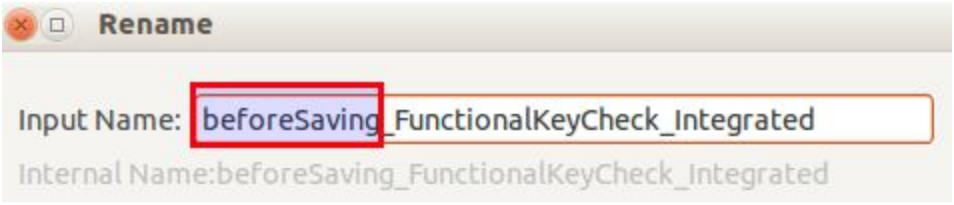


Click **Generate**.

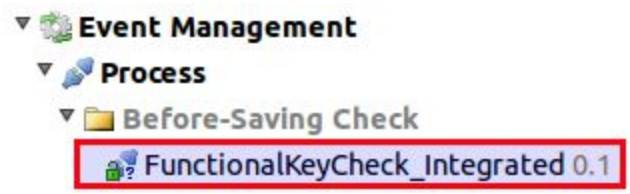
3. Now you can find the auto-generated process in **Event Management > Process > Other**:



4. Right click on the process and choose **Rename**. Replace the **CallJob** prefix with **beforeSaving**

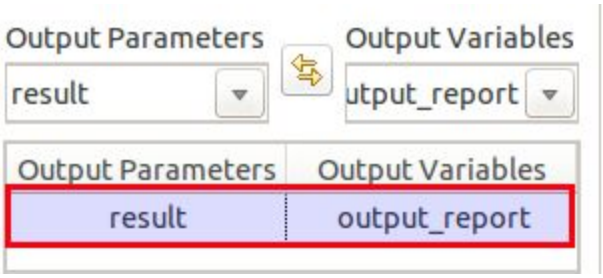


5. You will now find the process in **Event Management > Process > Before-Saving Check**:



6. There is one step left to do: We have to link the process to the entity. The only way to do this is to make the entity name part of the process name: Right click on the process and choose **Rename**. Then click the ellipsis [...] button next to entity and choose the **Lens** entity. Click Ok. (Note: If you followed the previous examples, you have already a process of the same name. In this case, first undeploy this process, then rename it to something else).

7. Double click on the process name. Mark the the **Invoke the job** step. In the **Step** area, set **Output Parameters** to **result** and **Output Variables** to **output_report**, then click the **Link** button:



8. Save the process and deploy it.
9. Add a few records now via the web interface to test to whole setup.

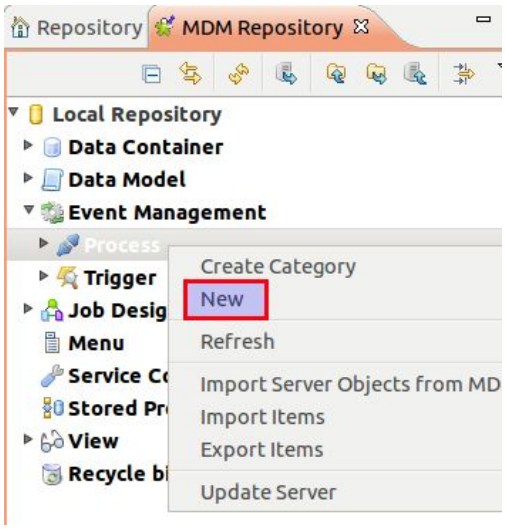
Processes - Advanced Topics

How to create a non-integrated processes manually

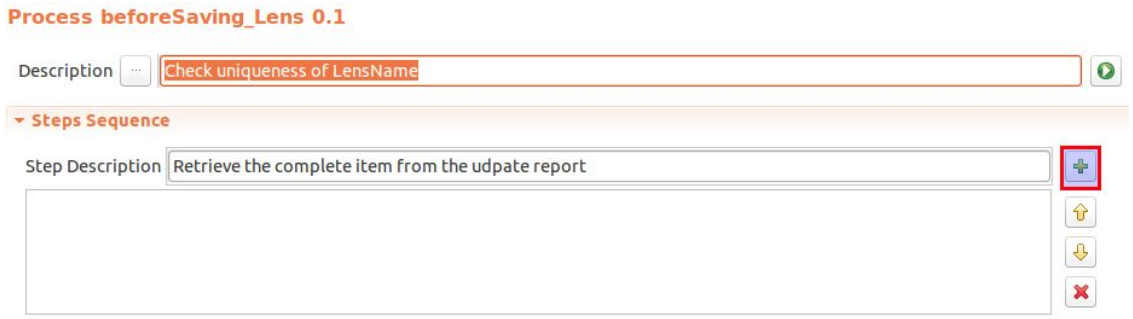
Preparing an XML document for a data integration job
Basically we have to create a special XML document called **Exchange Document** which holds all the information and data which we need for the job. In the following example, the first part of the document will consist of the *Update Report* and for the second part we will retrieve the *actual record data record* using the *mdm:getItemProjection function*. This is the first step of our process workflow. In the next step we have to do some decoding and the third step will pass the **Exchange Document** XML document to the data integration job and execute it.

What follows below is a general walkthrough (so not backed up by any example) of setting up an non-integrated process from scratch:

1. In the **MDM Repository** expend **Event Management** and then right click on **Process**. Choose **New**:



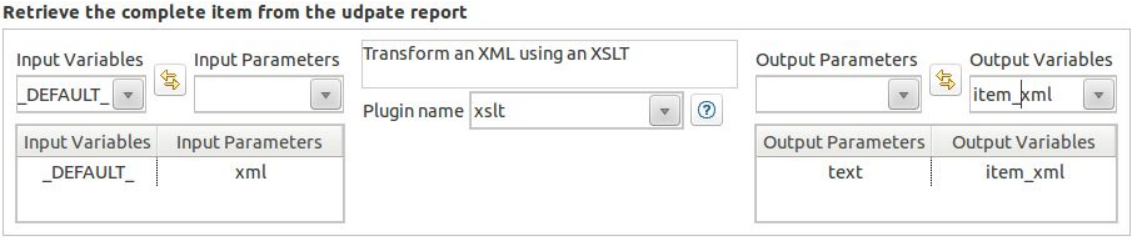
2. In the next dialog choose **Create a normal process** (v5.0) / **Create another process** (v.5.3) and give it a name.
3. Click **OK**. Your process should be now displayed like this in the repository tree.
4. In the **Process Editor** provide a **Description**.
5. If you are working with Talend MDM Studio V5.3 or later you will see that the **Step** section in the main design area is already prepopulated with 3 steps: UpdateReport, Escape the item XML, Redirect. Delete all of them by marking them and clicking on the red X button on the left hand side… we will try to create a process from scratch (which basically replaces the same approach).
6. Now we will start preparing our **Exchange Document XML document**. Write the following in the **Step Description field**: *Retrieve the complete item from the update report* and click the **+** button:



7. In the **Step Specification** section choose *xslt* as **Plugin name**. Click the **?** button next to the plugin name to get a help description of the selected plugin:

8. Next we have to specify how we map the input variable to the step's input parameters: The input for this step will be the **Update Report**.
   In the **Input Variables** combo box write *_DEFAULT_*, which basically means that it will receive the whole Update Report.
9. For **Input Parameters** choose *xml* from the pull down menu.
10. Click on the **Add link** button:

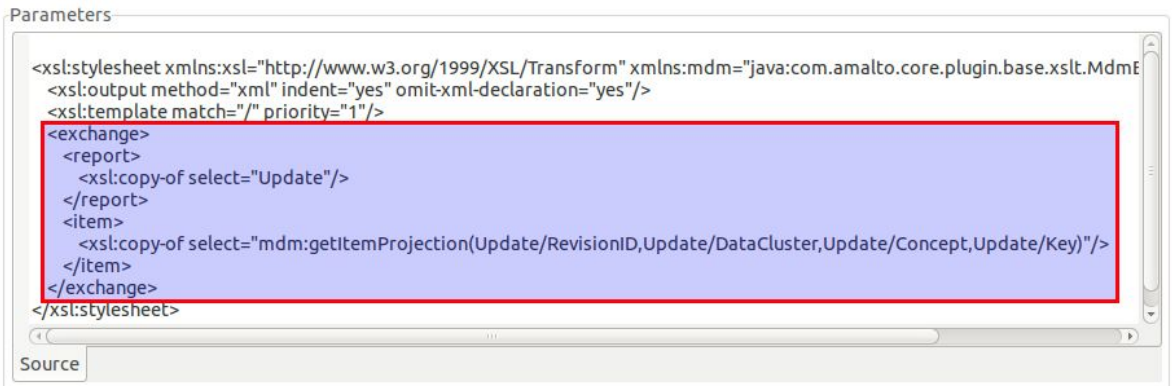
11. Choose *text* from **Output Parameters** to and insert *item_xml* for **Output variables**. Then click the **Add link** button. The setup should look like this now:



12. In the **Parameters** area add following lines after <xsl:template match="/" priority="1"/>:

```
<exchange>
        <report>
            <xsl:copy-of select="Update"/>
        </report>
        <item>
            <xsl:copy-of
select="mdm:getItemProjection(Update/RevisionID,Update/DataCluster,Update
/Concept,Update/Key)"/>
        </item>
</exchange>
```

The XML fragment should look like this now:

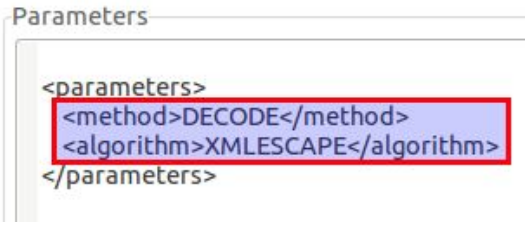


In a nutshell:

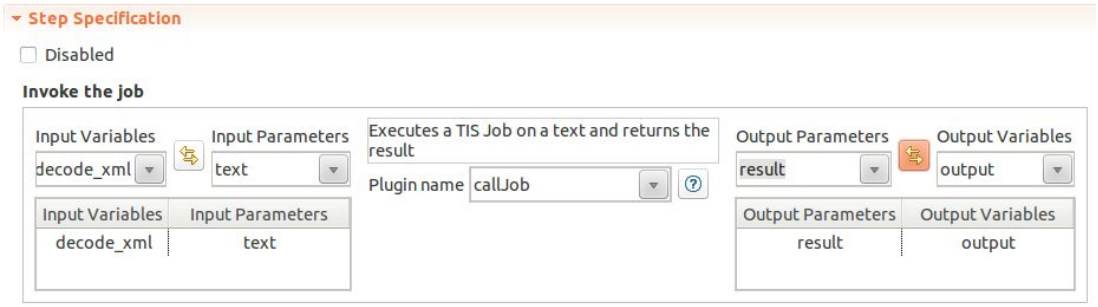
a. Via the input variable *_DEFAULT_* the **Update Report** XML document is sent to the input parameter of the **xslt** plugin.
b. We define the structure of a new XML document, the **Exchange Document**.
c. Within the <exchange><report> we store the **Update Report** by using the xsl:copy-of function.
d. Within <exchange><item> we want to store the related data record. Talend offers a special **XSLT extension** called *mdm:getItemProjection* which enables us to **retrieve the whole data record** (in encoded XML format). Note that the *mdm:getItemProjection* uses details from the Update Report like the Key/ID to retrieve the record from the MDM data container. In this process every XML sign is escaped to avoid character encoding conflicts. The result is then sent from the output parameter to the *item_xml* variable.

13. The next step is to decode the XML: In the **Step Description** write: *Decode item xml* and press the **+** button.
14. Choose **codec** as **plugin name**.
15. Map the **input variable** *item_xml* (which is the output variable of the previous xslt step) to *law_text* **input parameter**.
16. Map the **output parameter** *codec_text* to the **output_variable** *decode_xml*. The setup should now look like this:

17. In the **Parameter** section replace the question mark between the **method** tags with *DECODE* and the question mark between the **algorithm** tags with *XMLESCAPE*:



18. Now the decoded XML document is ready to be sent to the last step: Insert a new **Step Description** *Invoke the job* and press the **+** button to add it to the process flow.
19. Choose *callJob* as **Plugin name**.
20. Choose *decode_xml* as **Input Variable** and *text* as **Input Parameter**.
21. Choose *result* as **Output Parameter** and set *output* as **Output variable**.



22. Adjust the XML in the **Parameters** area so that the context Parameter name (here: xmlInput) matches the one in your DI job:

```
<configuration>
        <url>ltj://sampleJob/0.1</url>
        <contextParam>
                <name>xmlInput</name>
                <value>{decode_xml}</value>
        </contextParam>
</configuration>
```

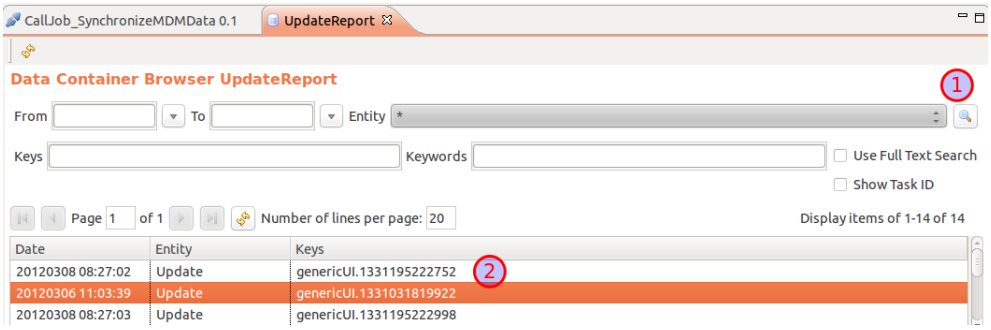
This basically specifies the job that should be executed (see url tag) and the name and the value of the context parameter which is passed on to the job. **Very important**: Note how **decode_xml** from the process is matched to the expected context variable in our data integration job called **xmlInput**. Remember that we defined **xmlInput** as context variable earlier on when we created our data integration jobs? This is how everything ties together!

How to test processes

There is a neat way to test processes:

1. In the MDM repository navigate to **Data Container > System > UpdateReport**. Double click on it:

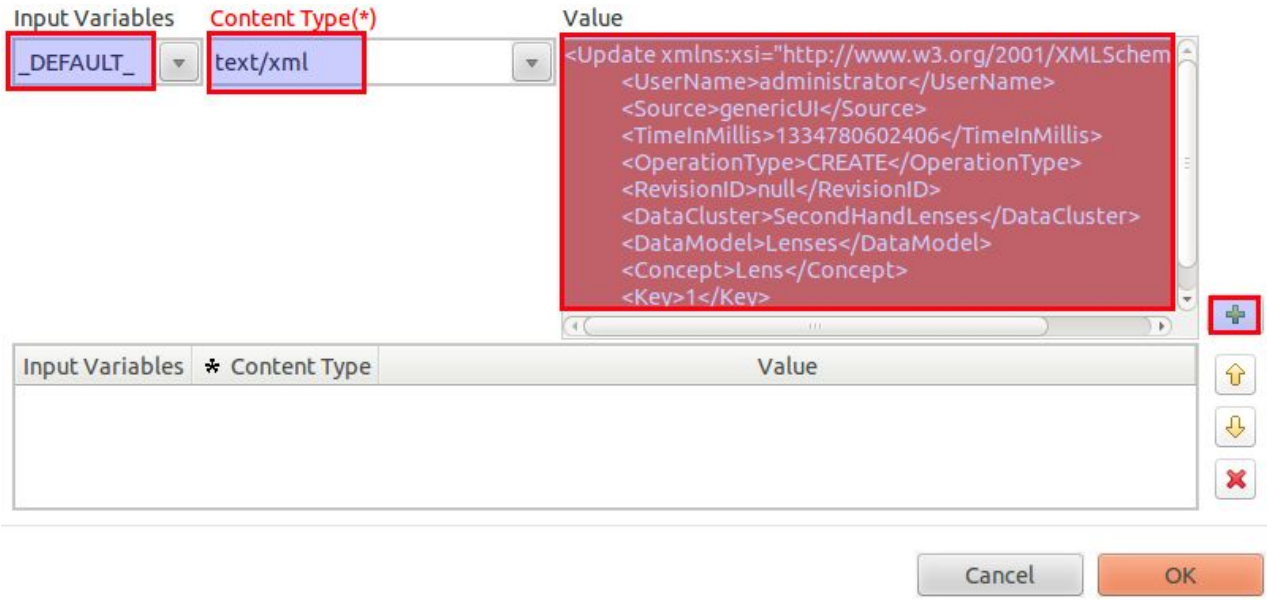
2. Specify the search criteria (if any), run a search and **double click** on the record for which you want to retrieve the XML fragment



3. In the XML viewer click the **Source** tab and **copy** the XML fragment:

Now open the process and click the **Execute** button:

Let's provide the value of the input variable of our very first step in the process - **Stylesheet**: Set **_DEFAULT_** as **Input Variable** and then choose **text/xml** as **Content Type**, paste the XML fragment of the **Update Report** in the **Value** text box and click the **+** button:

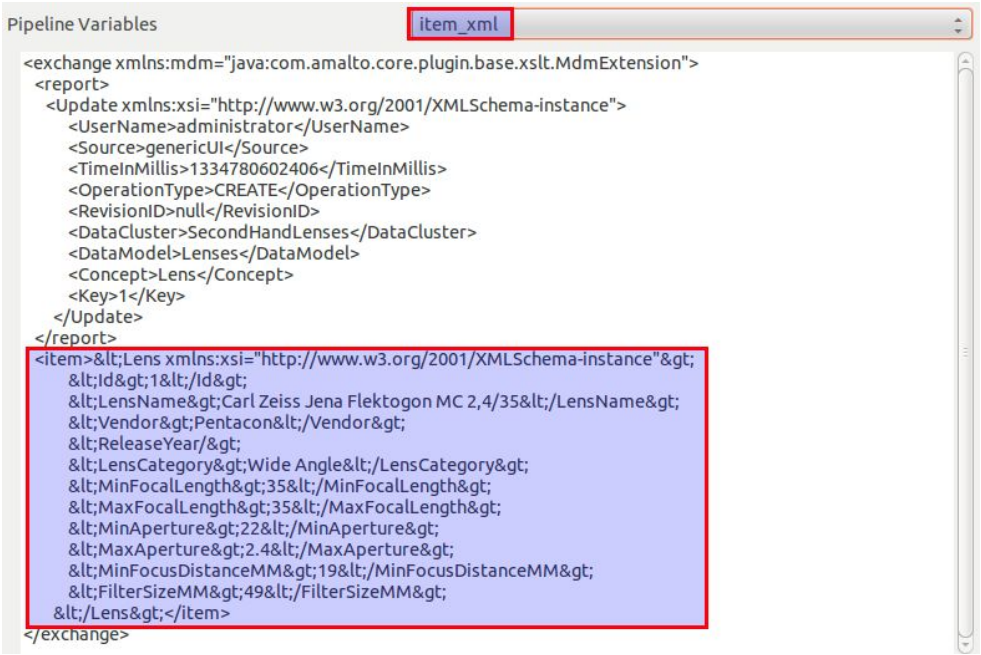


Click **OK**.

Please note that more recent versions of Talend MDM Studio make testing easier as you can retrieve the Update Report directly from the Execute dialog.

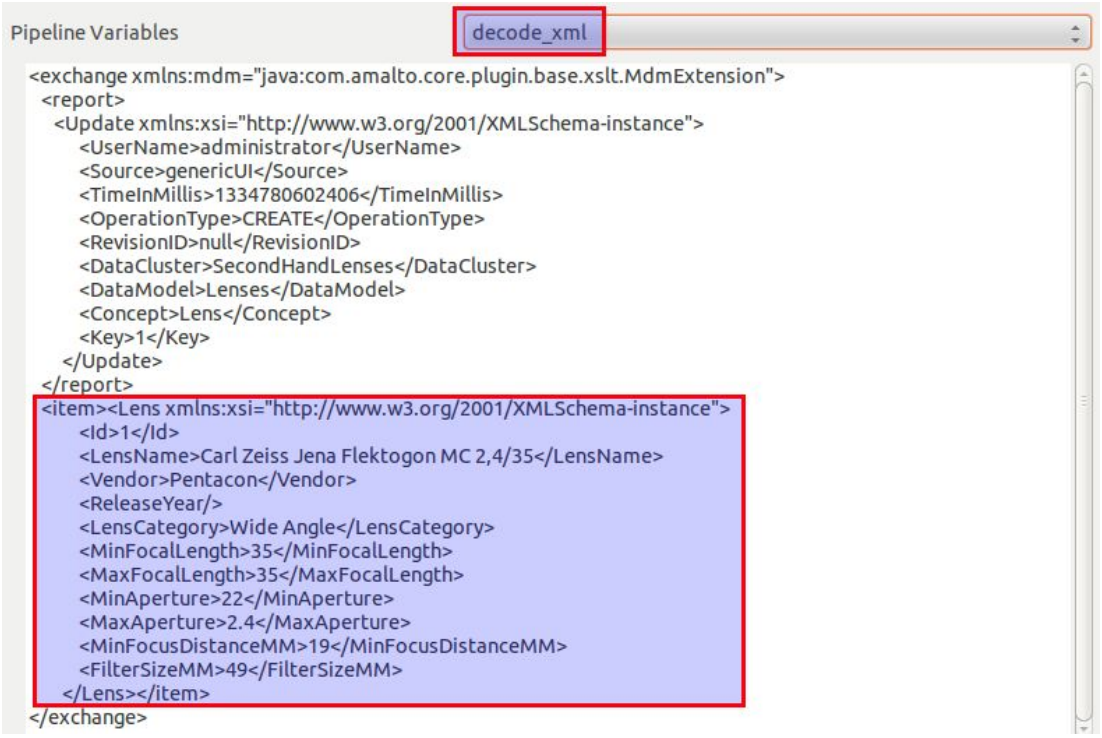
Then choose the MDM server you want to run your test on. In the next window you can have a look at the variable values of all steps. This is quite useful as you will now actually very easily understand what each step of the process is doing:

1. Let's analyse the output of the first step (**Stylesheet**), so choose **item_xml**:



   Notice that record information within the <item> element is encoded.

2. Now choose the output variable of the second step (**Escape the item XML**): **decode_xml**



As we would expect, the record information within the <item> element is now properly decoded. **The result you see here is the XML document which will be passed on to the data integration job.**

3. Finally, if you choose **output** you will see the value of the output variable of the 3rd step (**Invoke the job**):

Understanding the XML document which is passed on from our process to the data integration job

Before continuing it is essential to have a good understanding of the XML document (the **Exchange Document**) which is passed on to the data integration job in our process. Let's quickly go through the process again:

1. We basically create a special XML document: The first part consisting of the **Update Report** and for the second part we retrieve the actual data record data using the

mdm:getItemProjection function.
2. We decode the XML document
3. We call a job which receives the XML document

Below you can find an **Exchange Document** example for a **new record**. Note that the <exchange> element includes two child elements: <report> and <item>:

- <report> includes the **Update Report** (metadata about the submitted record). Pay attention to the <OperationType> element which tells you if the event was an update, deletion or new record. Also of importance are the <DataCluster>, <DataModel> and <Concept> elements. To map these names to their Talend MDM synonyms: data cluster is the **data container** and the concept is the **business entity**.
- <item> on the other hand includes all the record values that were submitted.

```
<exchange xmlns:mdm="java:com.amalto.core.plugin.base.xslt.MdmExtension">
	<report>
		<Update xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
			<UserName>administrator</UserName>
			<Source>genericUI</Source>
			<TimeInMillis>1336311970956</TimeInMillis>
			<OperationType>CREATE</OperationType>
			<RevisionID>null</RevisionID>
			<DataCluster>SecondHandLenses</DataCluster>
			<DataModel>Lenses</DataModel>
			<Concept>Lens</Concept>
			<Key>39</Key>
		</Update>
	</report>
	<item>
		<Lens
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><Id>39</Id><LensNam
	e>Asahi Super Takumar 55
	1.8</LensName><Vendor>Pentax</Vendor><ReleaseYear/><LensCategory/><MinFoc
	alLength/><MaxFocalLength/><MinAperture/><MaxAperture/><MinFocusDistanceM
	M/><FilterSizeMM/>
		</Lens>
	</item>
</exchange>
```

Below you can find an **Exchange Document** example for an **update**. Note the structure is a bit different: In this case the <item> element does not exist, but a similar one called <Item> [Note: starts with capital i], which is a child element of the <Update> element. <Item> holds the old and new values of the changed business attribute. In the example below only one business attribute value was changed:

```
<exchange xmlns:mdm="java:com.amalto.core.plugin.base.xslt.MdmExtension">
```

```
      <report>
            <Update xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                  <UserName>administrator</UserName>
                  <Source>genericUI</Source>
                  <TimeInMillis>1336311787613</TimeInMillis>
                  <OperationType>UPDATE</OperationType>
                  <RevisionID>null</RevisionID>
                  <DataCluster>SecondHandLenses</DataCluster>
                  <DataModel>Lenses</DataModel>
                  <Concept>Lens</Concept>
                  <Key>10</Key>
                  <Item>
                        <path>Vendor</path>
                        <oldValue>Carl Zeiss Jena</oldValue>
                    <newValue>Pentacon</newValue>
                  </Item>
            </Update>
      </report>
</exchange>
```

The next example is for a deletion:

```
<exchange xmlns:mdm="java:com.amalto.core.plugin.base.xslt.MdmExtension">
      <report>
            <Update xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                  <UserName>administrator</UserName>
                  <Source>genericUI</Source>
                  <TimeInMillis>1336302748679</TimeInMillis>
                  <OperationType>PHYSICAL_DELETE</OperationType>
                  <RevisionID>null</RevisionID>
                  <DataCluster>SecondHandLenses</DataCluster>
                  <DataModel>Lenses</DataModel>
                  <Concept>Lens</Concept>
                  <Key>29</Key>
            </Update>
      </report>
      <item/>
</exchange>
```

Testing process via the web interface

This gives you the chance to test everything (process, trigger, job):
Once the process, trigger and DI job are deployed, just log in the user console and add a new record to the business entity **Lens** so that the trigger gets called. Observe the console output! Your condition should validate to true.
If you get an error, it looks something like this:

```
11:42:16,068 INFO  [RoutingEngineV2CtrlSession] (C1 Or C2) And C4 And C5 : true
11:42:16,090 INFO  [RoutingEngineV2CtrlSession] Echo deactived ,skip it!
11:42:16,099 INFO  [RoutingEngineV2CtrlSession] C0 And C1 And C2 And C3 : false
11:42:16,104 INFO  [RoutingEngineV2CtrlSession] C1 And C2 And C3 : false
11:42:16,108 INFO  [RoutingEngineV2CtrlSession] C0 And C1 And C2 : false
11:42:18,112 INFO  [STDOUT] Batch entry 0 INSERT INTO "lenses"."lens_metadata" (
"lens_name","vendor","release_year","lens_category","min_focal_length","max_foca
_length","min_aperture","max_aperture","min_focus_distance_mm","FilterSizeMM")
VALUES (Tamron 24mm 2.5,Tamron,NULL,,NULL,NULL,NULL,NULL,NULL,NULL) was aborted.
 Call getNextException to see the cause.
```

If everything runs successfully, the terminal output will just be like this:

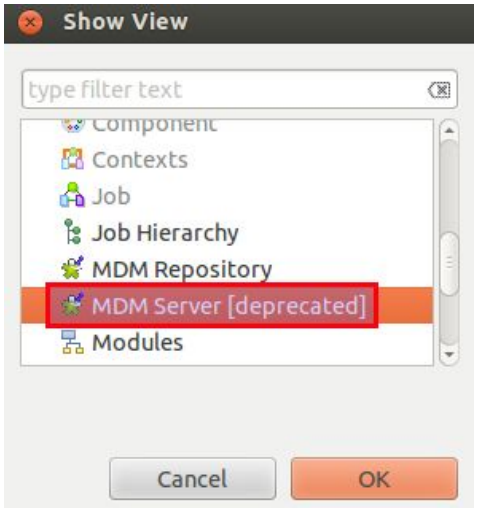
```
11:59:11,118 INFO  [RoutingEngineV2CtrlSession] (C1 Or C2) And C4 And C5 : true
11:59:11,133 INFO  [RoutingEngineV2CtrlSession] Echo deactived ,skip it!
11:59:11,138 INFO  [RoutingEngineV2CtrlSession] C0 And C1 And C2 And C3 : false
11:59:11,144 INFO  [RoutingEngineV2CtrlSession] C1 And C2 And C3 : false
11:59:11,149 INFO  [RoutingEngineV2CtrlSession] C0 And C1 And C2 : false
```

Managing the deployed DI jobs, views, processes and triggers on the MDM server

We have to keep on eye on what is already deployed on the server so that we don't end up having various jobs running with conflicting interests. In our example, we might have already the **CRU** process and DI job deployed, but we want to replace this now with the **CRUD** ones. I will not cover this in very much detail here as it is quite straightforward.

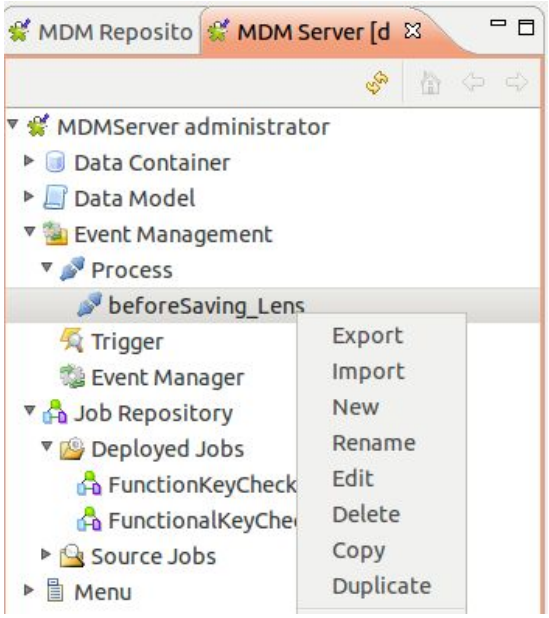
TOS MDM v5.0 [Note: More recent versions do not have this view, see alternative methods below] has a depreciated server view which can still be used to manage all the deployed files on the server.

To add this view, simply click on **Window > Show View …** and choose **Talend > MDM Server**:



Click **OK**. You have now a new **MDM Server** tab available. To manage the files, simply right

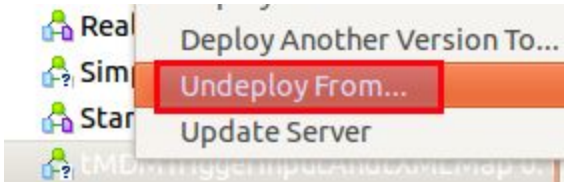
click on them and choose the required action:



Talend MDM Studio v5.3 does not have this server view any more: Please find below some alternative methods:

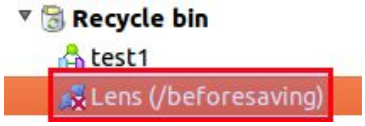
Deleting triggers, processes, job files, views etc

Since version 5.4 there is a convenient **Undeploy** option available in the context menu:



For earlier version, you have to use this workaround:

Once you delete files in the **Talend Open Studio for MDM**, the file ends up in the recycle bin:

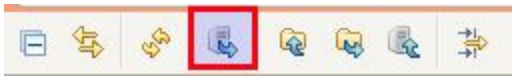


Then you can remove the file from the **Recycle bin**. If you forget to synchronize the studio with the server, you will get a warning message:

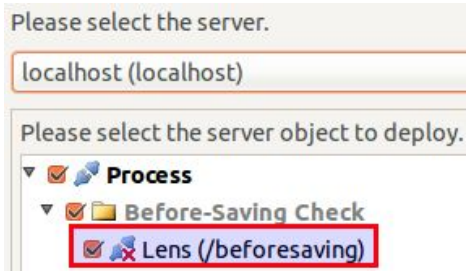
At this stage make sure you press **Cancel**!

So the recommended method is:

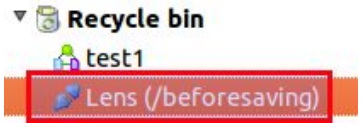
1. Delete the object (process, trigger, job etc)
2. Click the **Update Server** button:



3. The next pop-up window will allow you to select the object which should be synced:



Click **OK**. This should synchronize the changes from the Studio to the MDM server. Note that in the recycle bin the red cross next to the object name now gone:



4. You can now safely delete the object or restore it, if your intention was to only delete it from the MDM server but to keep it in the Studio.

Deactivating a trigger

Sometimes, in example when you are testing something, it might suffice to just deactivate the trigger, which can be specified directly in the trigger configuration:

Save the trigger and redeploy it then.

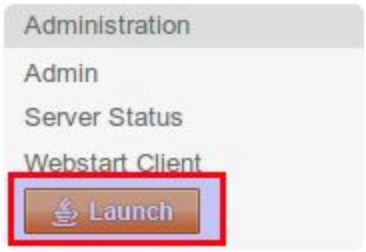
Caution when renaming files in the MDM Studio

When you rename files in the MDM Studio this will currently not be reflected on the MDM Server. If you click **Update Server** a file with the new name will be deployed, but the file with the old name is still there.

eXist

Accessing the eXist DB client

eXist DB comes with a Java query client. You can access it via http://localhost:8080/exist. Scroll down the page and on the left hand side you will find an **Administration** section. Click the **Launch** button



This usually requires the Oracle JRE. So if you have not installed it do so by running the following on the Terminal (these commands are Ubuntu specific):

```
sudo apt-get update
sudo apt-get install sun-java6-jdk
sudo update-java-alternatives -s java-6-sun
```

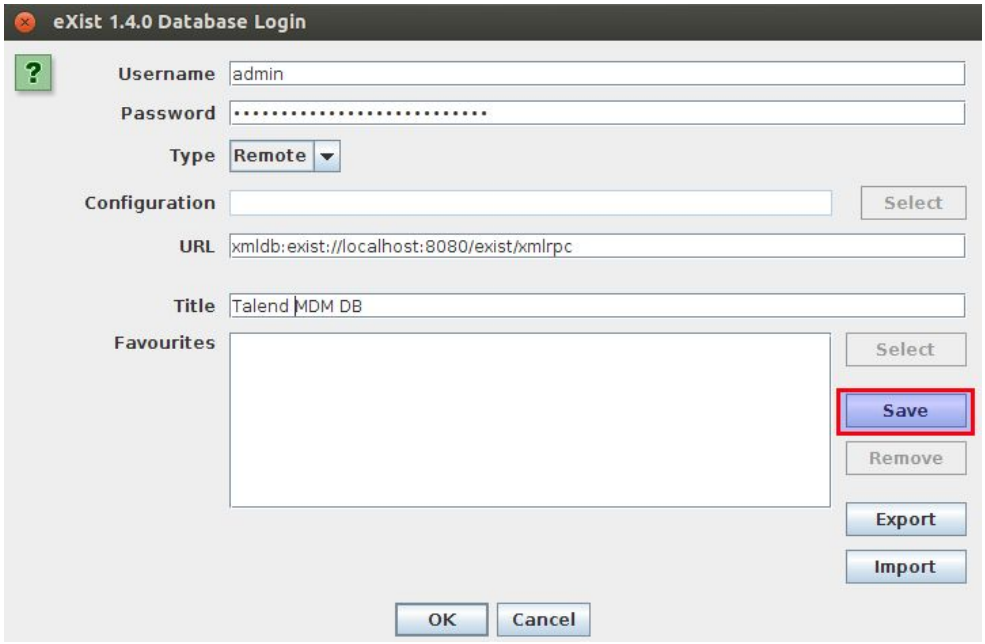
If the Java Web Start is not opening automatically, then just download the file. Right click on it Open > Sun Java 6 Web Start. If this doesn't work, then just open it from the command line with this command:

```
$JAVA_HOME/bin/javaws exist.jnlp
```

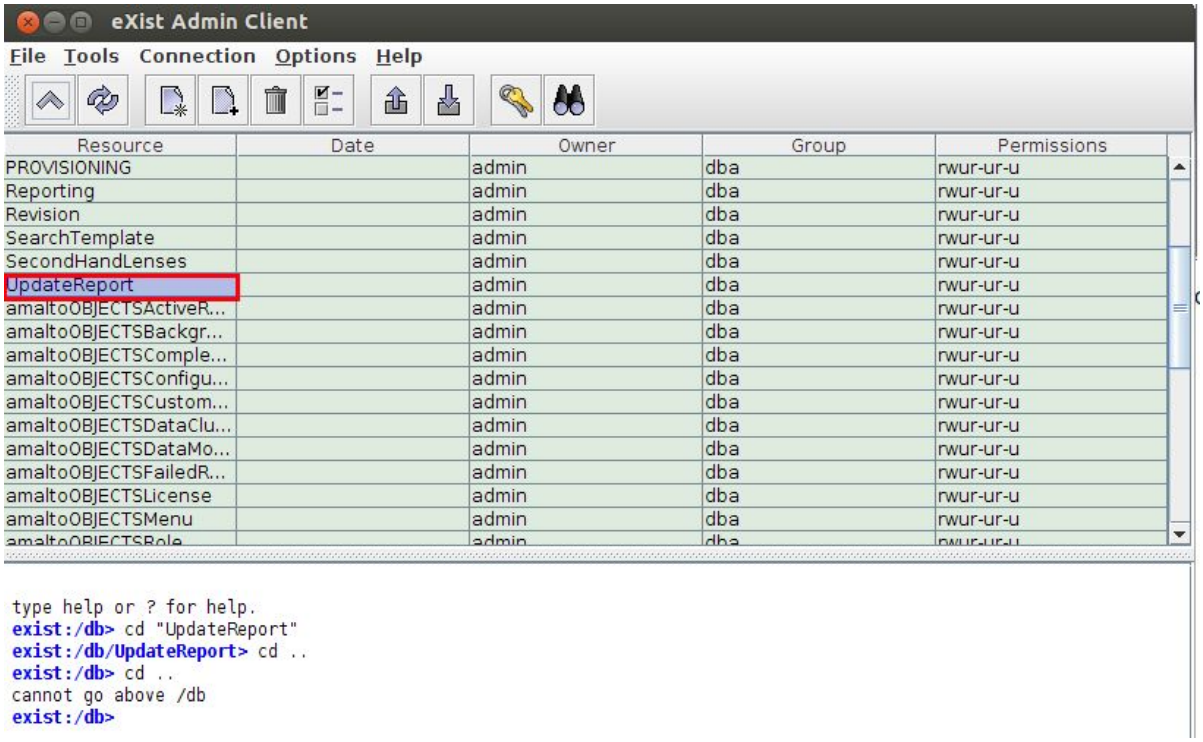
The login details are as follows:
user name: admin
password: 1bc29b36f623ba82aaf6724fd3b16718

The password can be changed in jboss-<version>/bin/mdm.conf

You can create a favourite so that you don't have to specify all the details next time again. Provide a title and then click the **Save** button:

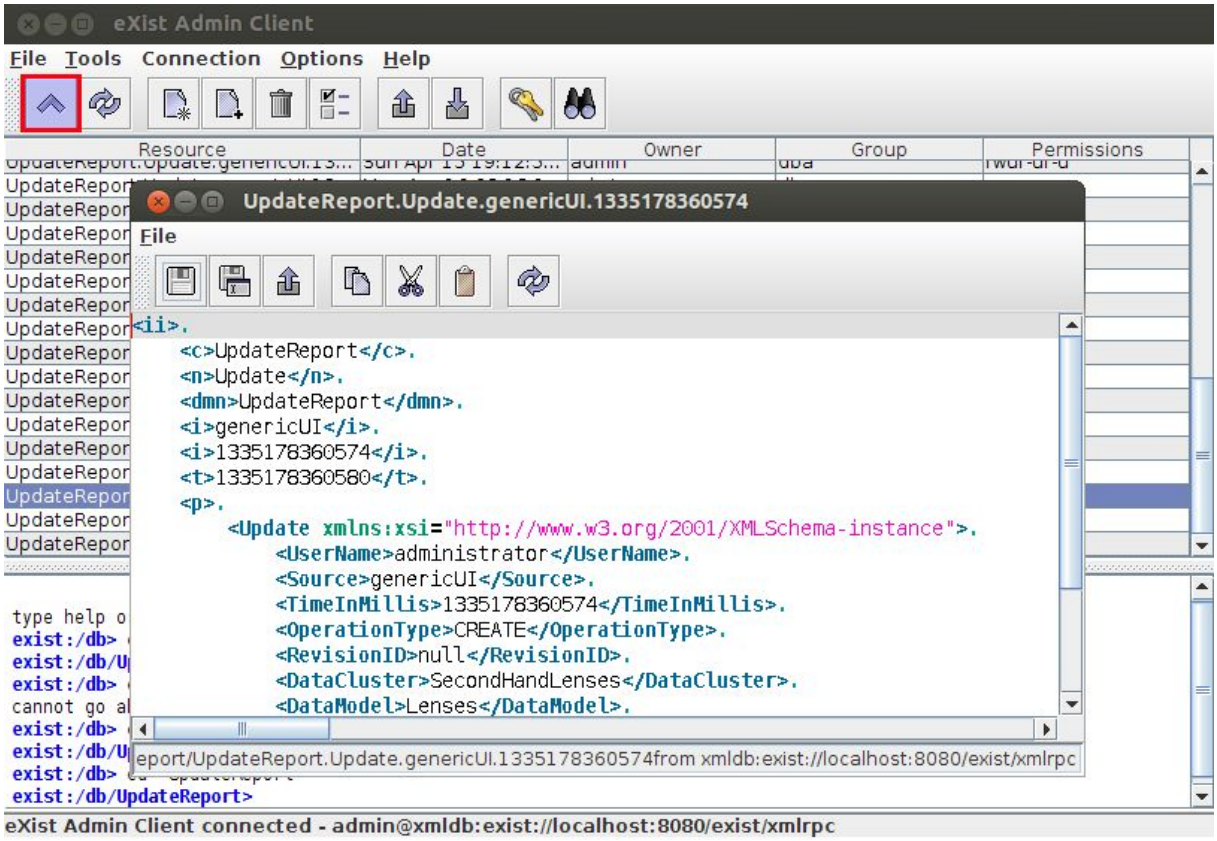


Then click **OK**. The query client will now come up:



Names listed in the **Resource** column represent the Talend **Data Containers**. You will find our **SecondHandLenses** as the **UpdateReport** here.

Double click on the **UpdateReport** record and then you get access to all the data records. If you want to see the content of the data record, simply double click on it:



Close the data record details window and press the **Up** button (highlighted in the screenshot above) to jump up one level.
This is an excellent way to learn more about the XML structure.

eXist Administration: Browsing data records

You can access the eXist admin pages using following URL: http://localhost:8080/exist/admin.
The login details are exactly the same as mentioned above for the query client:
user name: admin
password: 1bc29b36f623ba82aaf6724fd3b16718

In the **eXist** world **Data Containers** are called **Collections**, hence choose **Browse Collections** from the left hand side navigation:

Select a Page

- Home
- System Status
- Browse Collections
- User Management
- View Running Jobs
- Examples Setup
- Install Tools
- Backups
- Query Profiling
- Grammar cache
- Shutdown
- Logout

Then look for our **SecondHandLenses** collection and click on the lens:

☐	SearchTemplate	rwur-ur-u	admin	dba	Mar 4 2012 09:39:19
☐	SecondHandLenses	rwur-ur-u	admin	dba	Apr 18 2012 08:40:57
☐	UpdateReport	rwur-ur-u	admin	dba	Mar 4 2012 09:39:20

Finally you get an overview of all the records created for the business entity:

Browsing Collection: /db/SecondHandLenses

	Name	Permissions	Owner	Group	Created	Modified	Size (KB)	Revision
	Up							
☐	SecondHandLenses.Lens.1	rwur-ur-u	admin	dba	Apr 18 2012 21:23:22	Apr 18 2012 21:23:22	4	
☐	SecondHandLenses.Lens.10	rwur-ur-u	admin	dba	Apr 20 2012 21:23:21	Apr 20 2012 21:23:21	4	
☐	SecondHandLenses.Lens.11	rwur-ur-u	admin	dba	Apr 23 2012 11:42:15	Apr 23 2012 11:42:15	4	

Clicking on one of them will show you the record data:

```
-<ii>
   <c>SecondHandLenses</c>
   <n>Lens</n>
   <dmn>Lenses</dmn>
   <i>8</i>
   <t>1334866397999</t>
  -<p>
   -<Lens>
      <Id>8</Id>
      <LensName>Carl Zeiss Jena Biotar 75mm f/1.5</LensName>
      <Vendor>Carl Zeiss Jena</Vendor>
      <ReleaseYear>1951</ReleaseYear>
      <LensCategory>Short Telephoto</LensCategory>
      <MinFocalLength>75</MinFocalLength>
      <MaxFocalLength>75</MaxFocalLength>
      <MinAperture>1.5</MinAperture>
      <MaxAperture>1.5</MaxAperture>
      <MinFocusDistanceMM>80</MinFocusDistanceMM>
      <FilterSizeMM>58</FilterSizeMM>
    </Lens>
   </p>
 </ii>
```

The **Update Report** just happens to be an ordinary collections as well, so you can access its records exactly the same way. This is a fairly good way to learn about the XML structure.

Backup, adding indexes and more

A few other topics you might want to have a look at are backing up eXist, creating indexes (see here and here) and if you want to even find out more about eXist, visit their online documentation.

Where is the data stored

You can find the data in:
`<jboss_dir>/server/default/deploy/exist-1.4.0-rev11706-TalendPatch.war/WEB-INF/data/`

H2 embedded relational database

How to access the H2 database from a query client

Obtaining the JDBC Driver

If you require the JDBC driver, download the whole installation package from the h2 website and extract it. Within the bin folder you find a h2-<version>.jar file which you can use as the JDBC driver.

JDBC connection string

To easiest way to find out about the connection details, is to look at the datasources.xml file on the MDM server:
`<jboss-root>/server/default/conf/datasources.xml`

You will see something similar to this:

```
    <!-- H2 DATASOURCE -->
    <datasource name="H2-Default">
        <master>
            <type>RDBMS</type>
            <rdbms-configuration>
                <dialect>H2</dialect>

<connection-driver-class>org.h2.Driver</connection-driver-class>

<connection-url>jdbc:h2:///opt/talend/MDM/TOS_MDM-V5.3.1/TOS_MDM-Server-r104014
-V5.3.1/jboss-4.2.2.GA/server/default/data/H2-Default/${container}</connection-
```

```
url>
                <connection-username>sa</connection-username>
                <connection-password>sa</connection-password>

<fulltext-index-directory>/opt/talend/MDM/TOS_MDM-V5.3.1/TOS_MDM-Server-r104014
-V5.3.1/jboss-4.2.2.GA/server/default/data/indexes/H2-Default</fulltext-index-d
irectory>
```

How to setup the connection in your query client

Use following connection details to connect to the embedded H2 database:

JDBC URL example:

```
jdbc:h2:///home/dsteiner/development/software/talend/TOS_MDM-All-r111943-V5.4.1
/jboss-4.2.2.GA/server/default/data/H2-Default/Lenses;AUTO_SERVER=TRUE;MVCC=TRU
E;LOCK_TIMEOUT=15000
```

Where Lenses is the Data Container. You can find the exact details in:

```
<jboss-root>/server/default/conf/datasources.xml
```

In the same file you can find the username and password to connect as well:

```
▼<datasources xmlns="http://www.talend.com/mdm/datasources">
  <!--  H2 DATASOURCE  -->
  ▼<datasource name="H2-Default">
    ▼<master>
      <type>RDBMS</type>
      ▼<rdbms-configuration>
        <dialect>H2</dialect>
        <connection-driver-class>org.h2.Driver</connection-driver-class>
        ▼<connection-url>
          jdbc:h2:///home/dsteiner/development/software/talend/TOS_MDM-All-r111943-V5.4.1/jboss-
          4.2.2.GA/server/default/data/H2-Default/${container}
        </connection-url>
        <connection-username>sa</connection-username>
        <connection-password/>
        ▼<fulltext-index-directory>
```

In SQLWorkbench/J I set up this connection like this:

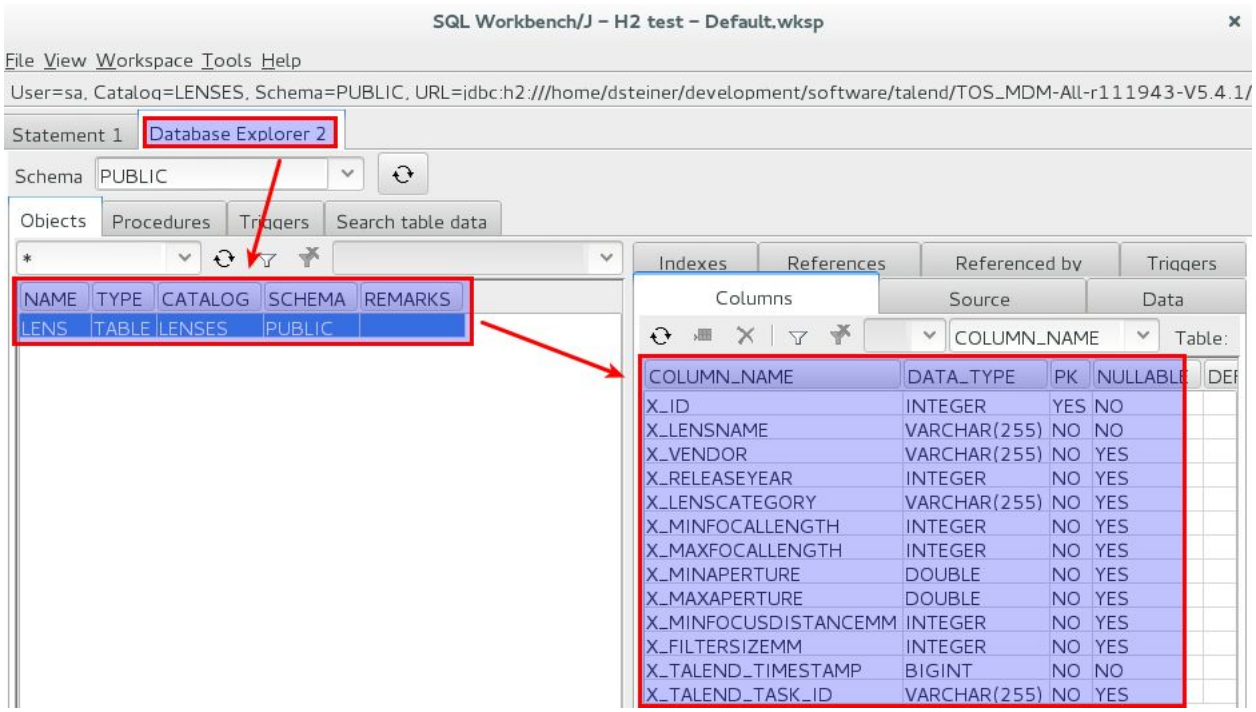
In this case, for the extended properties I just defined AUTO_SERVER.
Also note that in this case the login details didn't have a password specified.

Note: A port number is not required!

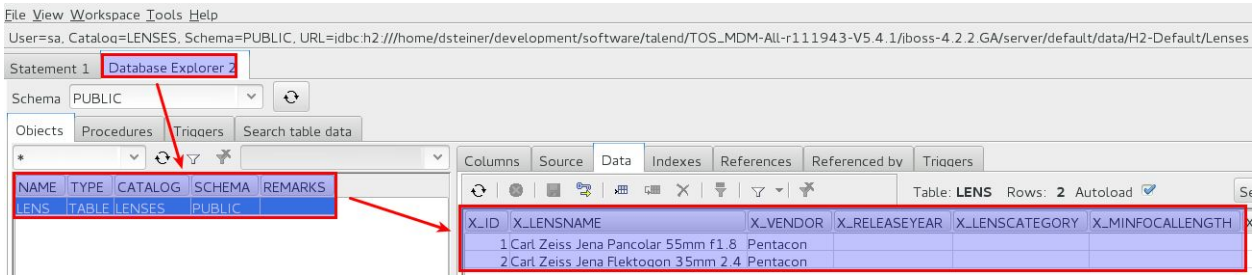
IMPORTANT: To access the H2 database from the SQL client you have to stop the MDM server, otherwise you will get following error message:

```
Database may be already in use: "Locked by another process". Possible
solutions: close all other connection(s); use the server mode [90020-174]
```

So connect via the query client once you stopped the MDM server and you can explore the tables in example in SQLWorkbench's **Database Explorer**:



And here looking at the data:



I goes without saying that you should be very careful here on what you are doing! Only use this for SELECT statements. If do something else, make sure that you know 100% what you are up to!

How to change the data type of an element once the data model is deployed

If your Talend job throws an error like this one:

```
 faultString: org.hibernate.exception.DataException: Could not execute JDBC
batch update; nested exception is:
      javax.ejb.TransactionRolledbackLocalException:
org.hibernate.exception.DataException: Could not execute JDBC batch update
```

… it might be that the field mapping is not correct. Inspect the server log (jboss/server/default/log/server.log) for more details. You might find an error message like this one:

```
Caused by: org.h2.jdbc.JdbcBatchUpdateException: Data conversion error
converting "'Joe' (EMPLOYEE: X_FIRSTNAME BOOLEAN)"; SQL statement:
```

This column is definitely wrongly defined in the data model. Follow these steps to solve this problem (as also highlighted here):

1. In your data model change the element type
2. Export all the data of the relational table
3. Drop the relational table
4. Redeploy the data model
5. Now your relational table should have the correct field types.
6. Import the data

Or let's take another example: You just created a data model and for one element you defined the wrong data type. You realized this shortly after deploying it. There is no data yet stored for this model. So you can just simply change the db column type by running a statement similar to this one (do this only if you are aware of the db data types that Talend MDM uses!) :

```
ALTER TABLE EMPLOYEE ALTER COLUMN X_FIRSTNAME VARCHAR(255);
```

Open your data model and correct the data type for this specific element. Redeploy the model. All should be good now.

Advanced MDM concepts

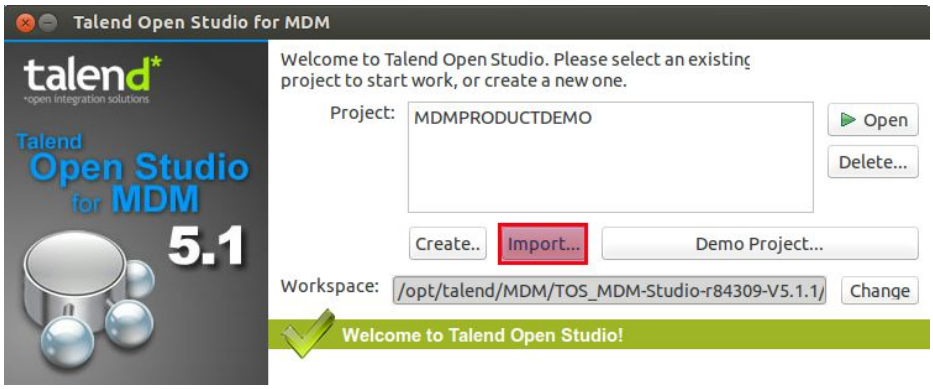
The intention of this guide was to only provide you with a quick starting point. I strongly suggest that you read the dedicated **Talend MDM** guide (which you can find on the Talend website) for advanced concepts like foreign keys, foreign key filters, data model inheritance and polymorphism, how to set lookup fields and much more. Also, in regards to **Talend DI** it is definitely worth having a look at the Talend DI components guide (which you can find on the Talend website as well) to understand the full potential of each **MDM component**.

Appendix

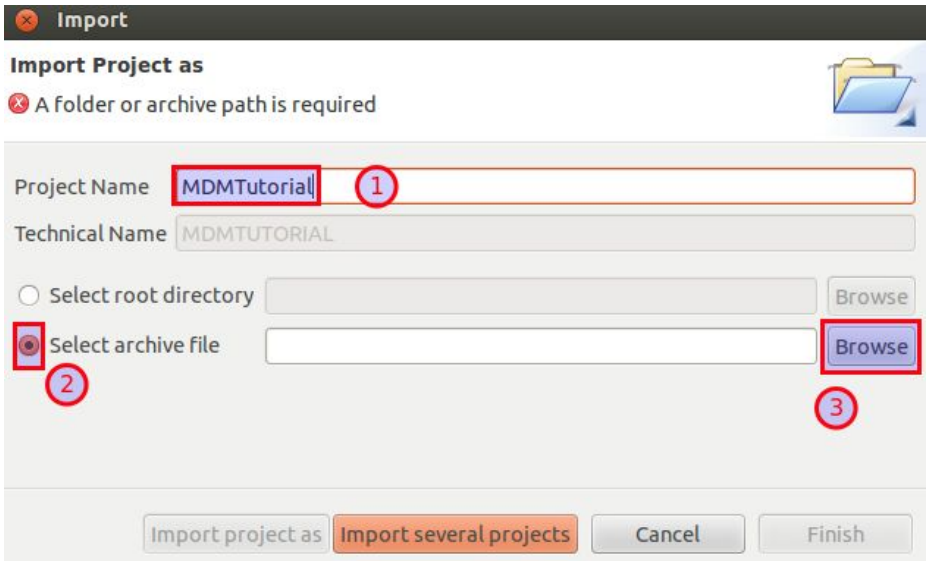
How to import the tutorial project archive

When you start up **TOS MDM Studio** you have the possibility to import existing projects. For your convenience I exported all my project files and made them available online so that you can compare your files to mine in case you get stuck at some point.

1. Download the **project archive** from here. In the menu bar click **File > Download**.
2. Start up **TOS MDM Studio**.
3. Press **Import ...**

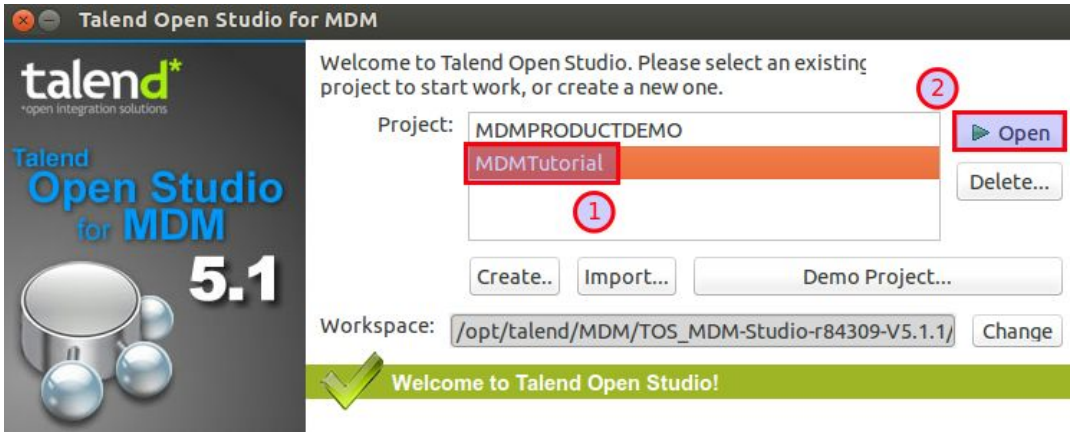


4. Assign a **project name** (in example *MDMTutorial*). Click on **Select archive file** and **Browse** for the archive file you just downloaded:



Click **Finish**.

5. The new project name should now show up on the startup window. Mark **MDMProject** and click **Open**:



Deploy solutions in following order:

1. data model
2. data container
3. view
4. DI job
5. processes
6. triggers