talend | Connecting the Data-Driven Enterprise

Lab Guide

TDM Optional Modules

Version 6.1

Talend Inc.
800 Bridge Parkway, Suite 200
Redwood City, CA 94065
United States
+1 (650) 539 3200

Welcome to Talend Training

Congratulations on choosing a Talend training module. Take a minute to review the following points to help you get the most from your experience.

Technical Difficulty

Instructor-Led

If you are following an instructor-led training (ILT) module, there will be periods for questions at regular intervals. However, if you need an answer in order to proceed with a particular lab, or if you encounter a situation with the software that prevents you from proceeding, don't hesitate to ask the instructor for assistance so it can be resolved quickly.

Self-Paced

If you are following a self-paced, on-demand training (ODT) module, and you need an answer in order to proceed with a particular lab, or you encounter a situation with the software that prevents you from proceeding with the training module, a Talend Support Engineer can provide assistance. Double-click the **Live Expert** icon on your desktop and follow the instructions to be placed in a queue. After a few minutes, a Support Engineer will contact you to determine your issue and help you on your way. Please be considerate of other students and only use this assistance if you are having difficulty with the training experience, not for general questions.

Exploring

Remember that you are interacting with an actual copy of the Talend software, not a simulation. Because of this, you may be tempted to perform tasks beyond the scope of the training module. Be aware that doing so can quickly derail your learning experience, leaving your project in a state that is not readily usable within the tutorial, or consuming your limited lab time before you have a chance to finish. For the best experience, stick to the tutorial steps! If you want to explore, feel free to do so with any time remaining after you've finished the tutorial (but note that you cannot receive assistance from Tech Support during such exploration).

Additional Resources

After completing this module, you may want to refer to the following additional resources to further clarify your understanding and refine and build upon the skills you have acquired:

- Talend product documentation (help.talend.com)
- Talend Forum (talendforge.org/)
- Documentation for the underlying technologies that Talend uses (such as Apache) and third-party applications that complement Talend products (such as MySQL Workbench)

This page intentionally left blank to ensure new chapters start on right (odd number) pages.

CONTENTS

LESSON 4 Mapping COBOL to XML

LESSON 1

Handling COBOL Redefined Fields (RDEF)

This chapter discusses the following.

Overview

Lesson Overview

Mainframe languages, especially COBOL, often reuse or **redefine** an area in a record to save space. A common example is a mailing list where the addressee may be either a person or a company, but never both. To include both an individual name field and a company name field would waste space, since only one of them would ever be filled, so the name field can be reused (redefined) as company name.

Furthermore, while a company name requires just one field, the individual name is usually composed of two fields, last name and first name. For example, bytes 1-12 might be last name, and bytes 13-20 the first name. But when redefined, bytes 1-20 would be the company name. Now you have two fields redefined by one field. COBOL can handle this just fine, but most PC applications don't use redefined fields and do not deal with this well.

Consider the highlighted fields in the following layout for a mailing list. Last Name and First Name are put into a group called `INDIVIDUAL-NAME`, then that group is redefined by `COMPANY-NAME`.

```
01  SUBSCRIBERS.
    05  INDIVIDUAL-NAME.
        10  LAST-NAME                          PIC X(12).
        10  FIRST-NAME                         PIC X(8).
    05  COMPANY-NAME REDEFINES INDIVIDUAL-NAME  PIC X(20).
    05  ADDRESS                                PIC X(20)
    05  CITY                                   PIC X(15)
    05  STATE                                  PIC X(2)
    05  ZIP                                    PIC X(5)
```

When a field is redefined, there should be a way to identify which definition was used in a given record. Unfortunately this is not always the case, and sometimes you have to try to figure it out by inspecting the data. The most common way to identify the use is to add a field to indicate it. For example, there might be a field called `TYPE-OF-NAME`, where `I` means the name is an **i**ndividual and `C` means it's a **c**ompany. This is how the layout might appear:

```
01  SUBSCRIBERS.
    05  TYPE-OF-NAME                           PIC X.
    05  INDIVIDUAL-NAME.
        10  LAST-NAME                          PIC X(12).
        10  FIRST-NAME                         PIC X(8).
    05  COMPANY-NAME REDEFINES INDIVIDUAL-NAME  PIC X(20).
    05  ADDRESS                                PIC X(20)
    05  CITY                                   PIC X(15)
    05  STATE                                  PIC X(2)
    05  ZIP                                    PIC X(5)
```

Let's look at these fields in two records, one addressed to John Smith and the other to Disc Interchange. The fields in those records would look like this:

```
ISmith       John    123 Main Street     Boston         MA01234
CDisc Interchange    124 Main Street     Boston         MA01234
```

If you ignore the redefined fields and print the name (on an envelope for example), one of the names will come out wrong. If you use the `COMPANY-NAME` definition, then **Disc Interchange** will be correct, but the mail to **John Smith** will be addressed to **Smith John**. If you use the `LAST-NAME, FIRST-NAME` fields, and print the first name followed by the last name, then the individual's name will be correct, like **John Smith**, but any company name will get scrambled, like **ange Disc Interch**.

Objectives

After completing this lesson, you will be able to:

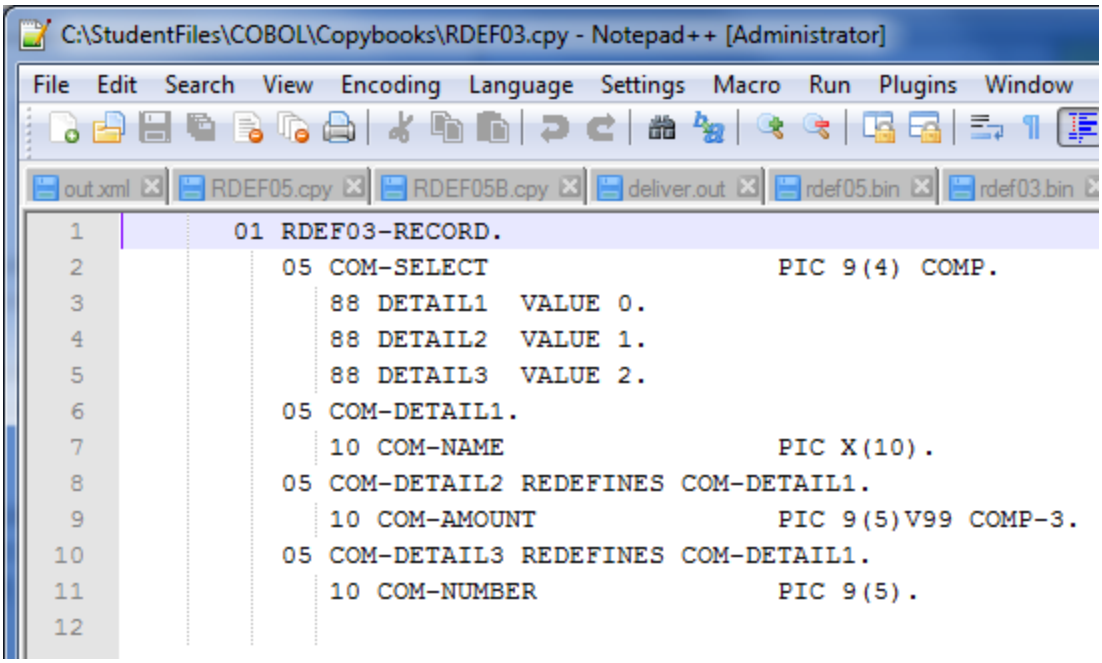
- Create a Structure that uses logical operations to parse the redefined fields in a complex COBOL Structure

Next Step

First, let's explore the sample data that will be used during this lab.

Exploring the Use Case

The *RDEF03.cpy* file contains a description for some simplified data.



- **COM-SELECT** can have 3 different values:
  - **0** indicates the record is to be treated as **DETAIL1**
  - **1** indicates the record is to be treated as **DETAIL2**
  - **2** indicates the record is to be treated as **DETAIL3**
- **10 COM-NAME** reserves space in memory according to the **PIC** clause (where **PIC X** indicates it is a string and **(10)** indicates 10 bytes).
- .**COM-DETAIL2 REDEFINES COM-DETAIL1** indicates there is another way to interpret those 10 bytes of data.
- **10 COM-AMOUNT** reserves space in memory: **9** indicates it is a number, **(5)** indicates 5 digits, and **COMP-3** indicates it is to be compressed.

As *Talend Data Mapper* (TDM) is not able to select the right alternative automatically, it has to be defined manually. So the person developing the data description and copybook decides which convention should be used.

In this example our developer decides that **COM-SELECT** indicates whether the record is to be treated as a **DETAIL1**, **DETAIL2** or **DETAIL3**.

The solution in TDM is to use an **Equal** function on the **IsPresent** field, instead of letting TDM set this field to **true** or **false** automatically.

Next Step

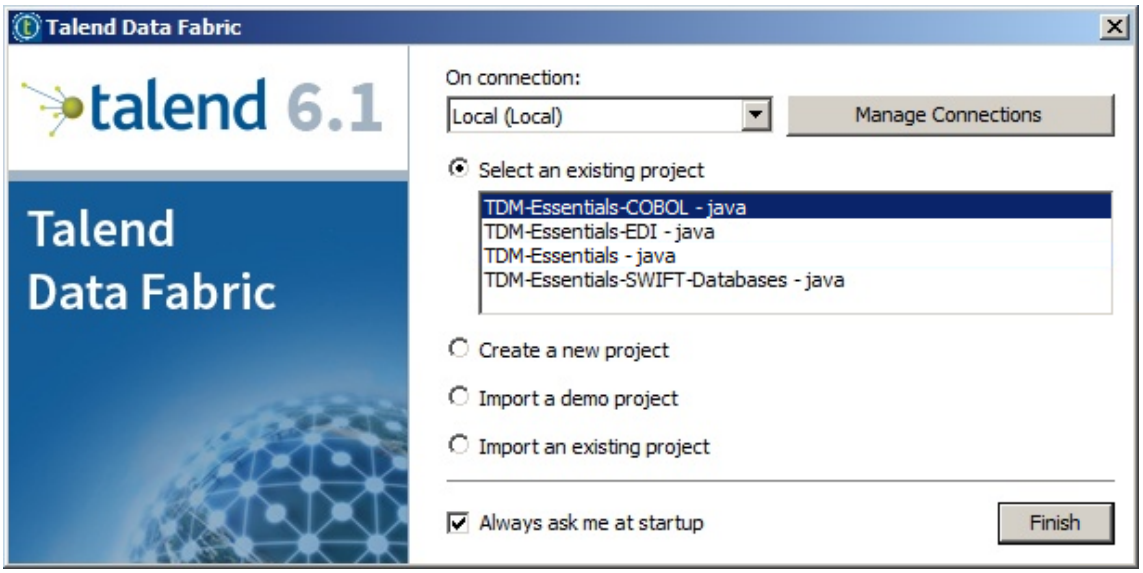
First, let's create the Structure matching this simplified data.

Creating the Structure

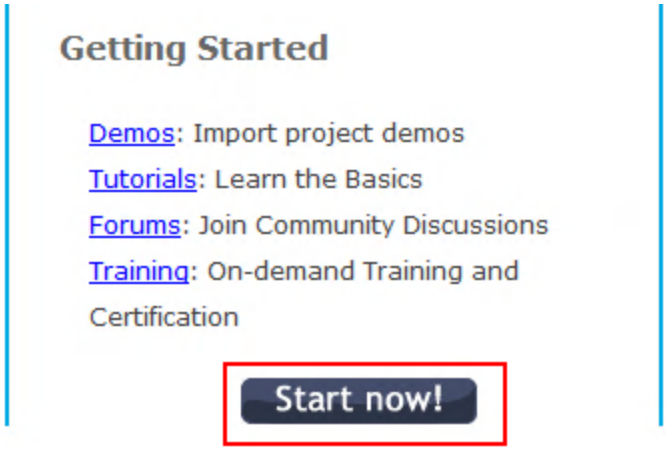
Starting Talend Studio

1. Click the **Talend Studio** link on the desktop.

2. Select the existing **TDM-Essentials-COBOL** project and click **Finish**.

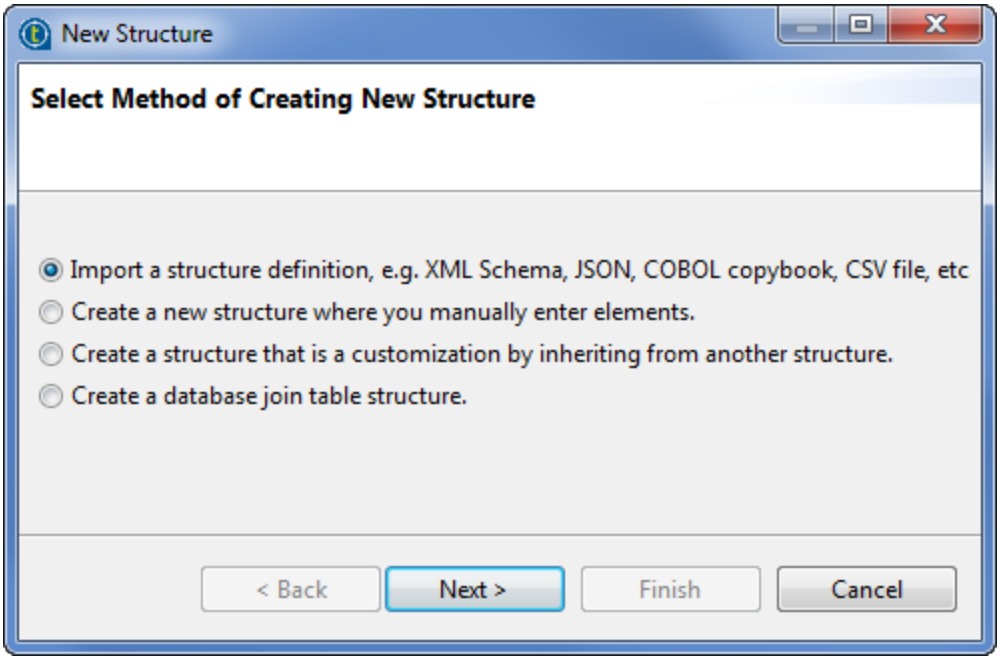


3. Click the **Start Now** button if the Welcome screen shows up. Otherwise, go to the next step.

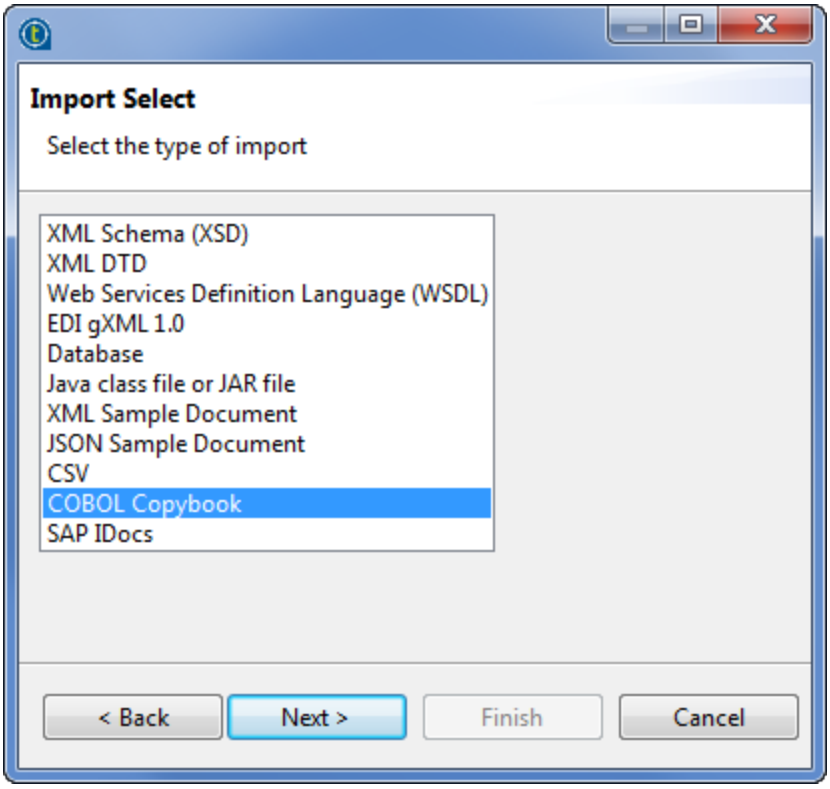


Creating the Structure

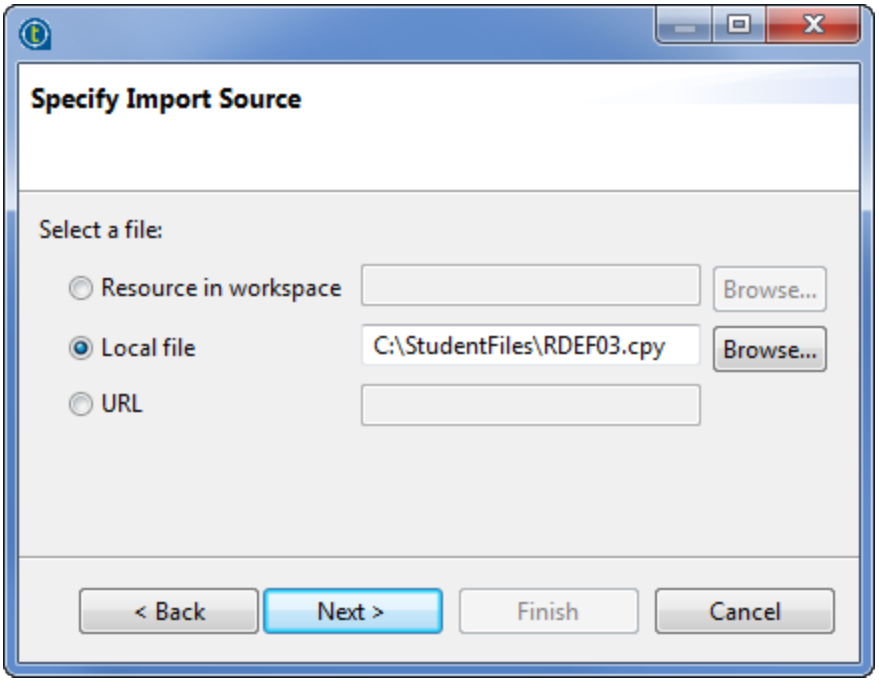
1. In the **Mapping** Perspective, right-click **Hierachical Mapper > Structures** and select **New > Structure**.
2. Select the first **Import a structure definition...** option and click **Next >**:



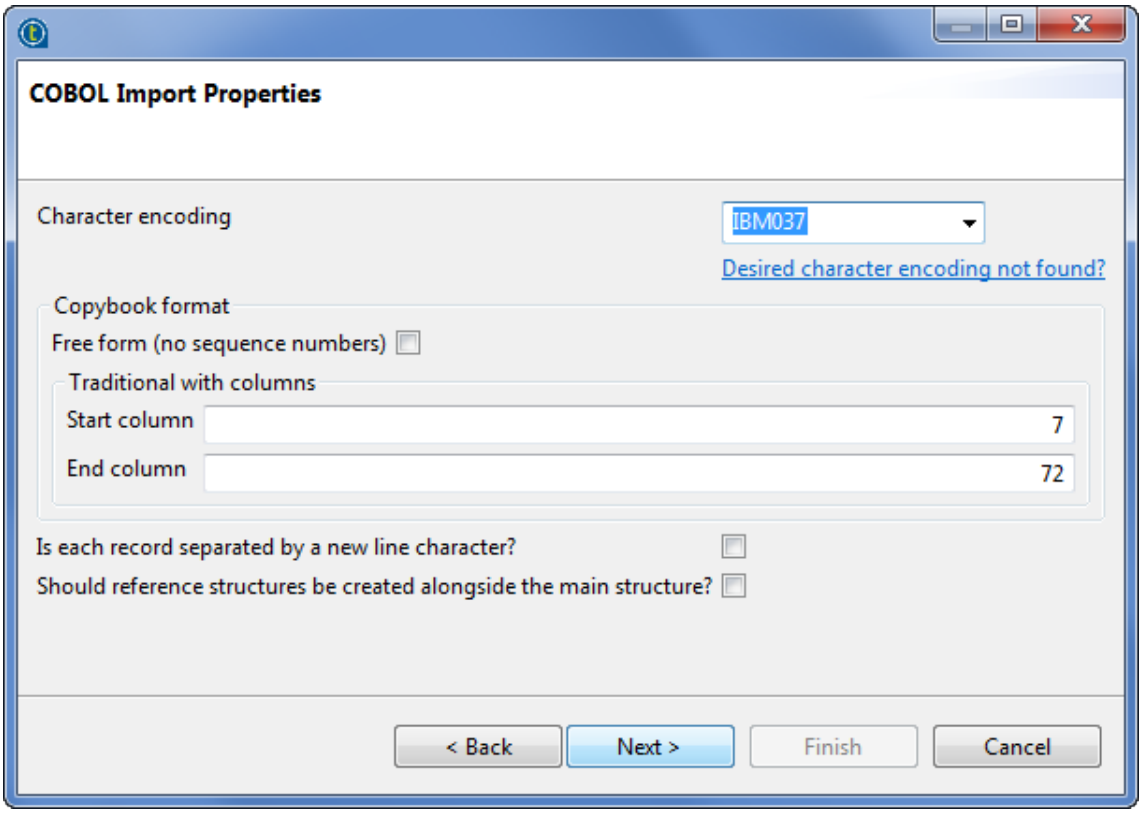
3. Select **COBOL Copybook** on the next screen and click **Next >**.



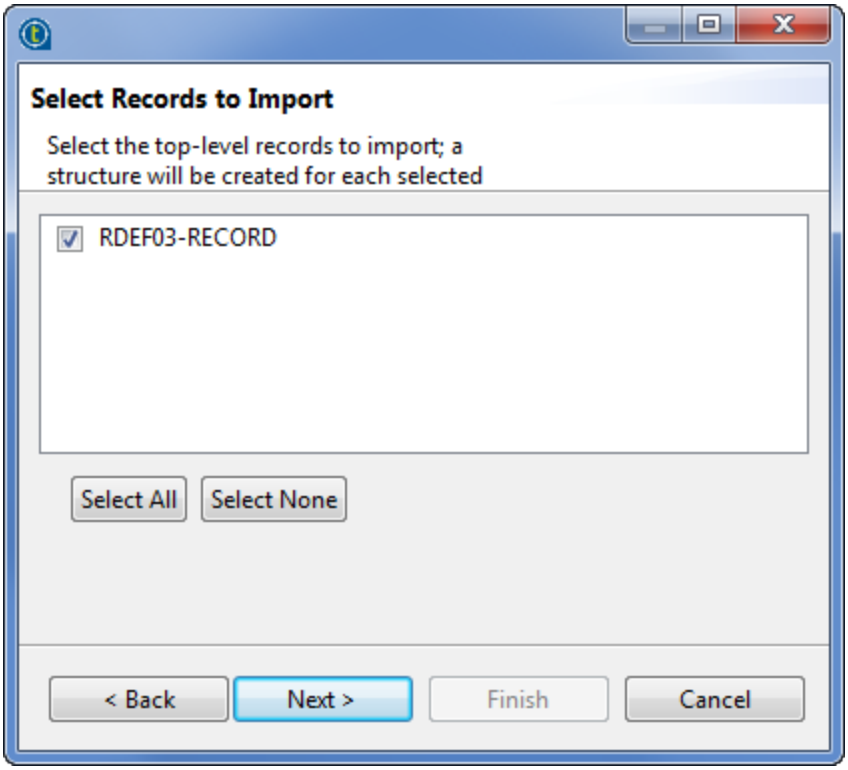
4. Select the file *RDEF03.cpy* from the *C:\StudentFiles* folder. and click **Next >**.



5. Select *IBM037* for the **Character encoding**, keep the default entries for **Start column** and **End column** and click **Next**:

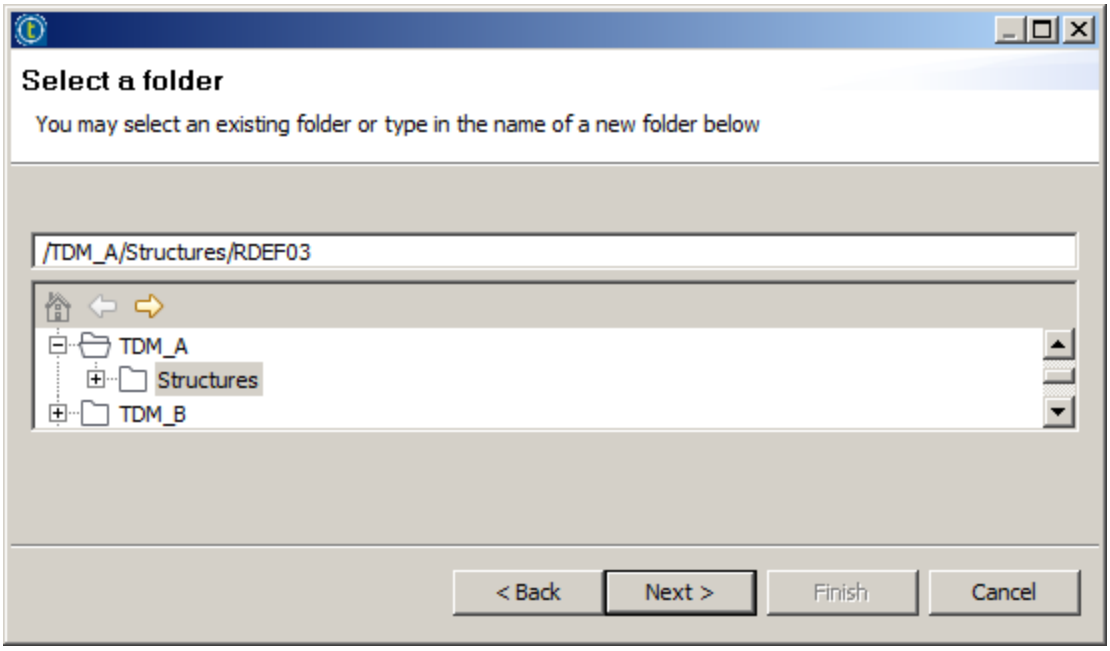


6. Check that the wizard identified a *RDEF03-RECORD* entry and click **Next >**:

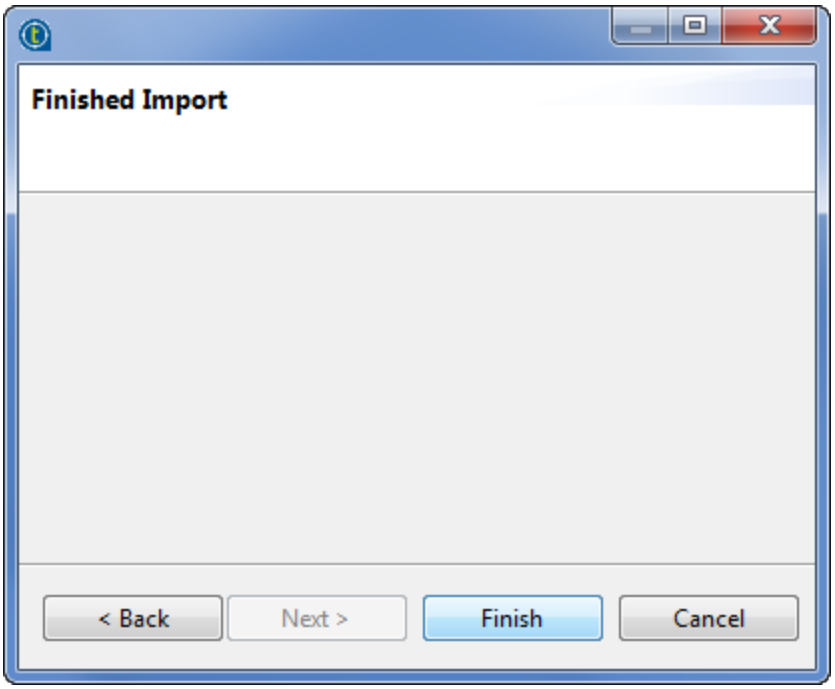


7. Click **Next >** again to generate loops for the identified record:

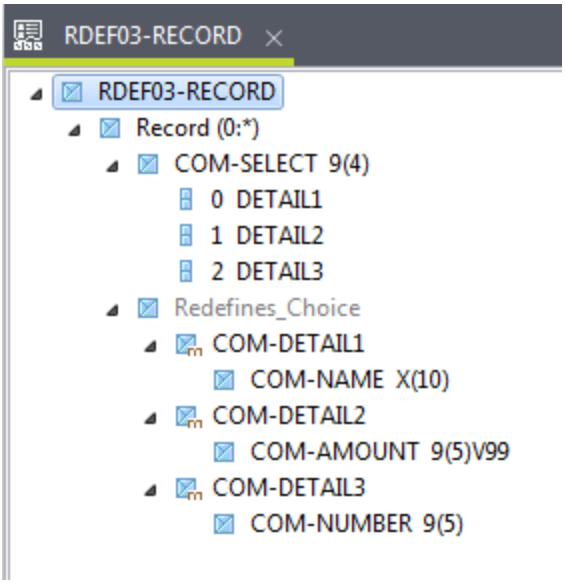
8. Select the *Structures* folder, keep the default *RDEF03* filename and click **Next >**:



9. Click **Finish** to close the wizard and create the Structure.



10. Double-click the new **Hierarchical Mapper > Structures > RDEF03-Record** Structure to display it:

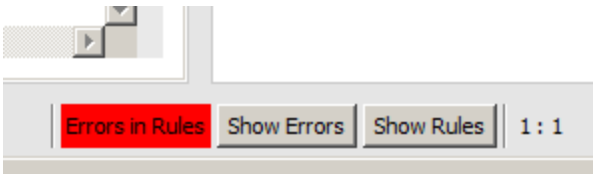


Next Step

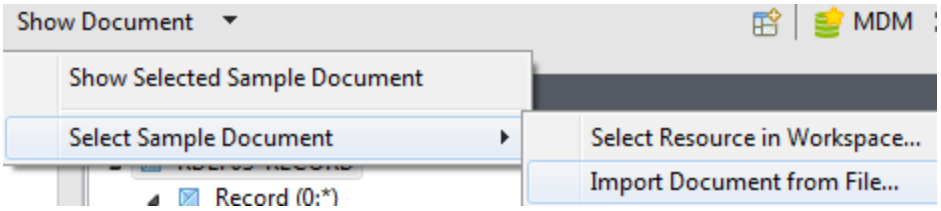
Now let's select the sample data file and configure the Structure settings in order to get a correct highlighting of the elements.

Configuring the Structure

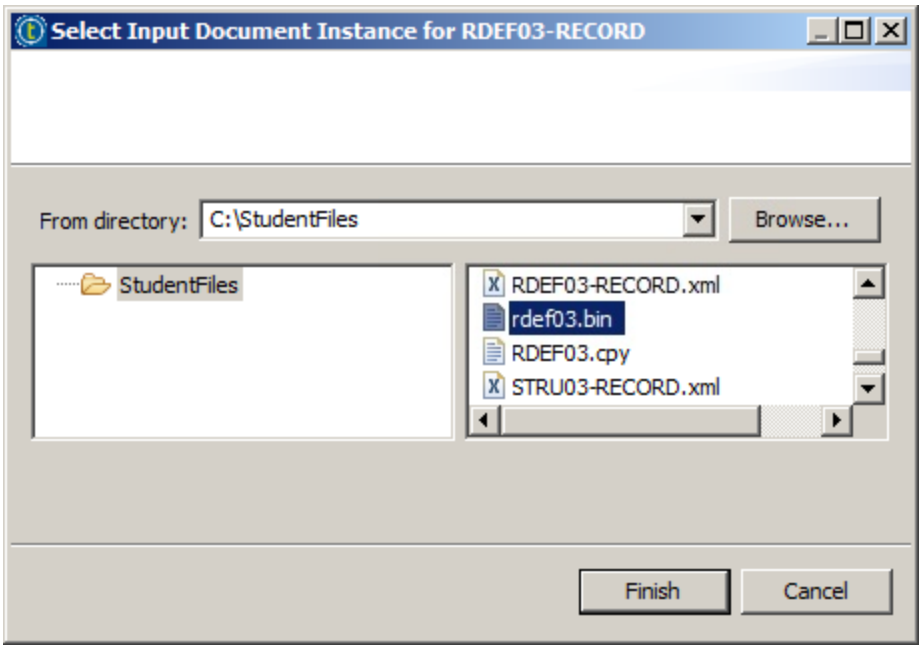
The sample data for this lab has 3 Redefines: `0` for `DETAIL1`, `1` for `DETAIL2` and `2` for `DETAIL3`. This definition is an individual setting from the person who developed this structure and data. If this information is missing you will not be able to setup a proper structure or mapping. A function will need to be entered to allow the sample data to be highlighted and mapped correctly. Note that, until that is done, you may see the **Errors in Rules** button in the status bar highlighted in red due to the configuration not yet representing the data.



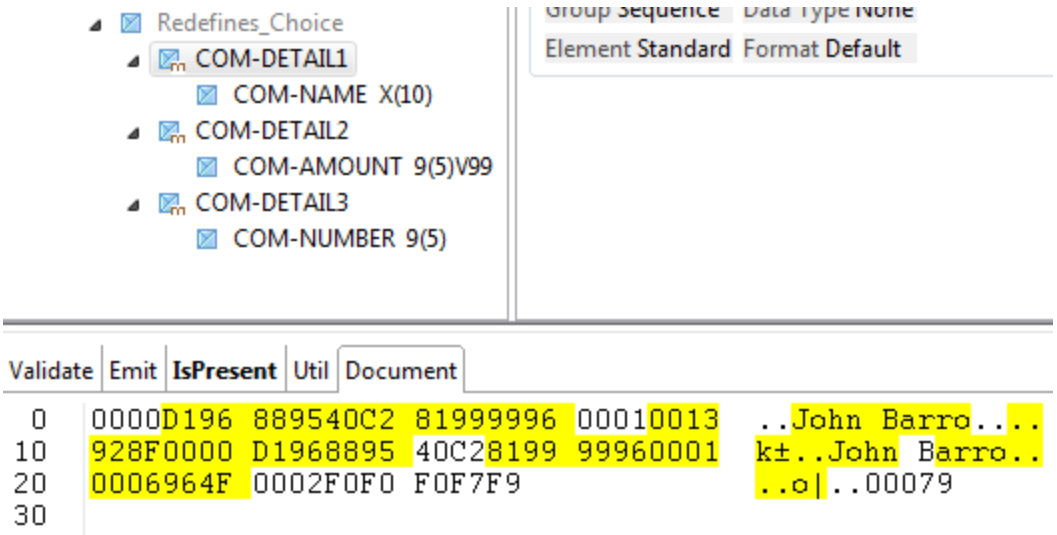
1. Expand the **Show Document** menu and click **Select Sample Document > Import Document from File...**.



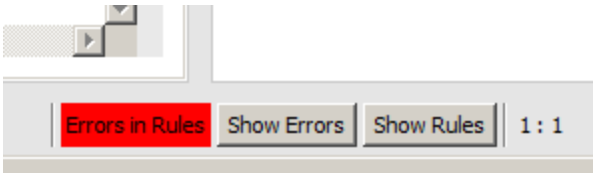
2. Select *rdef03.bin* from *C:\StudentFiles* and click *Finish*.



3. Now notice that only clicking *COM-DETAIL1* will highlight some data. Clicking *COM-DETAIL2* and *COM-DETAIL3* will not highlight any data:

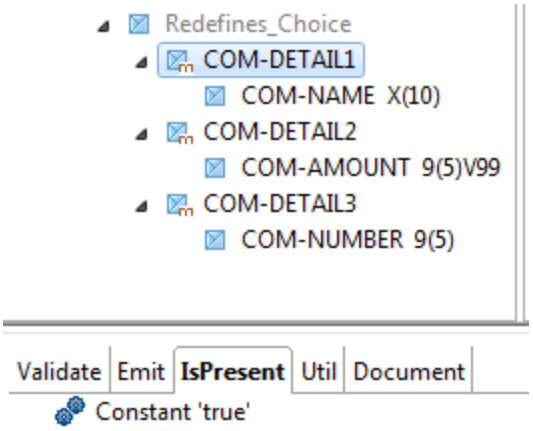


Also note the red **Errors in Rules** button in the status bar, indicating some error occurred during the parsing.

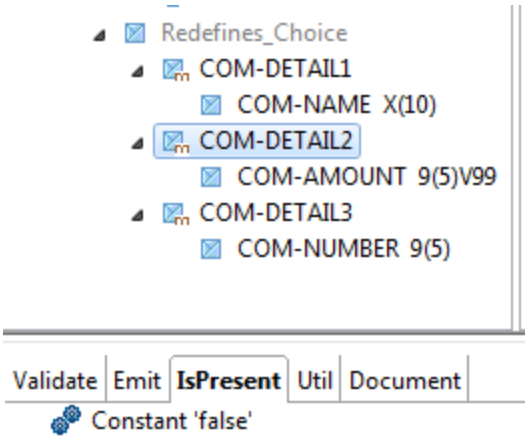


Let's fix these errors by setting up the conventions defined by the developer.

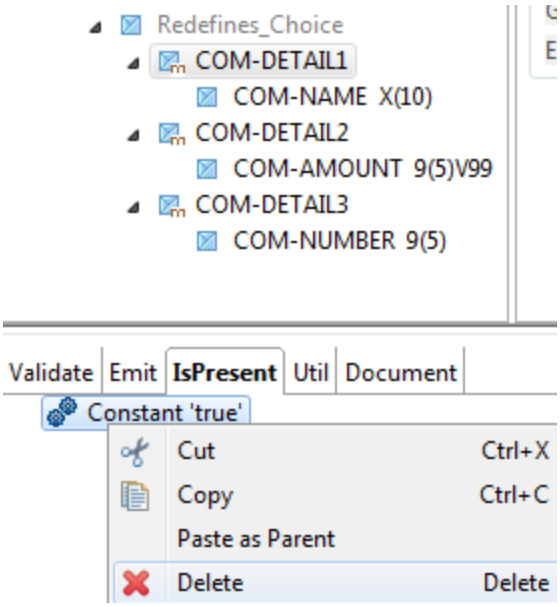
4. Select *COM-DETAIL1* and click the **IsPresent** tab for this element. Note that the Constant function is set to *true*.



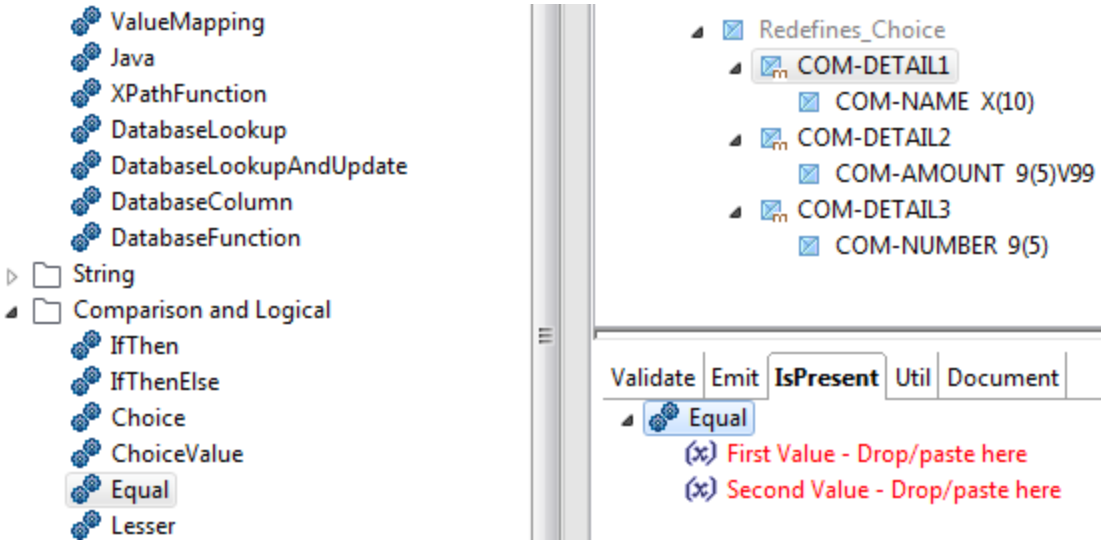
5. Also note that selecting *COM-DETAIL2* and *COM-DETAIL3* shows that the *Constant* function is set to *false* and no data is highlighted.



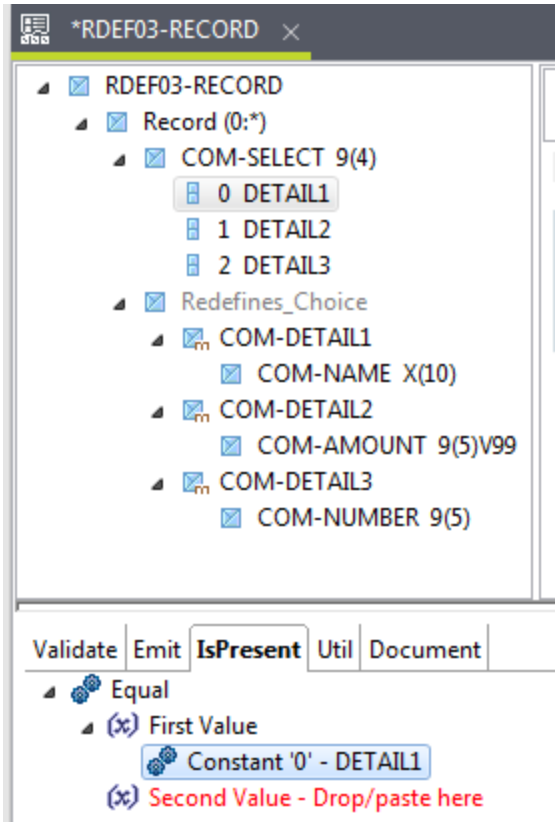
6. According to the settings of the *COM-SELECT* element, the value *0* should select and highlight *Detail1*, the value *1* should select and highlight *Detail2*, and the value *2* should select and highlight *Detail3*. This can be set manually using an **Equal** Function. Starting with **Detail1**, delete the *Constant* function from the **IsPresent** tab:



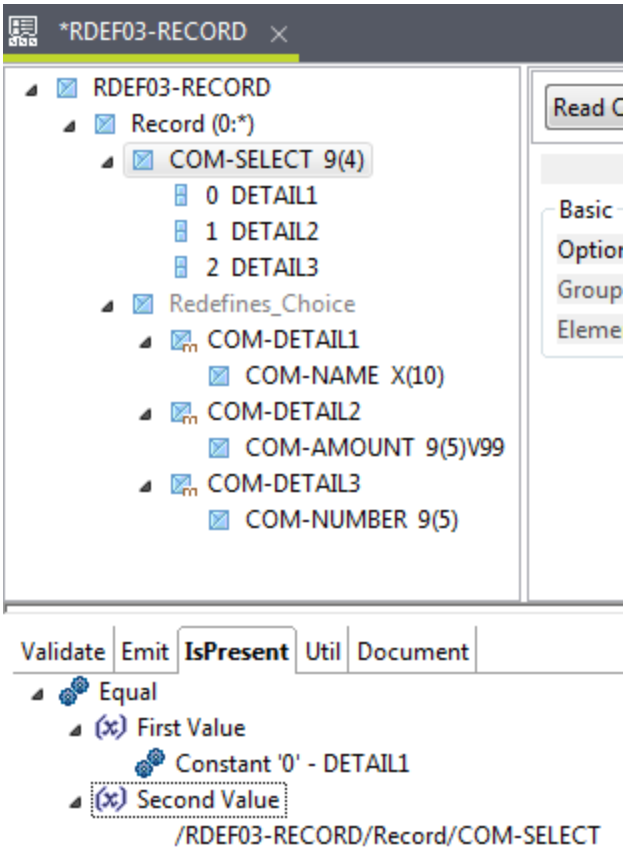
7. Drag a **Functions > Comparison and Logical > Equal** Function to the **IsPresent** tab of the *COM-DETAIL1* element:



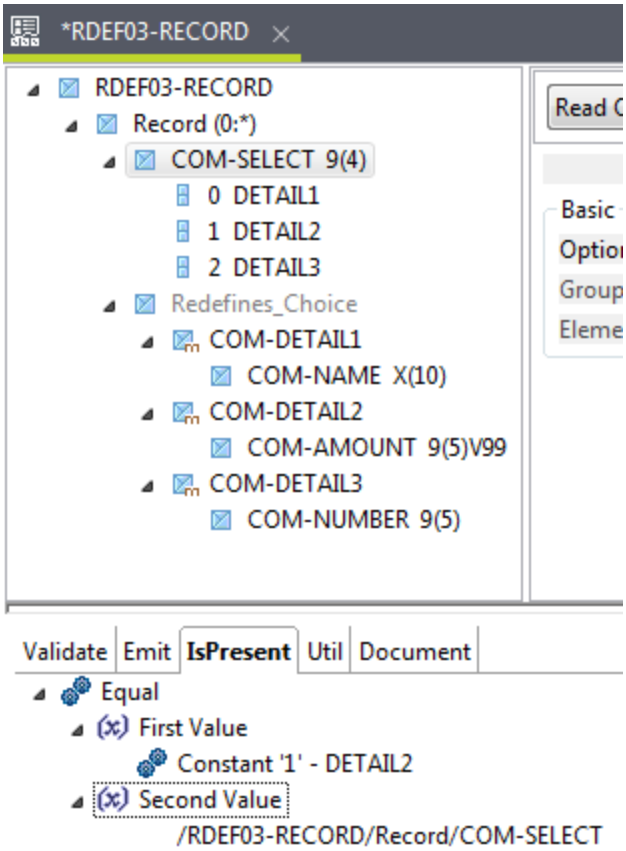
8. Drag *0 DETAIL1* from *COM-SELECT* onto the **First Value** field.



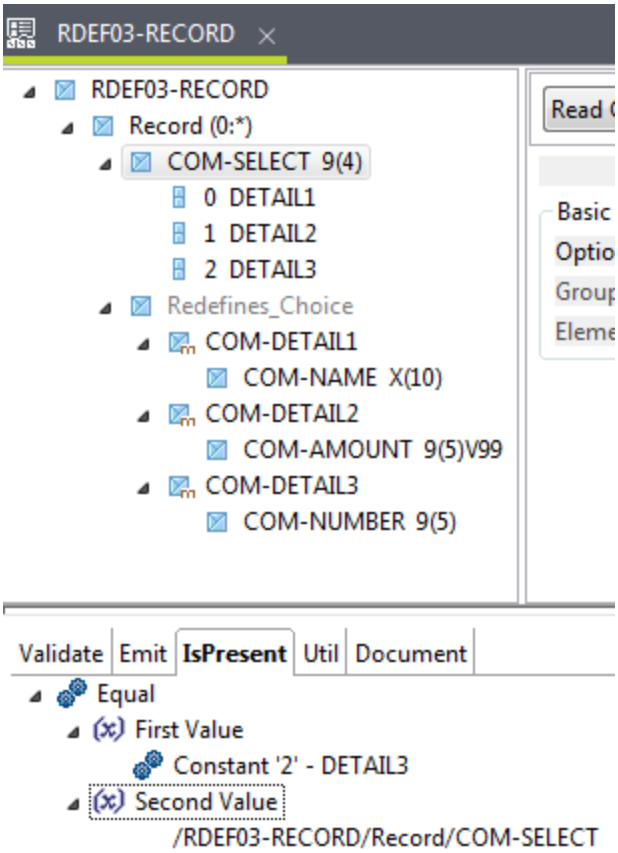
9. Drag *COM-SELECT* to the **Second Value** field.



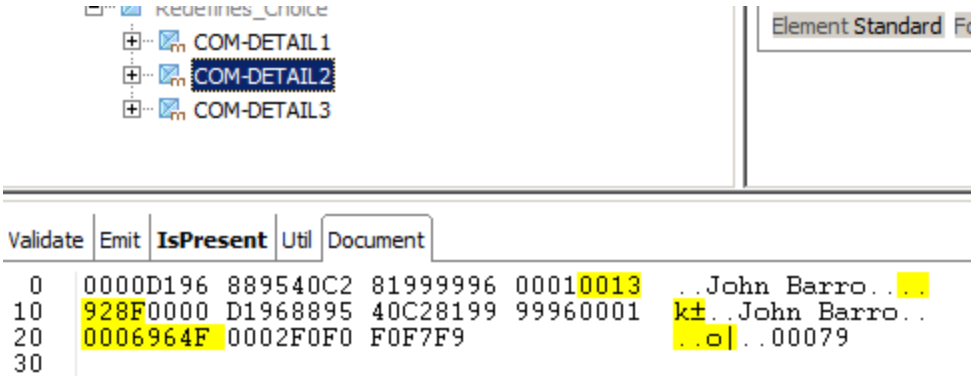
10. Repeat the last two steps for *COM-DETAIL2*, dragging *1 DETAIL2* onto the **First Value** field.



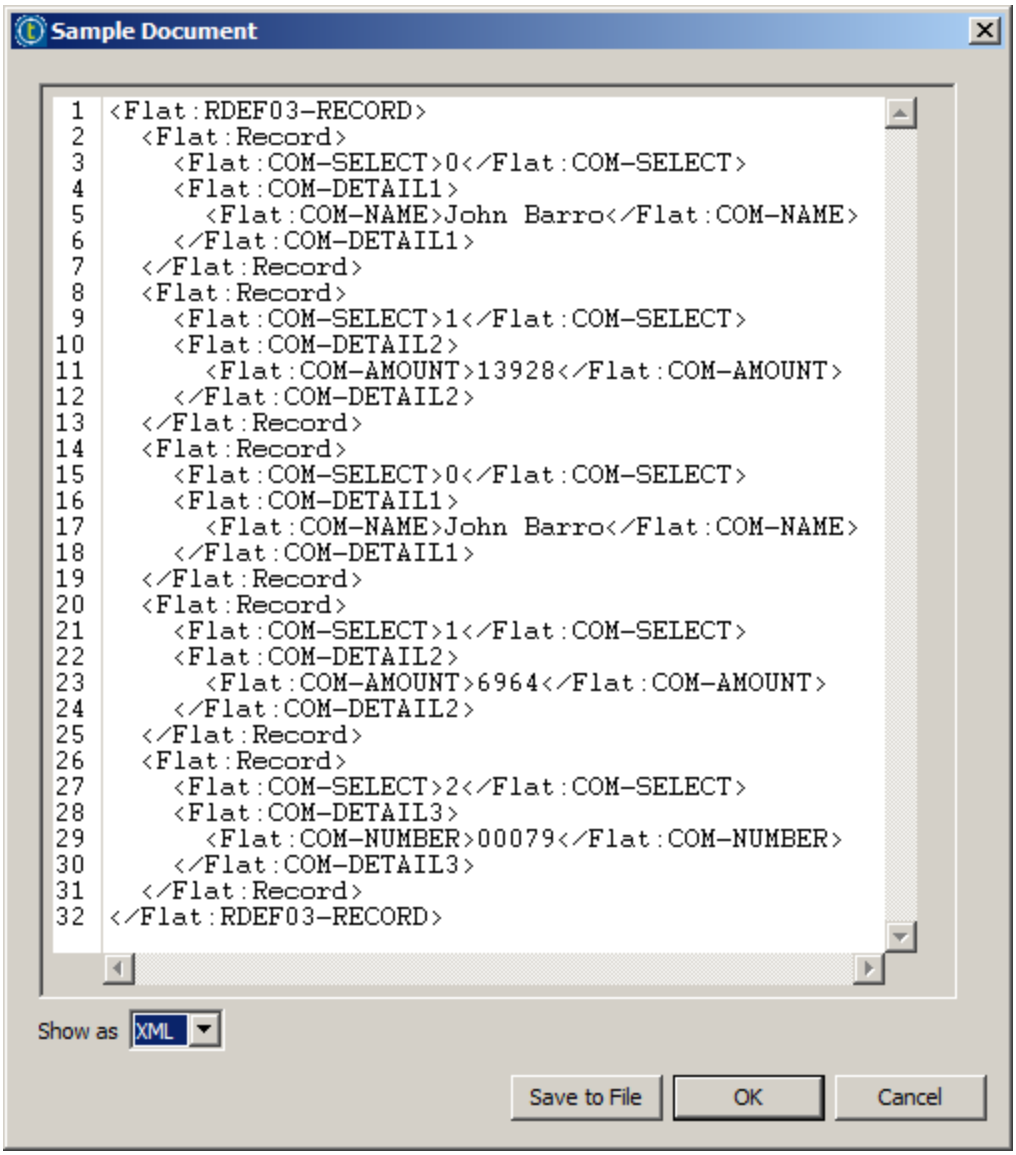
11. Repeat these steps again for *COM-DETAIL3*, dragging *2 DETAIL3* onto the **First Value** field.



12. Note that the red **Errors in Rules** button disappeared. Also note that clicking *COM-DETAIL1*, *COM-DETAIL2* and *COM-DETAIL3* successfully highlights data in the *Document* tab.



13. Select the *COM-DETAIL1* element and click the **Show Document** button. Select the **XML** Representation at the bottom left of the dialog. The data should be displayed correctly:



Next Step

This lesson is almost over. Head to the Wrap-Up section for a summary of the concepts reviewed in this lesson.

Wrap-Up

This short example demonstrates how you prepare a Structure and a Map for this common approach to storage reuse in a COBOL data description that uses the `REDEFINES` clause.

In this lesson, you learned how to:

- Create a Structure that uses logical operations to parse the redefined fields in a complex COBOL Structure

Next Step

Congratulations, you successfully completed this lesson. Click the **Check your status with this unit** button below in order to save your progress. Then click **Completed. Let's continue >** on the next screen to jump to the next lesson.

This page intentionally left blank to ensure new chapters start on right (odd number) pages.

LESSON 2

Mapping COBOL to Excel

This chapter discusses the following.

Overview

Lesson Overview

Imagine you have to convert some existing COBOL data to an Excel Output. This result can be achieved by creating a Data Integration (DI) Job that relies on a **tHMap** Component to create a COBOL to Excel mapping.

Objectives

After completing this lesson, you will be able to:

- Import a COBOL Structure
- Validate an imported Structure against some sample data
- Create a DI Job that relies on a **tHMap** Component to map a COBOL Structure to a custom output Schema stored as an Excel file
- Use the **tHMap** wizard to create a Map that reuses existing Structures or Schemas as input and output

Next Step

First, let's import an existing COBOL Structure and validate it against some sample data.

Exploring the Use Case

This lab uses an existing COBOL data description called *STRU03-RECORD* comprised of the following:

- an unsigned integer elementary item called `COM-NUMBER`,
- an alphanumeric elementary item called `COM-NAME`,
- an unsigned decimal elementary item called `COM-AMOUNT`, and
- a group item repeating 5 times called `COM-ARRAY` comprised of
- a signed integer elementary item called `COM-ITEM1` and
- an alphanumeric elementary item called `COM-ITEM2`

The non-repeating portion of the COBOL Copybook will be parsed and represented in an Excel spreadsheet showing data triplets like:

```
000075 Earl Curzon 278.56
000076 John Franklin 139.28
000077 Francis Beaufort 92.85
000078 John Franklin 69.64
000079 Francis Beaufort 55.71
```

To do this, you will first create a Structure in the **Mapping** perspective from the copy file. Secondly, you will create a Job in the **Integration** perspective to configure components as follows:

tFileRawInputcomponent > tHMap > tMap > tOutputFileExcel

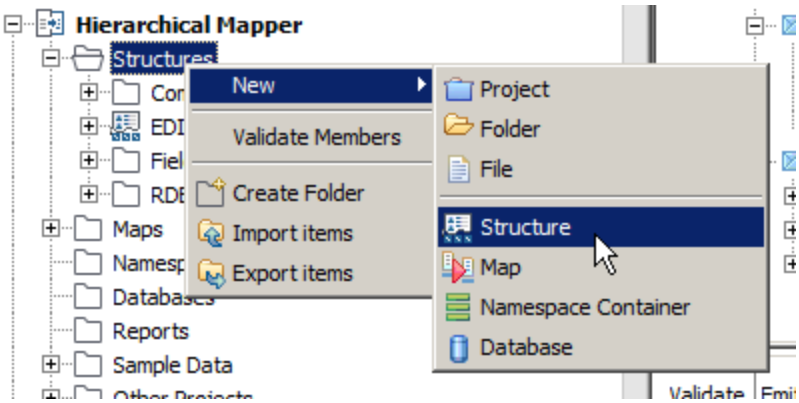
Finally, you will configure the **tHMap** component to specify each Structure and then automatically generate the Map into the **Mapping** perspective.

Next Step

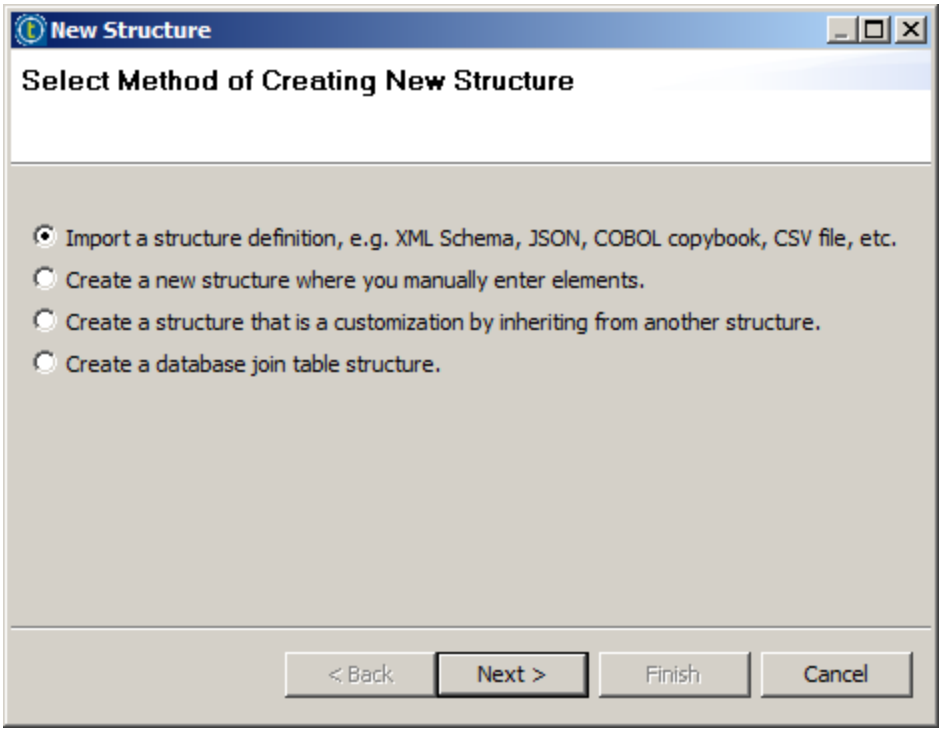
First, let's create a Structure from the *STRU03-RECORD* file.

Creating the Structure

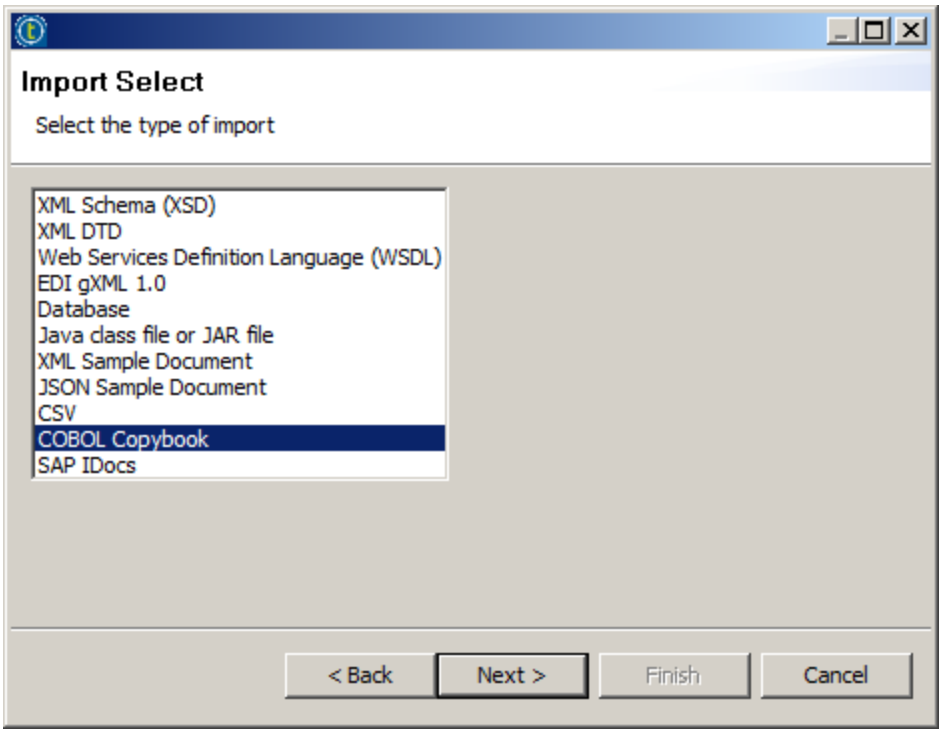
1. Right-click **Hierarchical Mapper > Structures** and select **New > Structure**.



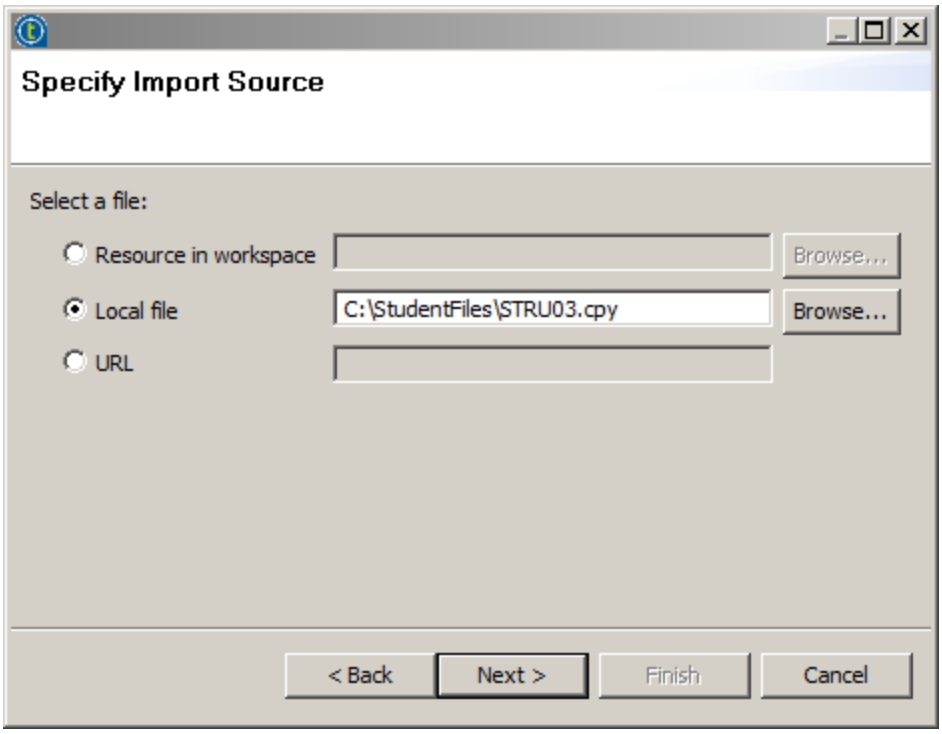
2. Select the **Import a structure definition, e.g. XML Schema, JSON, COBOL copybook, CSV file, etc.** option and click **Next >**.



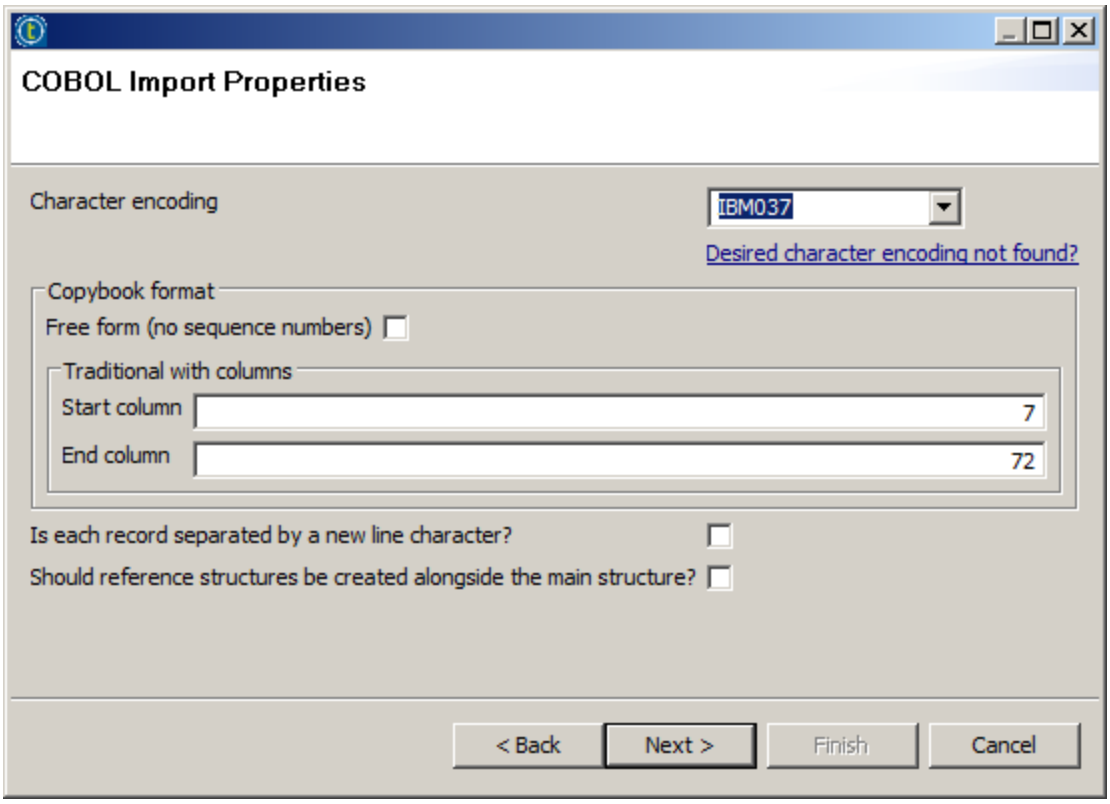
3. Select **COBOL copybook** and click **Next >**.



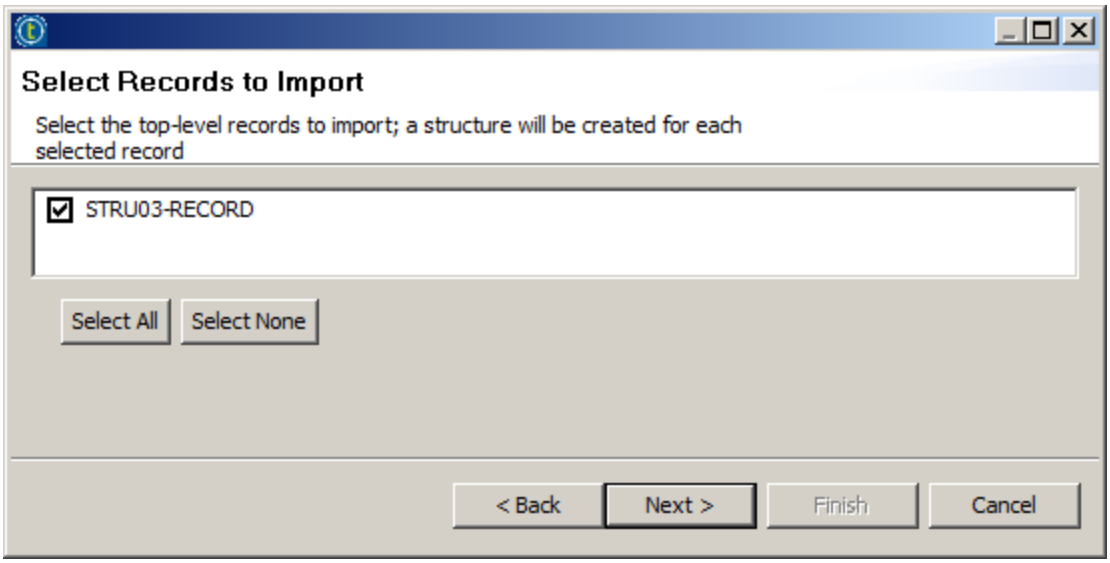
4. Select **Local file**, click **Browse...** to select *STRU03.cpy* in *C:\StudentFiles* then click **Open** and **Next >**.



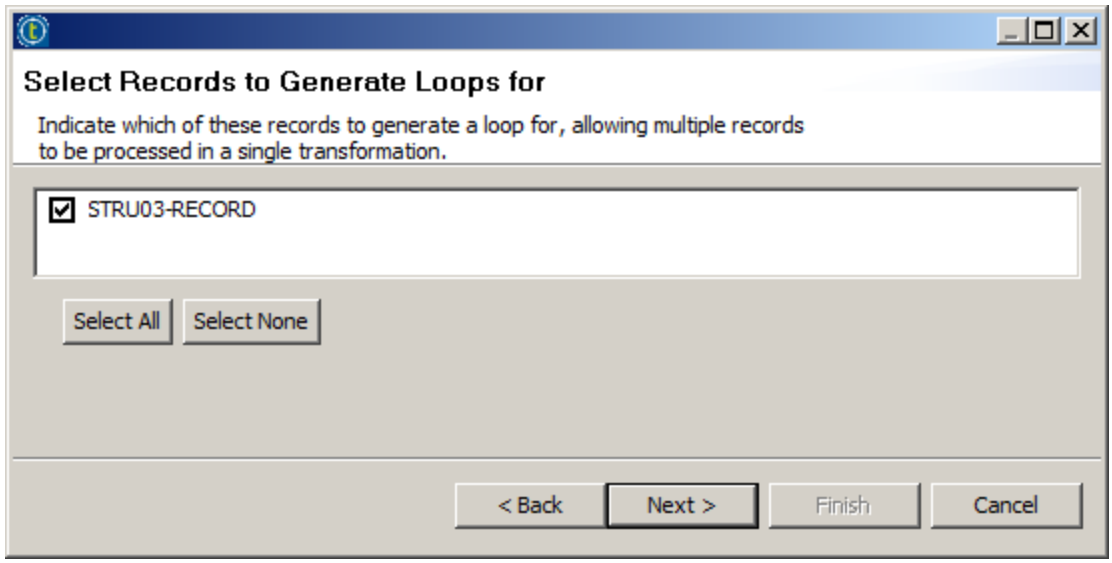
5. In the **COBOL Import Properties** screen, select *IBM037* for the **Character encoding** and click **Next >**.



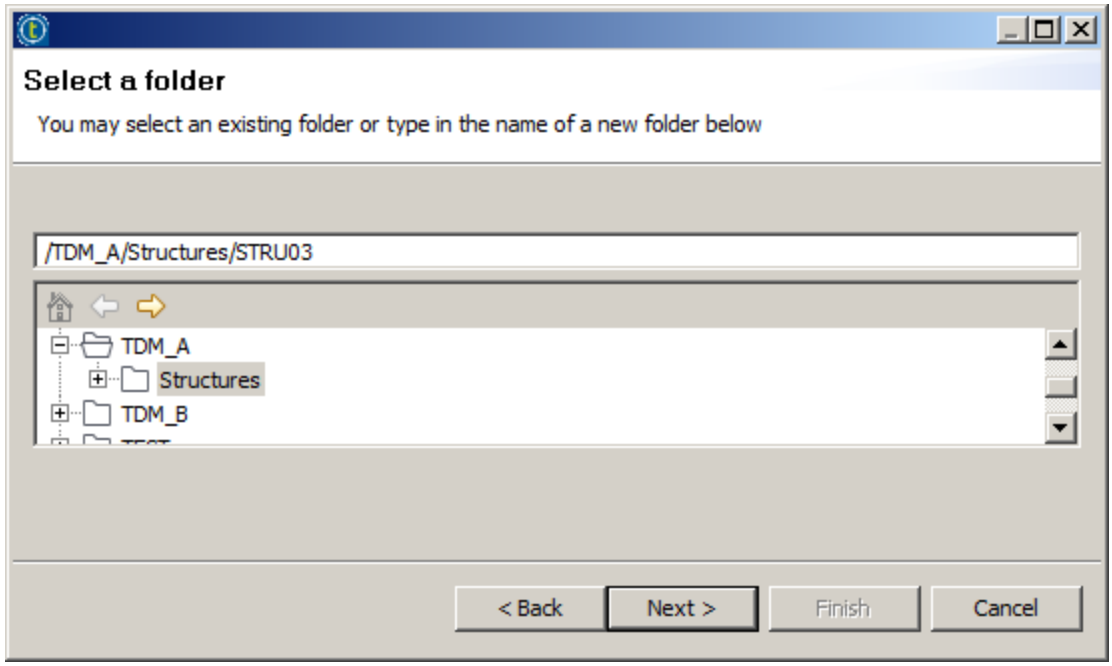
6. Make sure the **STRU03-RECORD** entry is marked and click **Next >**.



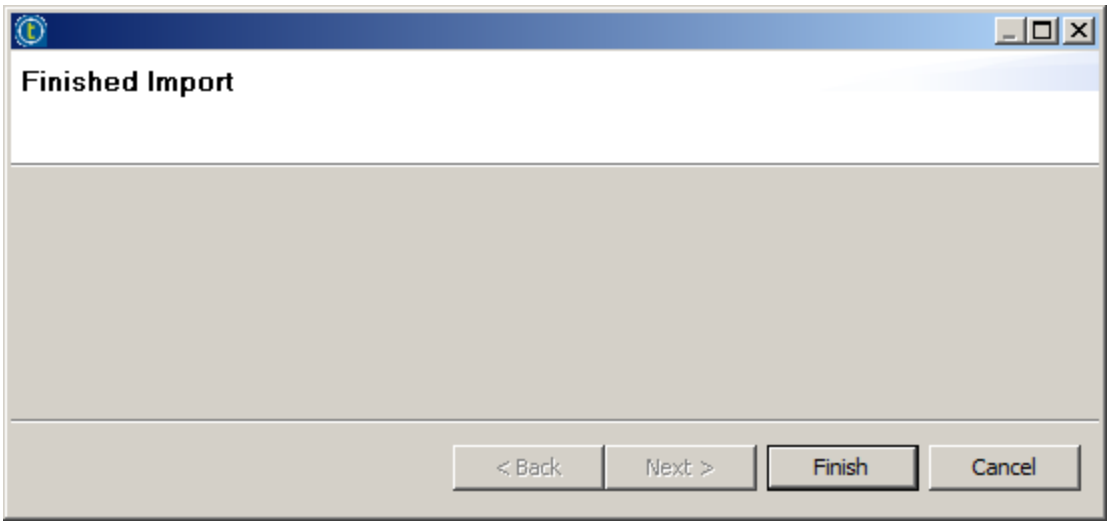
7. Make sure the **STRU03-RECORD** is selected again on the next screen, so a loop is created for it, and click **Next >**.



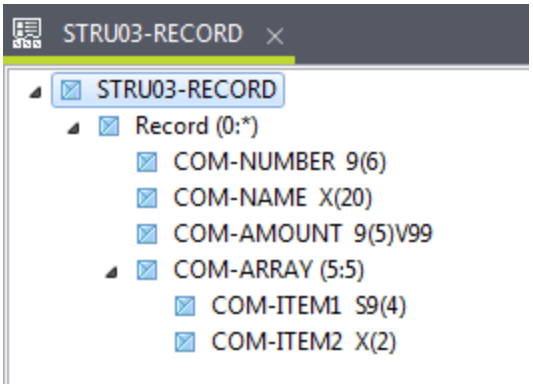
8. Click **Next** to accept the default location:



9. Click **Finish** to actually create the Structure.



10. Double-click and expand the new **Hierarchical Mapper > Structures > STRU03-RECORD** Structure and examine it.



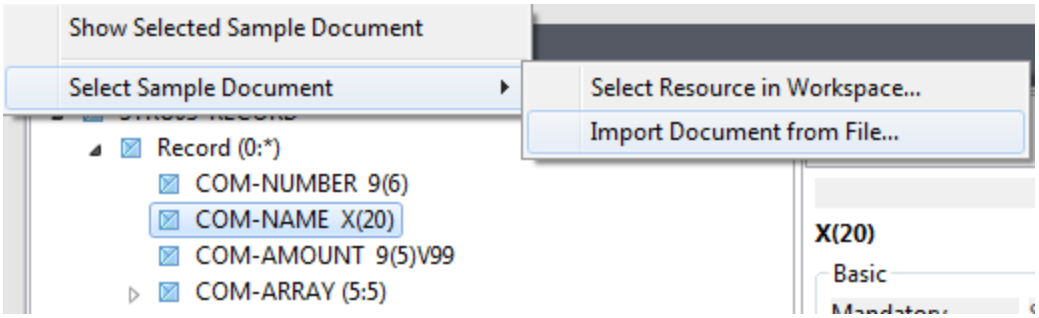
Next Step

Now that the input Structure is created, let's validate it by displaying some sample data and checking the right fields are highlighted.

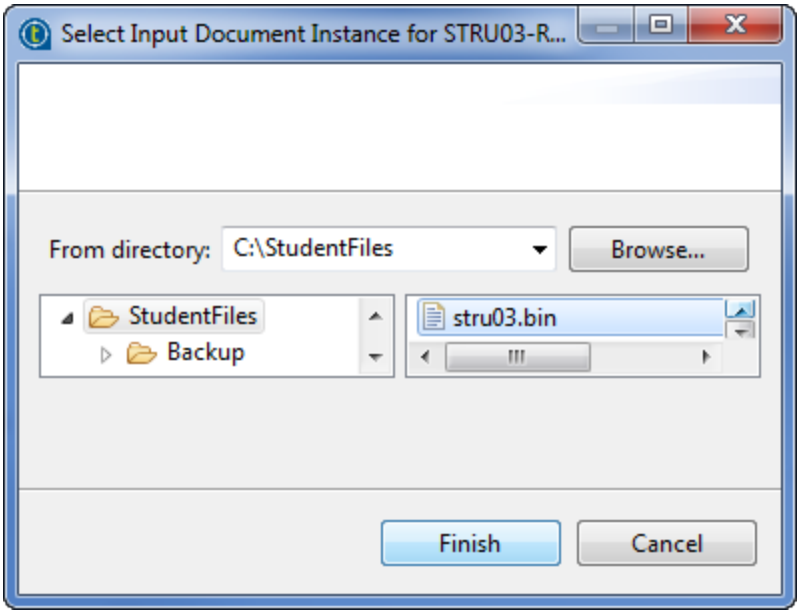
Validating the Structure

Let's validate the Structure just created by displaying some sample data and checking that the fields are correctly highlighted in the Document View.

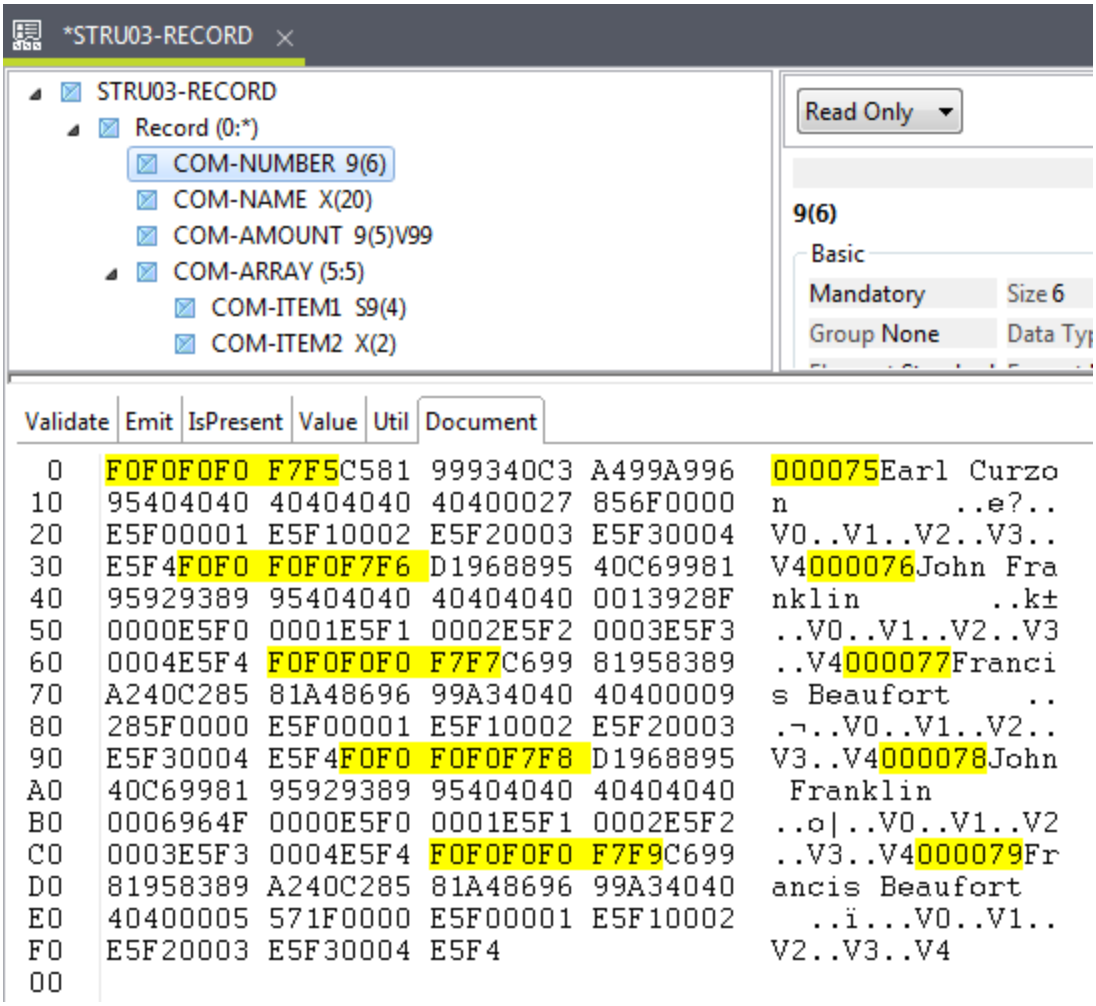
1. Expand the **Show Document** menu and select **Select Sample Document > Import Document from File...** :



2. Select the *stru03.bin* file from *C:\StudentFiles* then click **Finish**.



3. Select the *COM-NUMBER* element in the Structure and check that the following data is highlighted in the **Document** View.



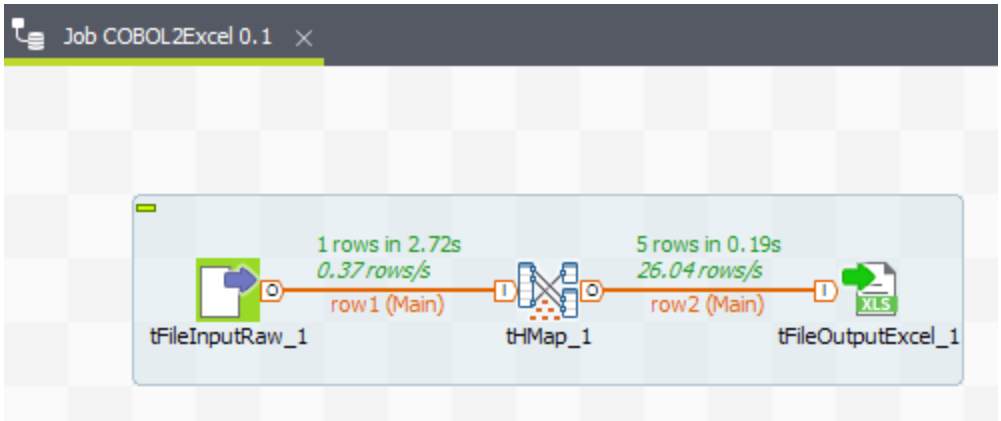
Try other elements and check the correct data is highlighted. Note that data is not always displayed in readable form as it is stored in a binary and compressed format.

Next Step

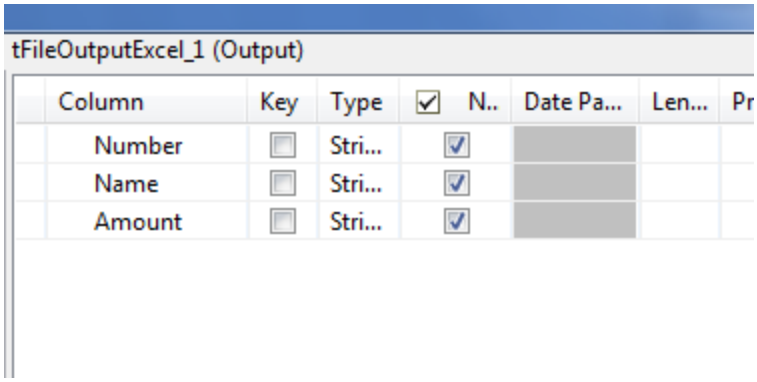
Now that the input Structure is created and validated, let's create a DI Job to perform the COBOL to Excel transformation.

Creating the DI Job

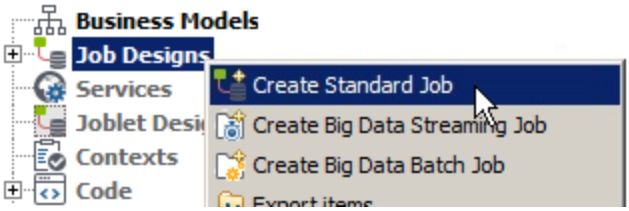
This section details the creation of a DI Job to convert COBOL input to Excel output using a **tHMap** Component. The final Job will look like:



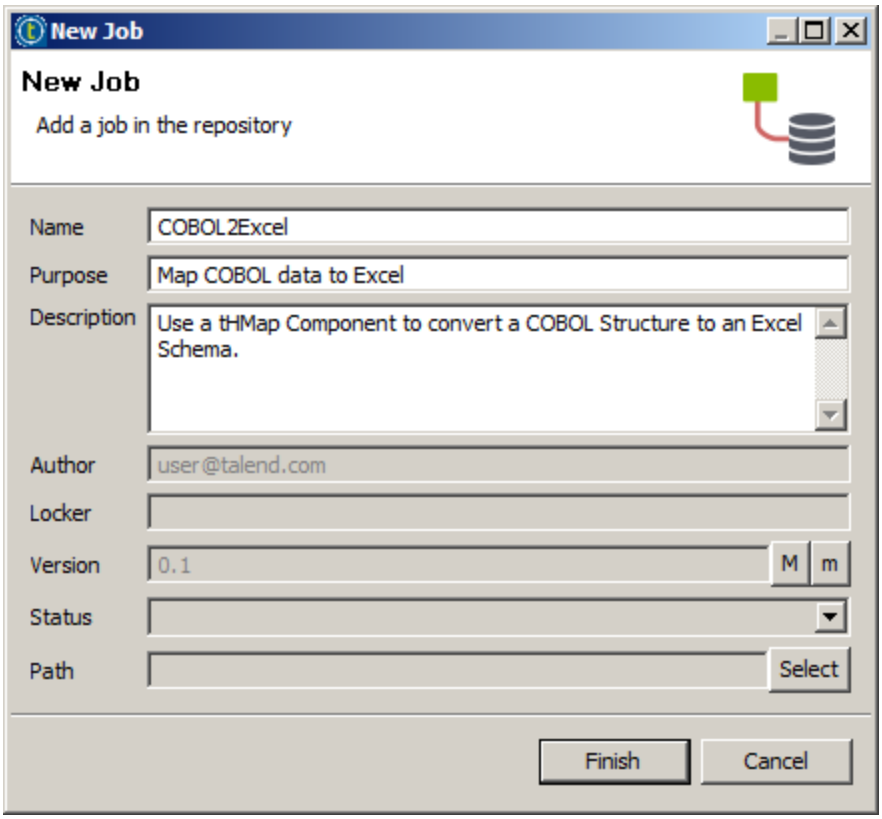
The output Schema used to create the Excel file looks like:



1. Switch to the **Integration** perspective, right-click **Job Designs** then select **Create Job**.

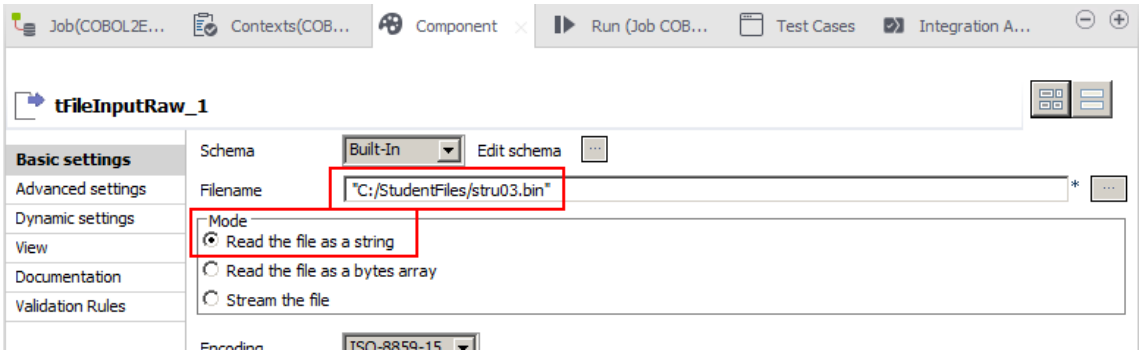


2. Configure the Job as illustrated below.

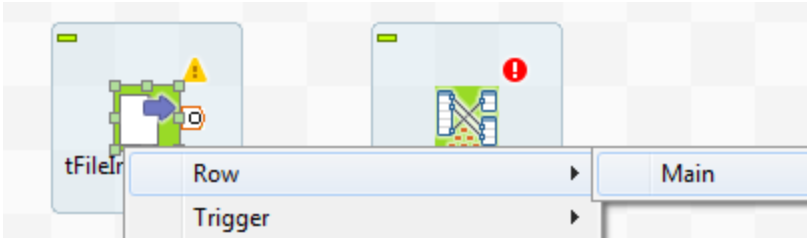


3. Now place a **tFileInputRaw** Component on the Design area, and double-click it to display the **Component** View.
4. Set **Mode** to *Read the file as a string*.

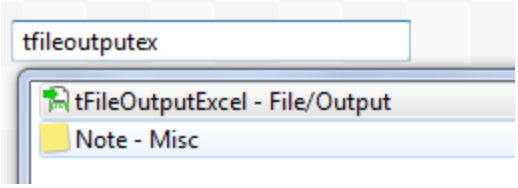
   Click the ... button to the right of the **File name** field, select *stru03.bin* in *C:\StudentFiles* and click **Open**.



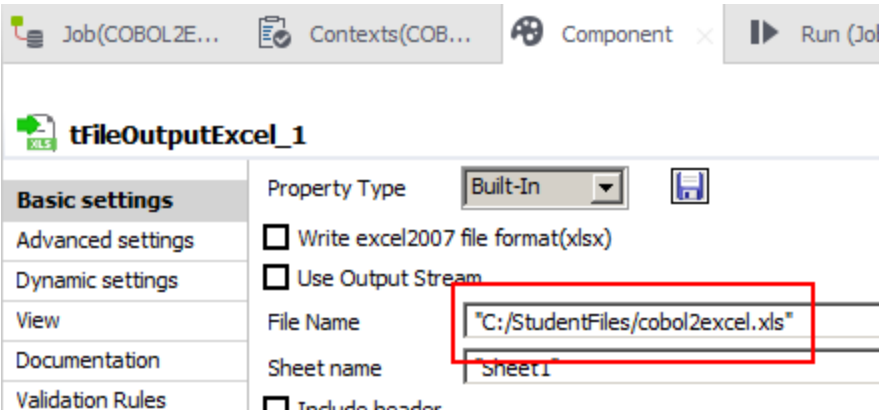
5. Place a **tHMap** Component to the right of the **tFileInputRaw** Component on your workspace. Right-click **tFileInputRaw**, select **Row > Main** and drop the link on the **tHMap** Component.



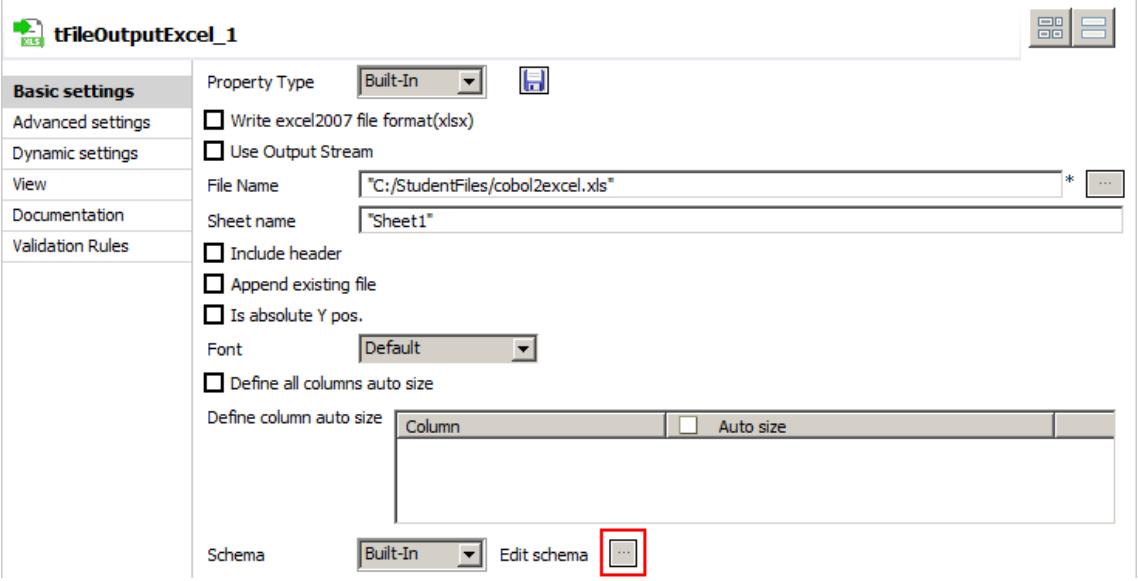
6. Finally, place a **tFileOutputExcel** Component to the right of **tHMap** and double-click it to display the **Component** View.



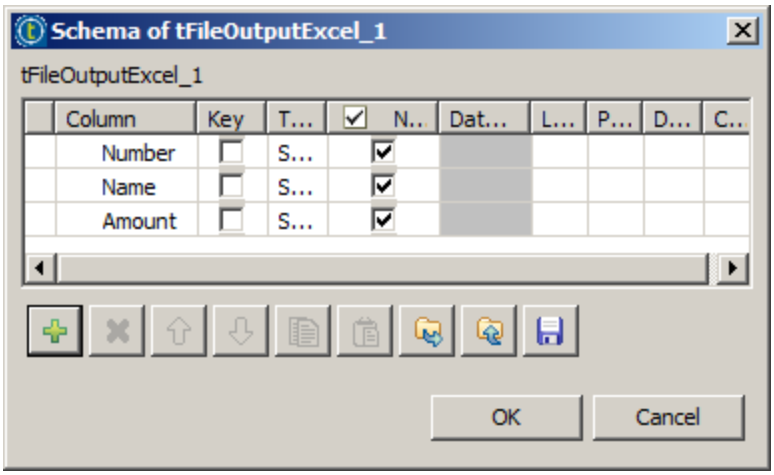
7. Click the **...** button next to the **File Name** field and browse to *C:\StudentFiles*. Change **File name** to *cobol2excel.xls* and click **Open**.



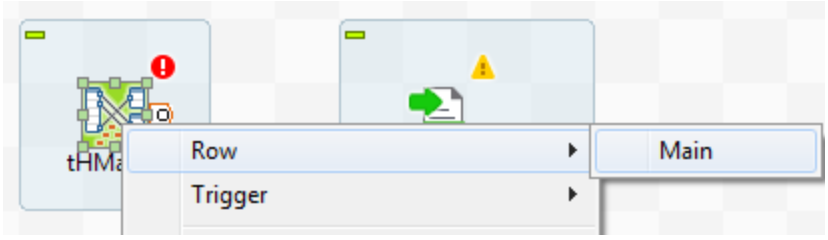
8. Now let's create the output Schema by clicking the **...** button next to **Edit schema**.

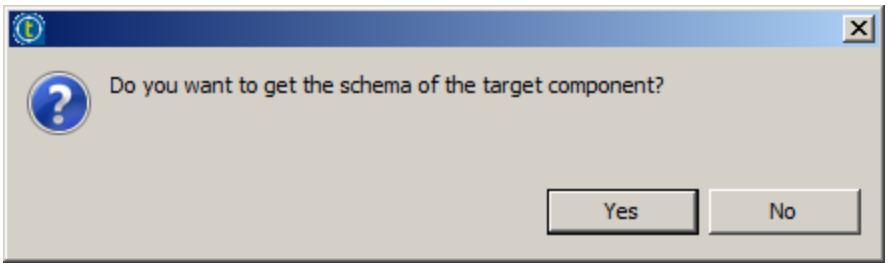


9. The default Schema is empty. Click the green **+** button at the bottom left three times to add three fields. Change the Column labels to **Number**, **Name** and **Amount**, then click **OK**:

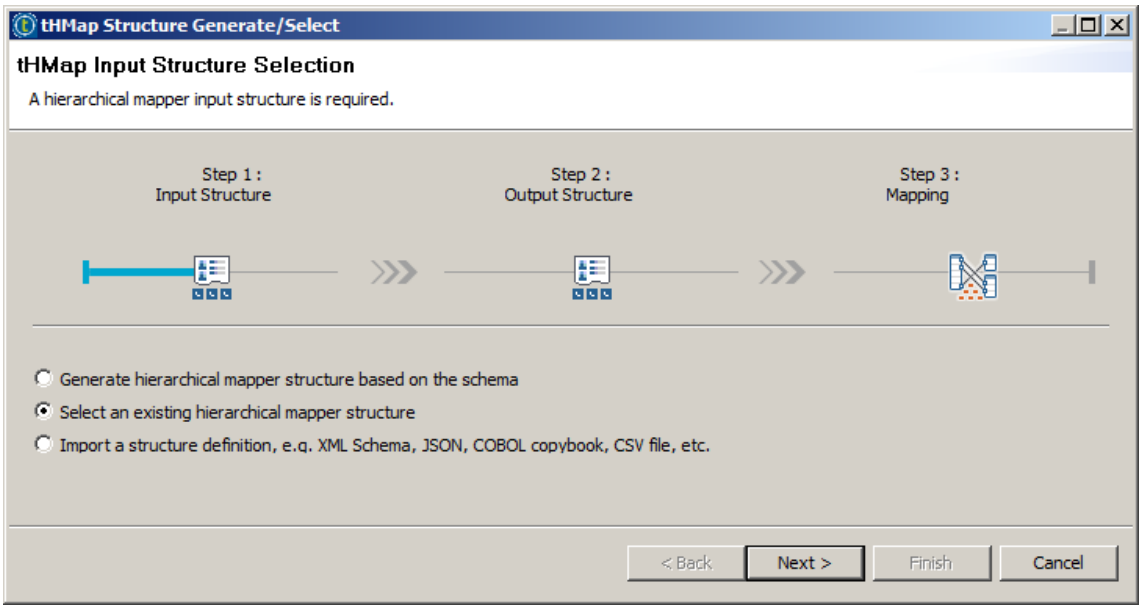


10. Right-click **tHMap**, select **Row > Main** and drop the link on **tFileOutputExcel**. Answer **Yes** to propagate the output Schema from **tFileOutputExcel** to **tHMap**.

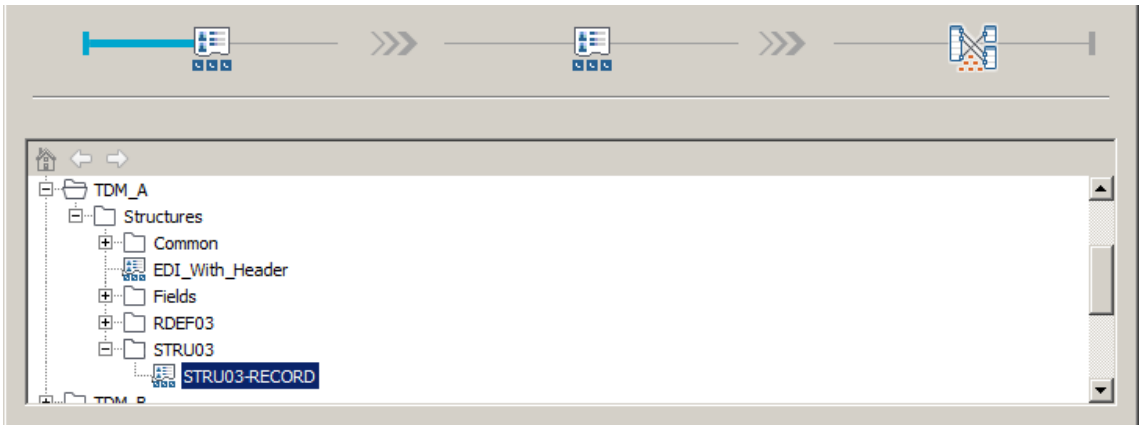




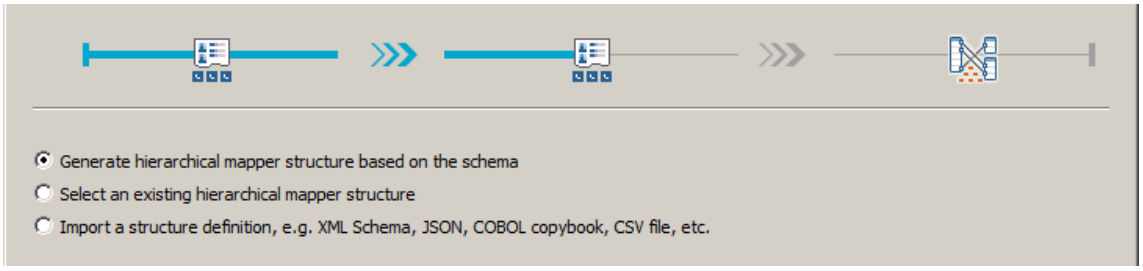
11. Now let's configure the **tHMap** Component. Double-click it to start the interactive wizard.
12. Select the **Select an existing hierarchical mapper structure** option and click **Next >**:



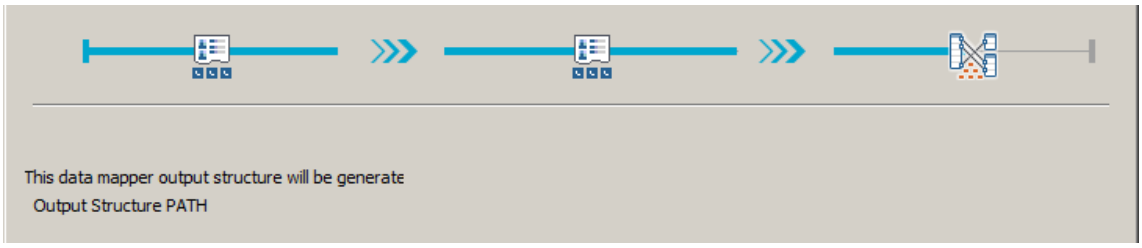
13. Select the **STRU03-RECORD** Structure previously created and click **Next >**.



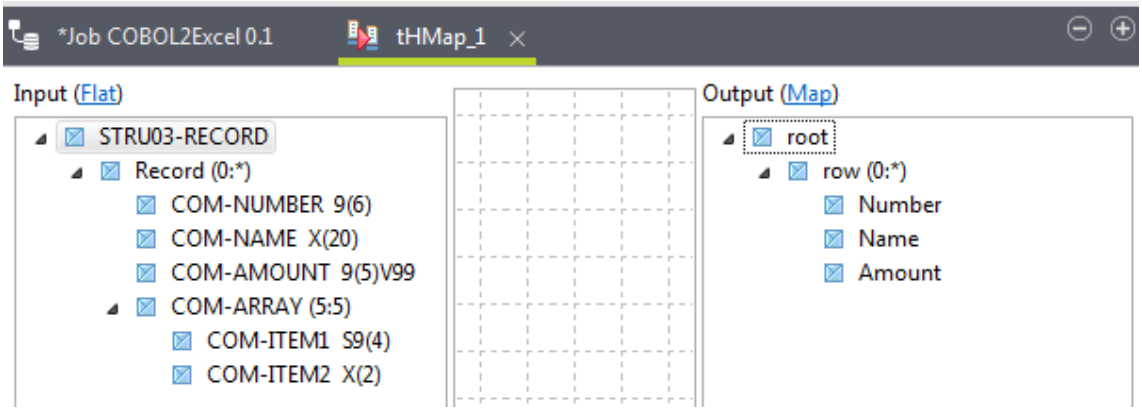
14. Click **Next >** again in the confirmation screen.
15. The wizard moves to **Step 2**, dedicated to the configuration of the output Structure. Select the **Generate hierarchical mapper structure based on the schema** option to reuse the output Schema from **tFileOutputExcel** and click **Next >**:



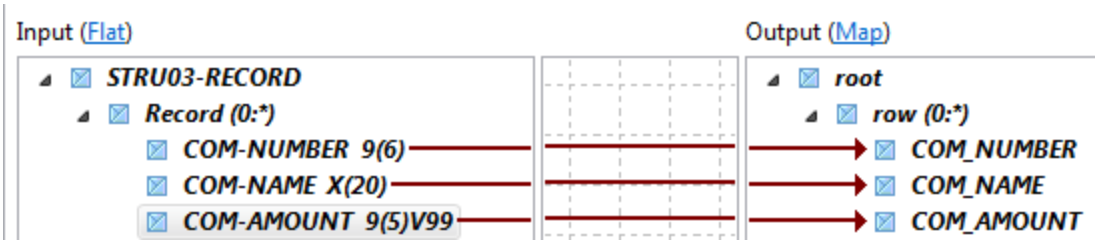
16. **Step 3** confirms the data mapper is about to be generated according to the specified settings. Click **Finish**.



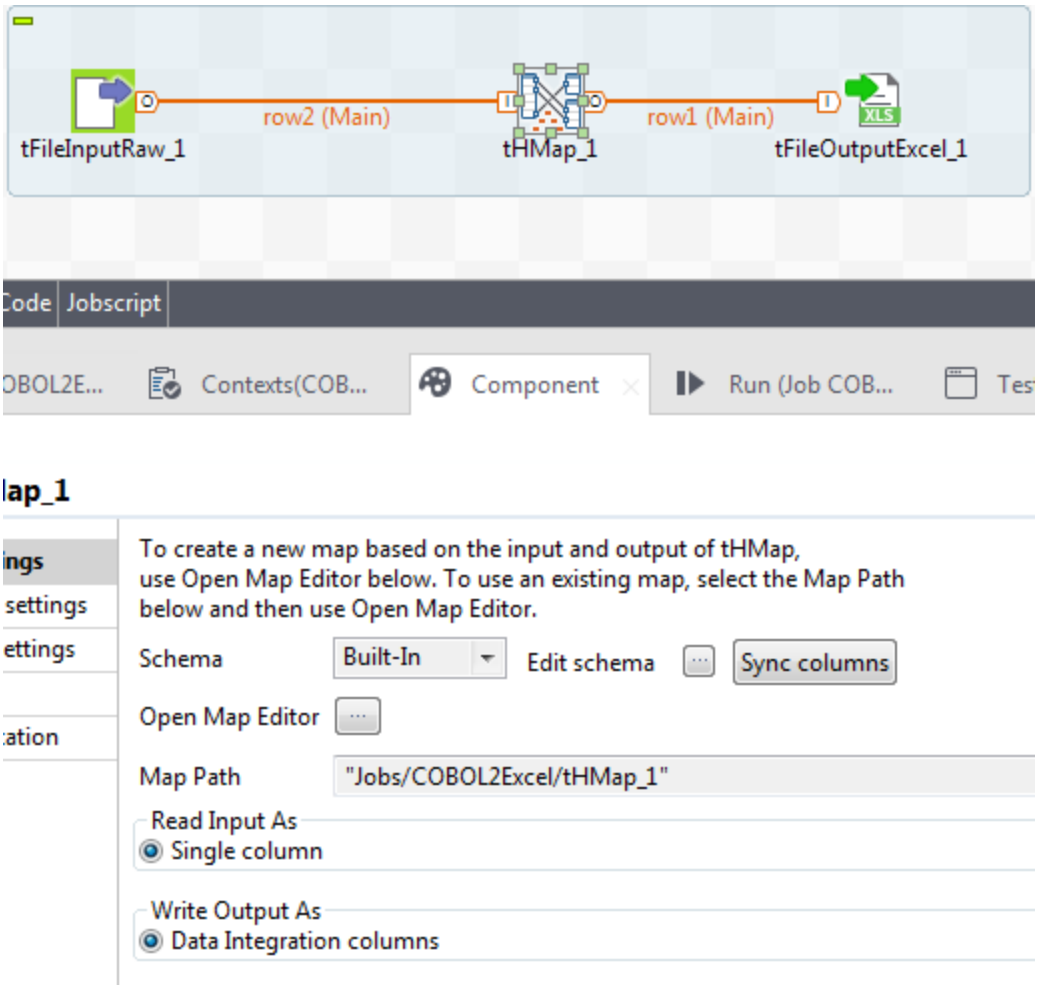
17. Once it is generated, the system will automatically open the new Map. The Input and Output areas are automatically populated with the **STRU03-RECORD** Structure and the output Structure created from the output Schema.



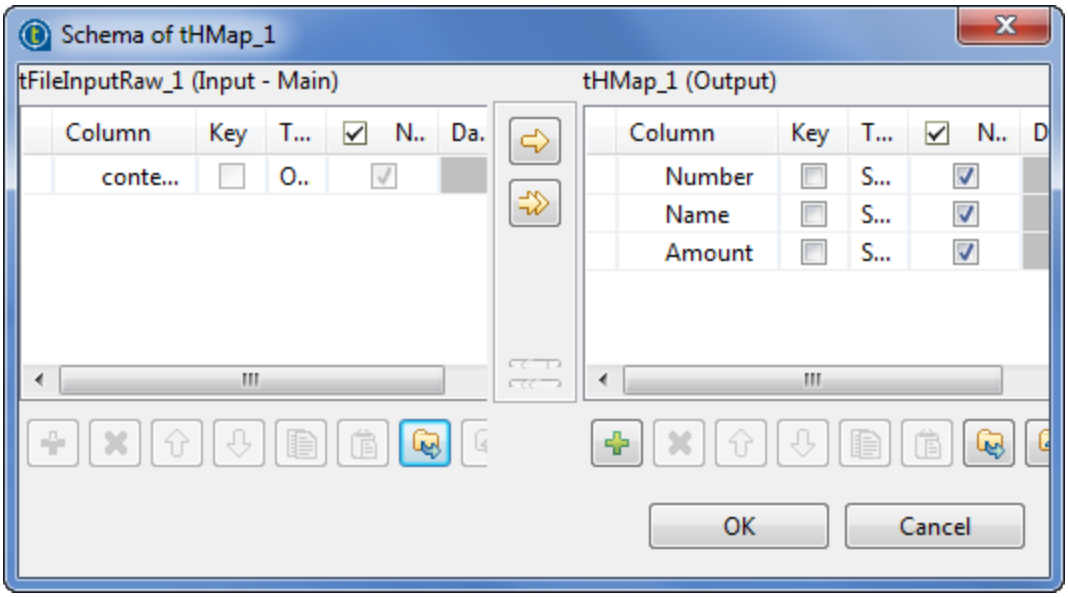
18. Drag the three non-repeating elements to the output side:
    - *COM-NUMBER* to *Number*
    - *COM-NAME* to *Name*
    - *COM-AMOUNT* to *Amount*



19. Save the Map and switch to the **Integration** perspective. Single-click the **tHMap** Component and notice that **Read Input As** and **Write Output As** have been set automatically to *Single column* and *Data Integration columns* respectively.



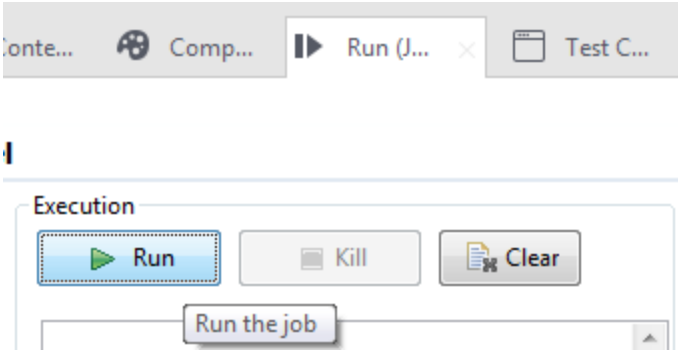
20. Click the **...** button next to **Edit schema** to display the computed schema, examine it and click **OK** to close the dialog.



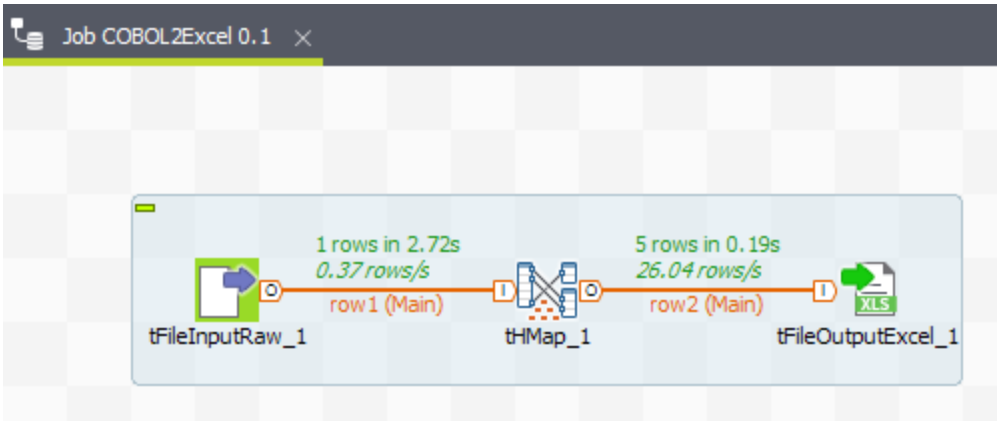
21. Now let's run the Job and display the output file to check the result.

    Save all the changes.

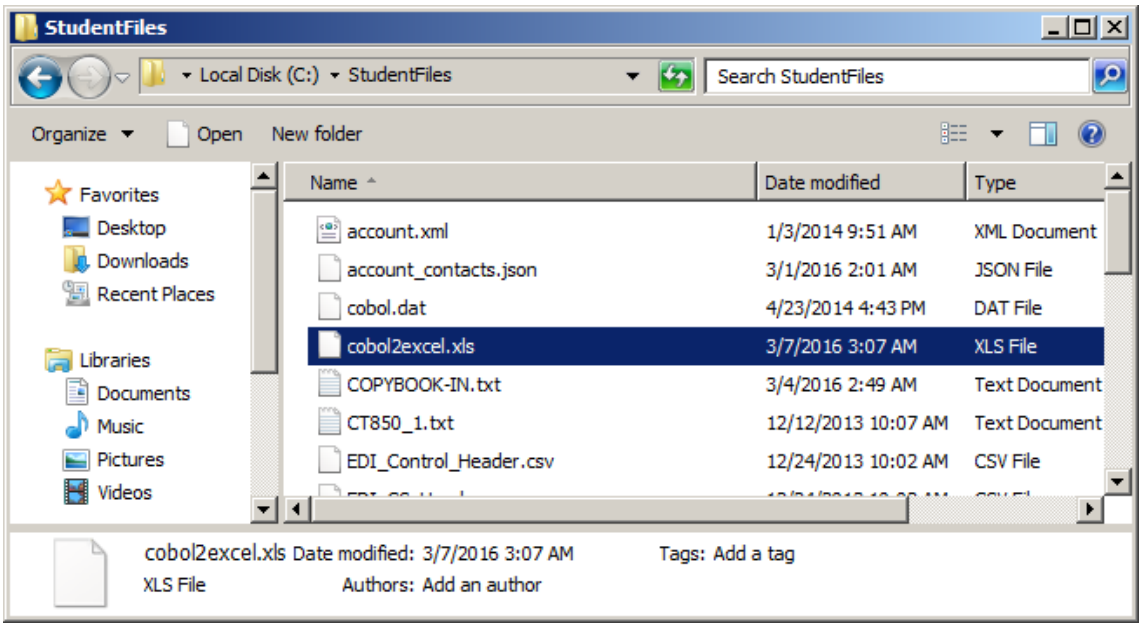
    Switch to the **Run** View and click the **Run** button:



22. The Job runs successfully if 1 input row and 5 output rows are displayed in the workspace.



23. Open the output file *cobol2excel.xls* from *C:\StudentFiles*.



24. Check the results and close it.

	A	B	C
1	000075	Earl Curzon	278.56
2	000076	John Franklin	139.28
3	000077	Francis Beaufort	92.85
4	000078	John Franklin	69.64
5	000079	Francis Beaufort	55.71
6			

Next Step

This lesson is almost over. Head to the Wrap-Up section for a summary of the concepts reviewed in this lesson.

Wrap-Up

In this lesson, you learned how to:

- Import a COBOL Structure
- Validate an imported Structure against some sample data
- Create a DI Job that relies on a tHMap Component to map a COBOL Structure to a custom output Schema stored as an Excel file
- Use the tHMap wizard to create a Map that reuses existing Structures or Schemas as input and output

Next Step

Congratulations, you successfully completed this lesson. Click the **Check your status with this unit** button below in order to save your progress. Then click **Completed. Let's continue >** on the next screen to jump to the next lesson.

LESSON 3

Flattening a Looping COBOL Structure

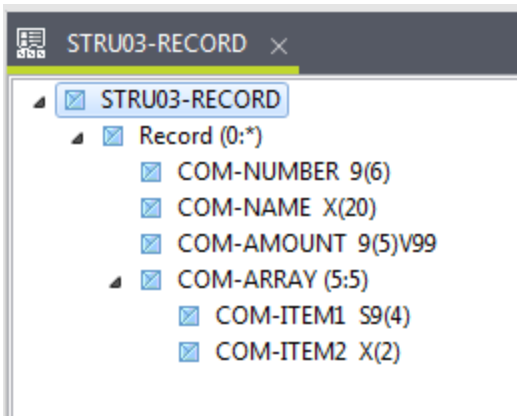
This chapter discusses the following.

Overview

Lesson Overview

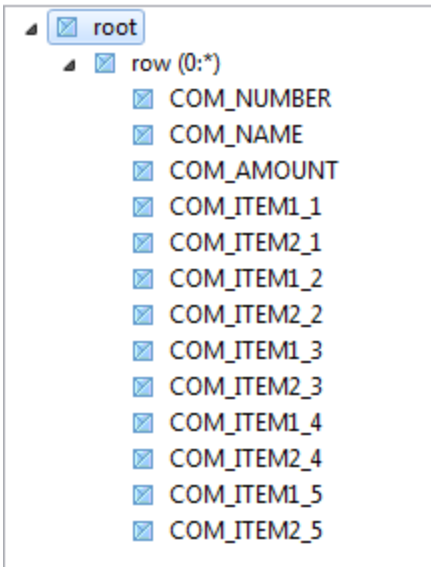
The COBOL to Excel exercise focused on the non-repeating part of a COBOL data description. This lab extends the concept by acting on the repeating portion.

The input is the same *STRU03* Structure used previously:



The objective is to convert this Structure to a Comma Separated Values (CSV) file with headers by flattening the *COM-ARRAY* loop as *COM-ITEM1-1*, *COM-ITEM1-2*, *COM-ITEM2-1*, *COM-ITEM2-2*, etc.

The output structure and the actual output must look like, respectively:




```
COM-NUMBER,COM-NAME,COM-AMOUNT,COM-ITEM1-1,COM-ITEM2-1,COM-ITEM1-2
000075,Earl Curzon          ,27856,00,V0,01,V1,02,V2,03,V3,04,V4
000076,John Franklin        ,13928,00,V0,01,V1,02,V2,03,V3,04,V4
000077,Francis Beaufort     ,9285,00,V0,01,V1,02,V2,03,V3,04,V4
000078,John Franklin        ,6964,00,V0,01,V1,02,V2,03,V3,04,V4
000079,Francis Beaufort     ,5571,00,V0,01,V1,02,V2,03,V3,04,V4
```

The challenge is to correctly map the looping structure to a non-looping structure.

Objectives

After completing this lesson, you will be able to:

- Create a complex mapping between repeating and non-repeating elements, to convert a loop into a flat sequence
- Use a tHMap and a tMap to perform the transformations from a DI Job, and convert some structured COBOL data to a flat CSV file

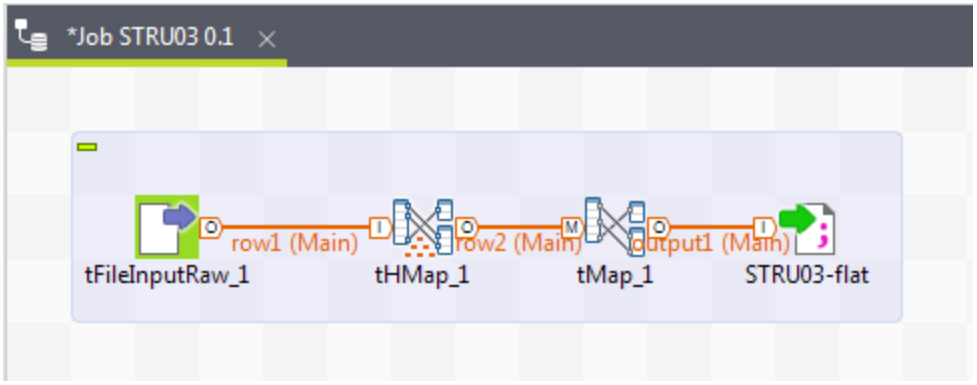
Next Step

First, let's create the DI Job responsible for the transformation.

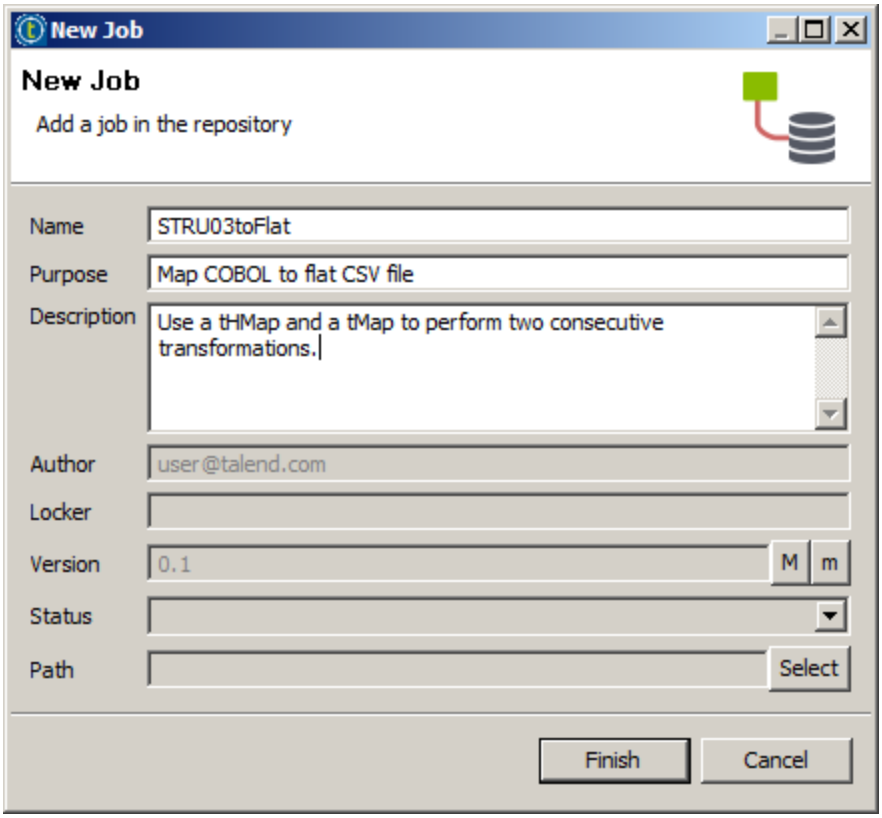
Creating the Job

Let's create the Job responsible for transforming the COBOL data into a CSV file.

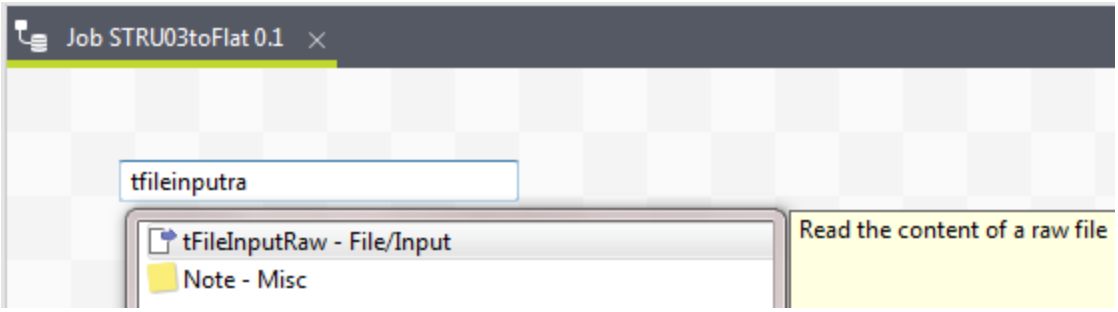
The final Job will look like:



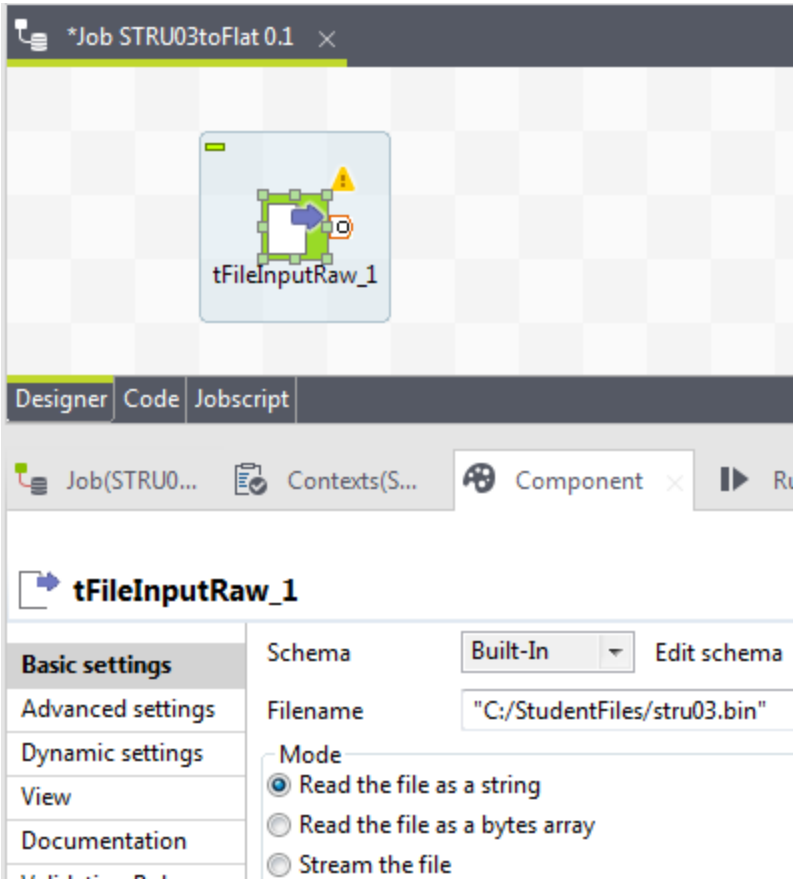
1. First, switch to the **Integration** perspective. Right-click **Job Designs** and select **Create Standard Job**. Configure the new Job as illustrated below then click **Finish**.



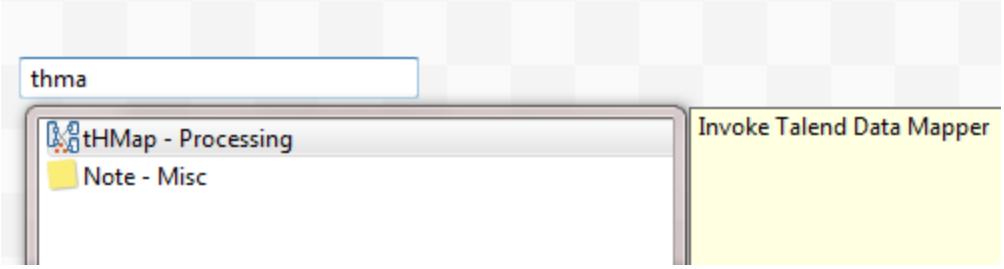
2. Place a **tFileInputRaw** on the Design area.



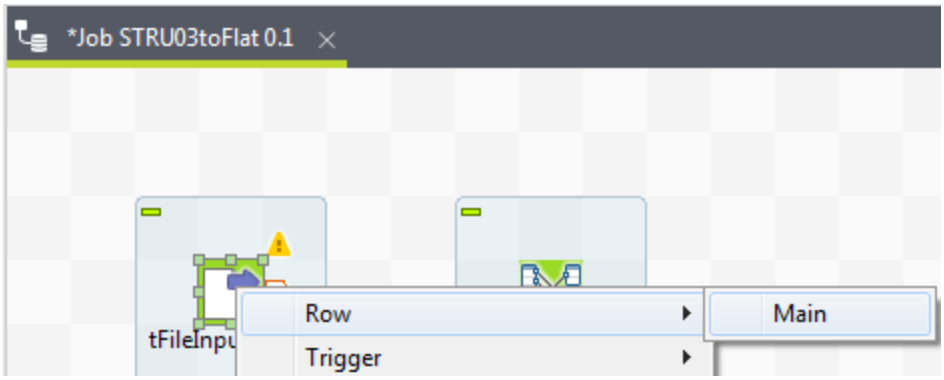
3. Double-click the new component and set the **Mode** to *Read the file as a string*. Then click the **...** button next to the **Filename** field and select *stru03.bin* stored in *C:\StudentFiles.*



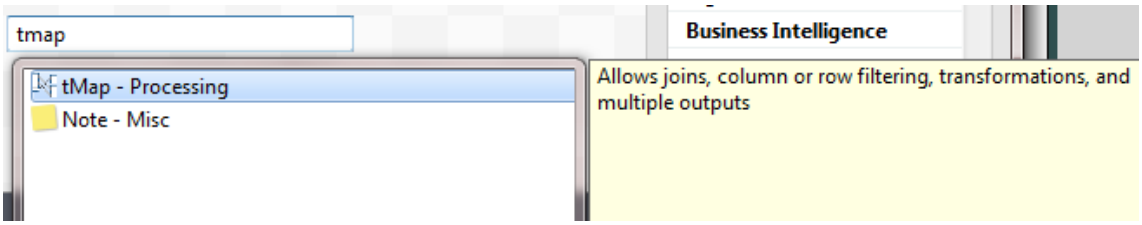
4. Place a **tHMap** component to the right of the input component.



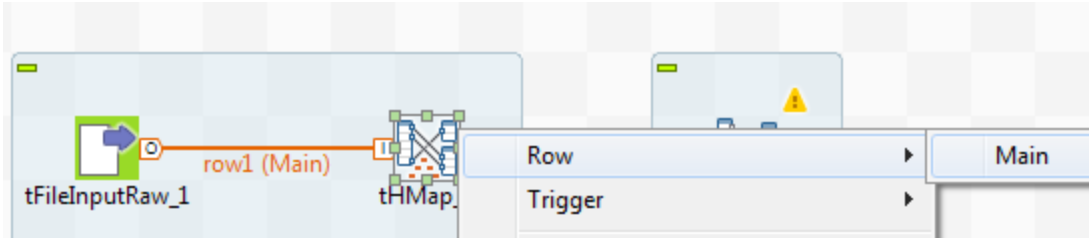
5. Right-click **tFileInputRaw**, select **Row > Main** and connect it to the **tHMap** component.



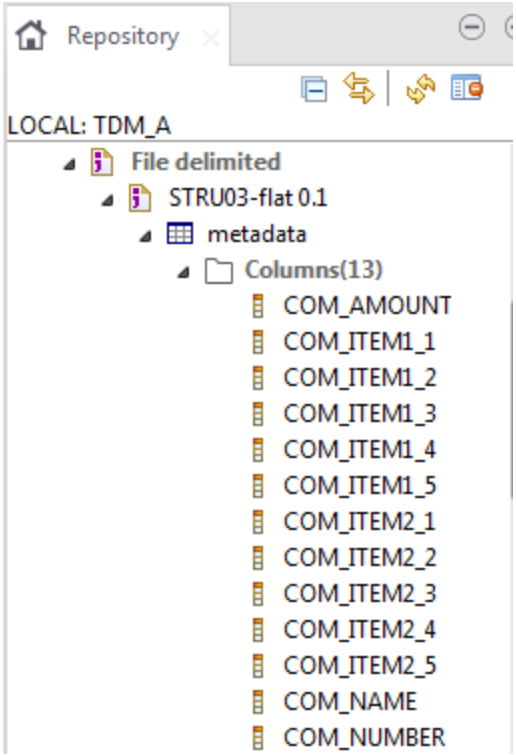
6. Place a **tMap** component to the right of the **tHMap** component:

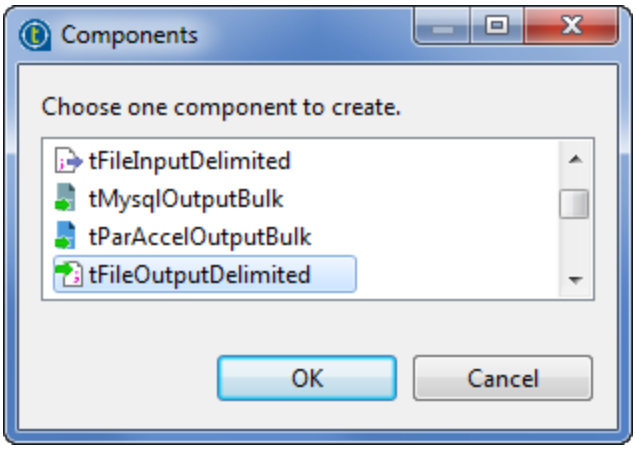


7. Right-click **tHMap**, select **Row > Main** connect **tMap** with **tHMap**.



8. Finally, the output Component will be created from some existing metadata located in **Repository > Metadata > File delimited > STRU03-flat**. Drag this item from the Repository to the Design area, to the right of **tMap**, and select a **tFileOutputDelimited** when requested.

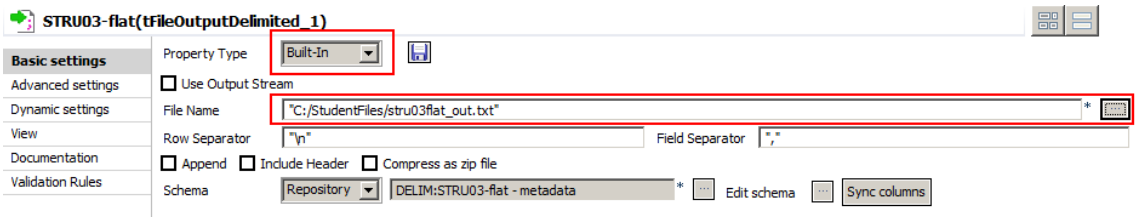




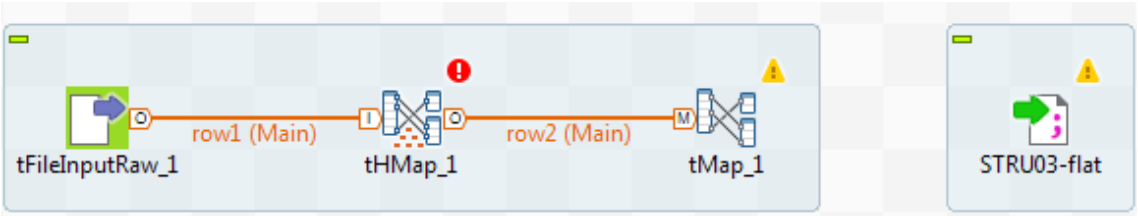
9. Double-click the output Component to display its **Component** View.

   Set **Property Type** to *Built-In* instead of *Repository* to be able to modify the output filename.

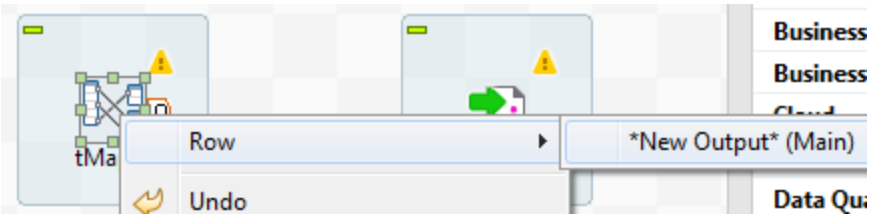
   Click the **...** button next to the **File Name** field, browse to *C:\StudentFiles* and enter *stru03flat_out.txt* for the output file-name.



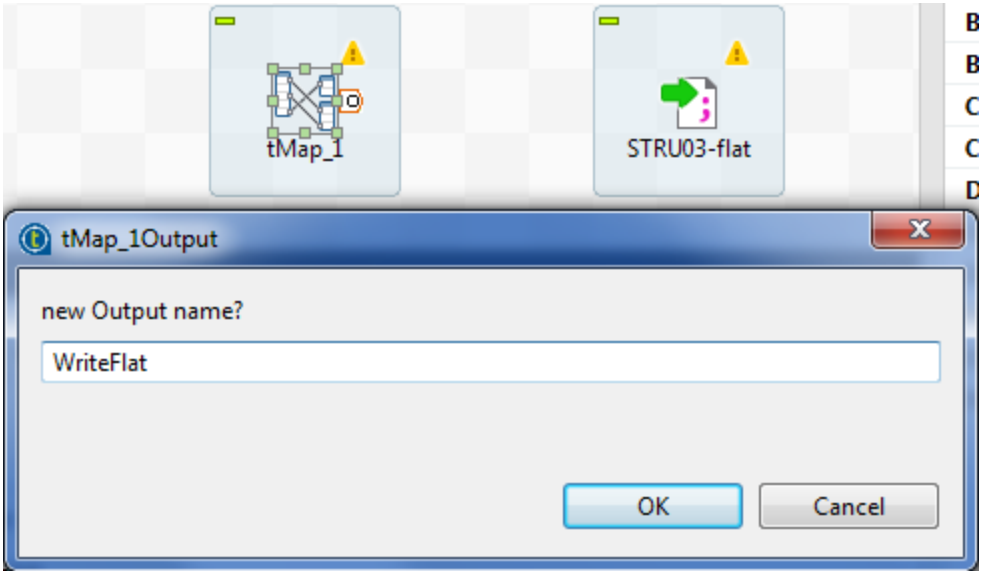
10. The Job should look like this:



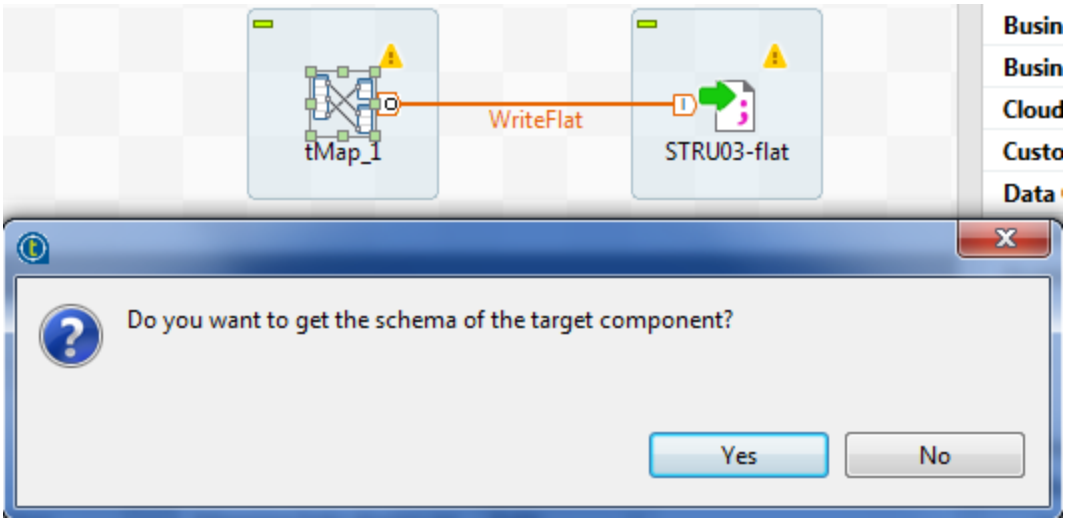
11. Connect **tMap** with **STRU03-flat** after right-clicking **tMap** and selecting **Row > *New Output* Main**. This option creates a new output in **tMap** automatically.



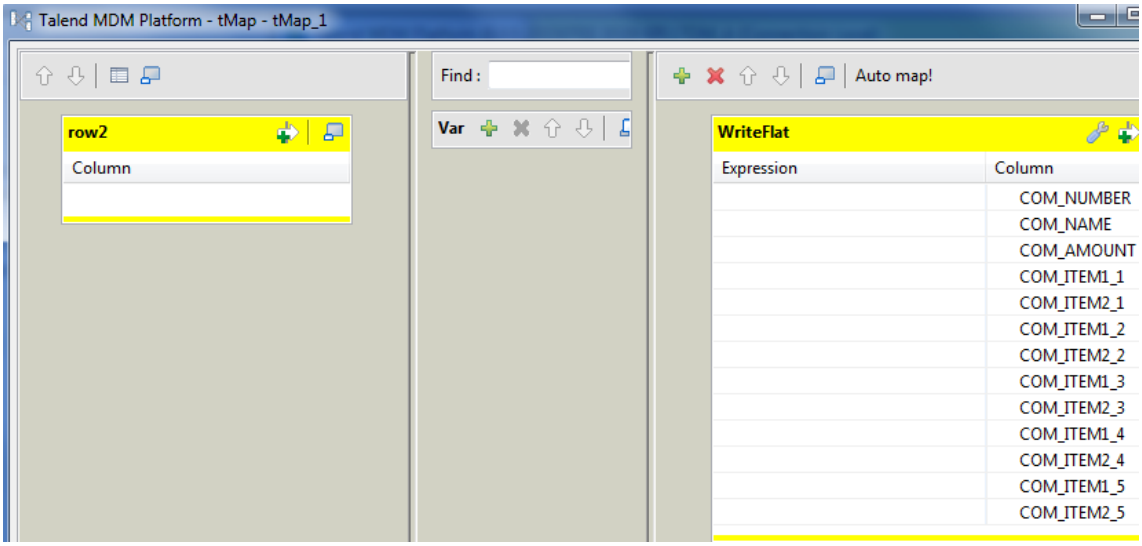
12. Enter **WriteFlat** when requested to provide a name for the new output and click **OK**.



13. Click **Yes** when asked **Do you want to get the schema from the target component?**.

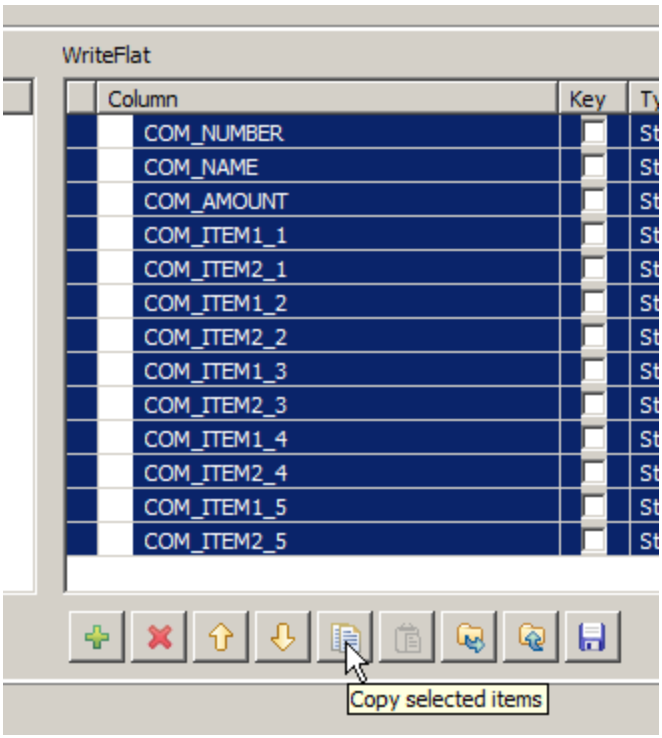


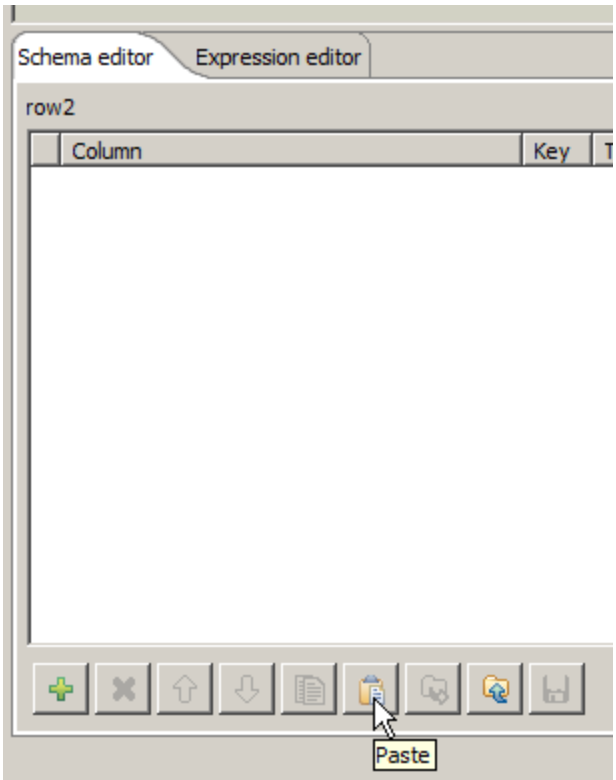
14. Double-click **tMap** to display the output schema for the **WriteFlat** output.

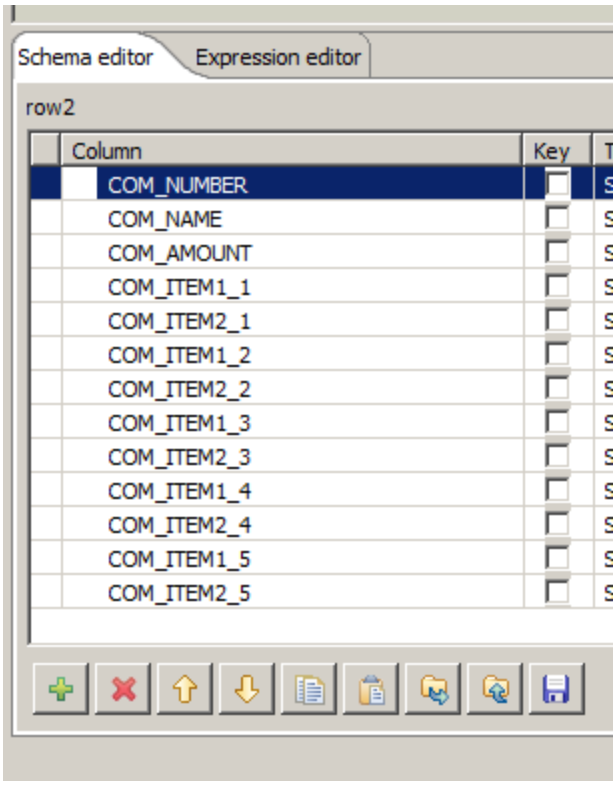


15. Note that the input on the left is not defined yet. As the output Structure from **tHMap** will look exactly the same, it can be copy/pasted from the output.

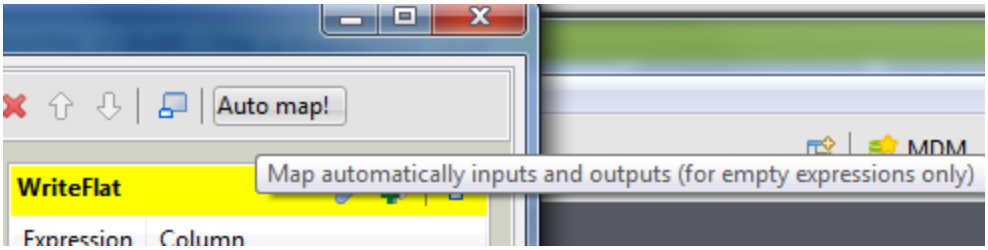
    To do so, select all columns from the **WriteFlat** output table at the bottom right of the window, click the **Copy select items** button in the output toolbar and finally click the **Paste** button below the input list.


WriteFlat
Column
Key
COM_NUMBER
COM_NAME
COM_AMOUNT
COM_ITEM1_1
COM_ITEM2_1
COM_ITEM1_2
COM_ITEM2_2
COM_ITEM1_3
COM_ITEM2_3
COM_ITEM1_4
COM_ITEM2_4
COM_ITEM1_5
COM_ITEM2_5
Copy selected items

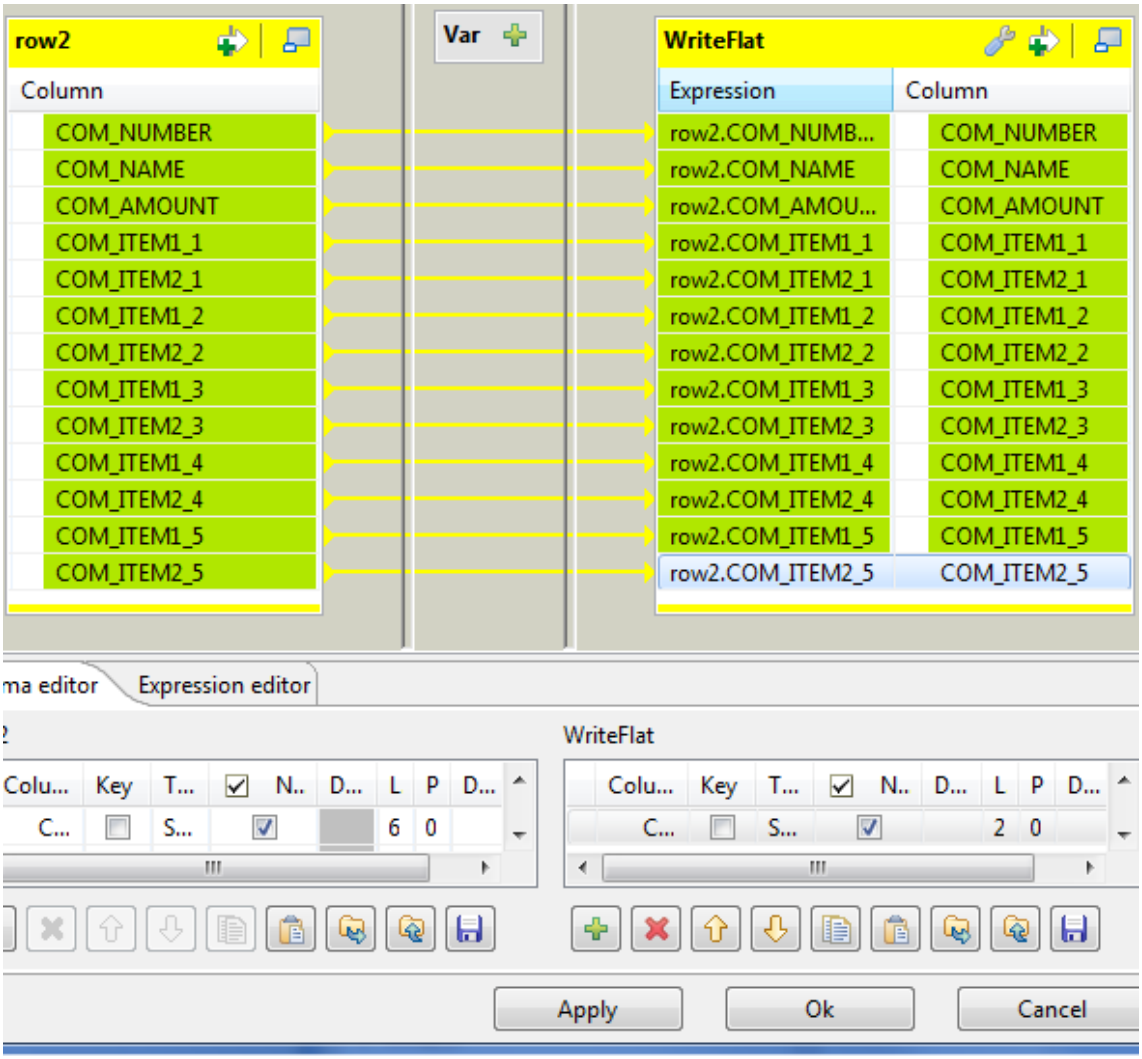

Schema editor
Expression editor
row2
Column
Key
Paste



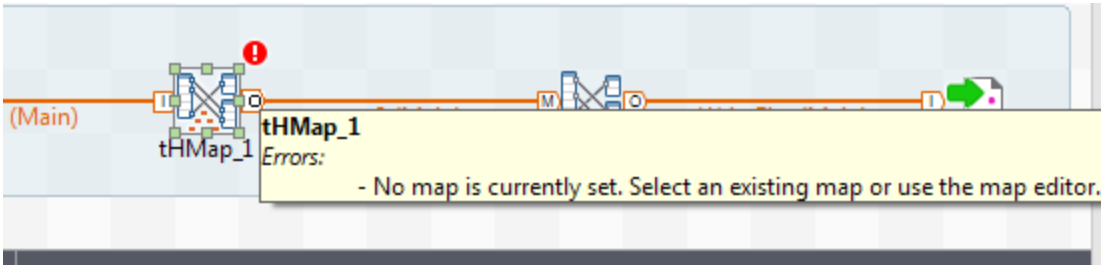
16. Now that the input and output have the needed columns, let's do the mapping. As all elements from the input should be mapped to the output, leverage the Automap functionality by clicking the **Automap** button on top of the output table:



17. Check that all elements are mapped as illustrated below and click **OK** to close the mapping editor.



18. Finally, notice the error message on top of **tHMap**. It indicates a Map is still missing.



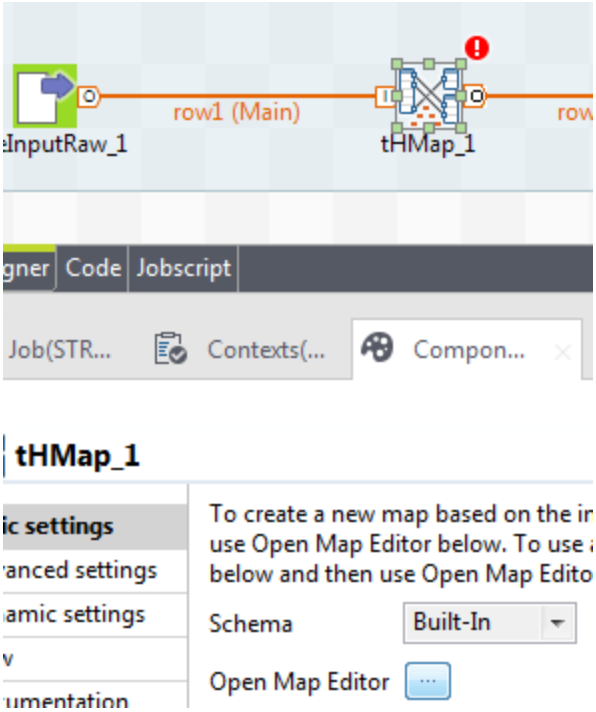
Next Step

Let's fix the error message by starting the Map editor for tHMap.

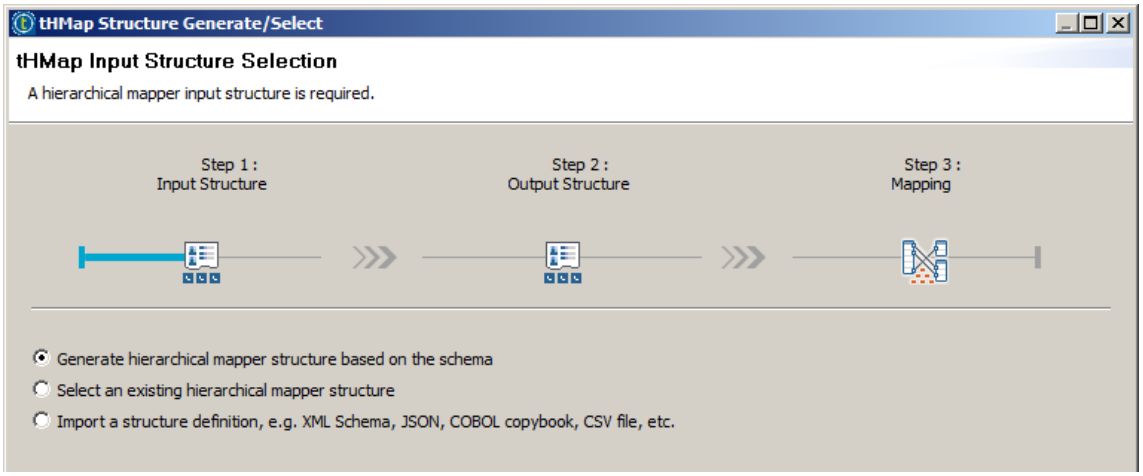
Running the Map Editor

The Map Editor Wizard will support you in creating a Map with the right input and output Structures, matched later on in the Data Mapper.

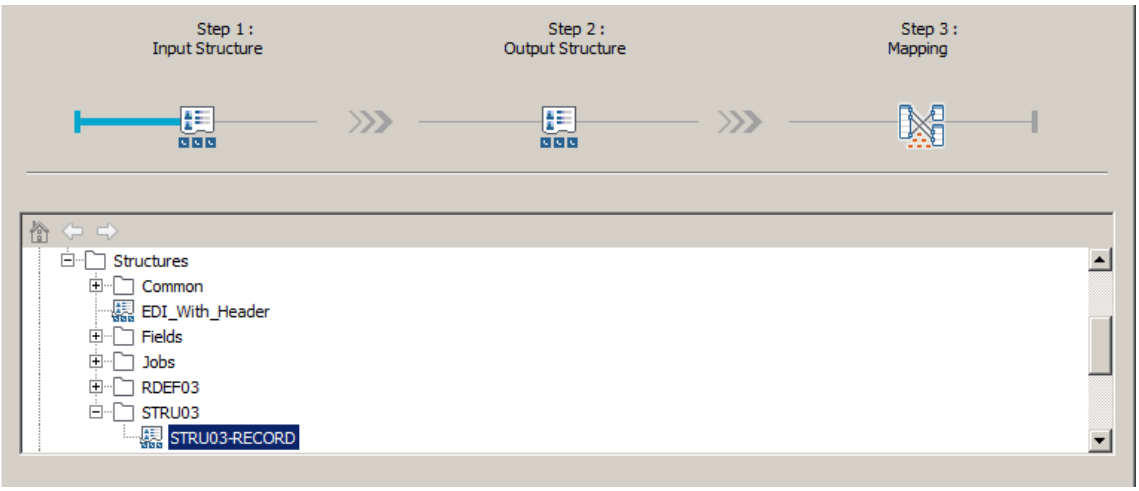
1. Click the **tHMap** component to display its **Component** View, then click the **...** button next to **Open Map Editor** to start the wizard.



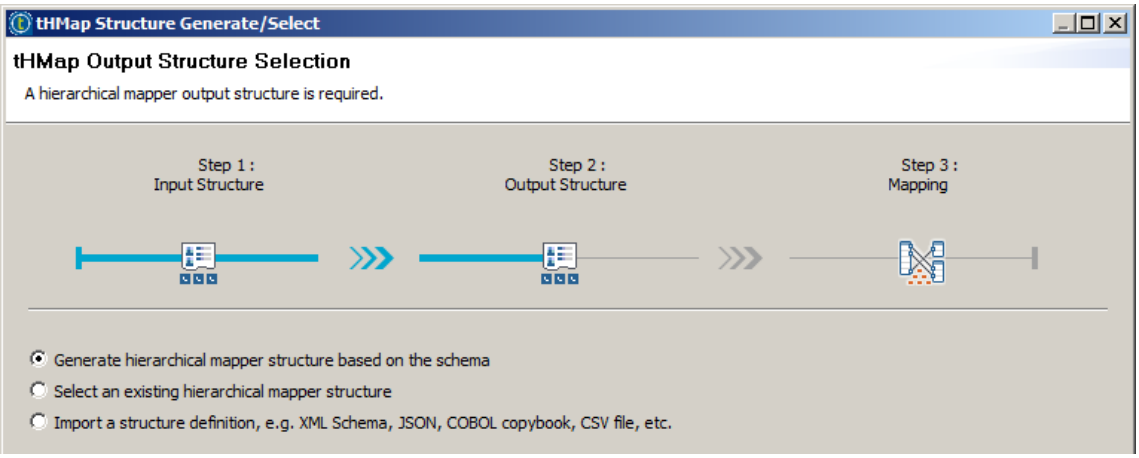
2. Select the **Select an existing hierarchical mapper structure** option in **Step 1** then click **Next >**.



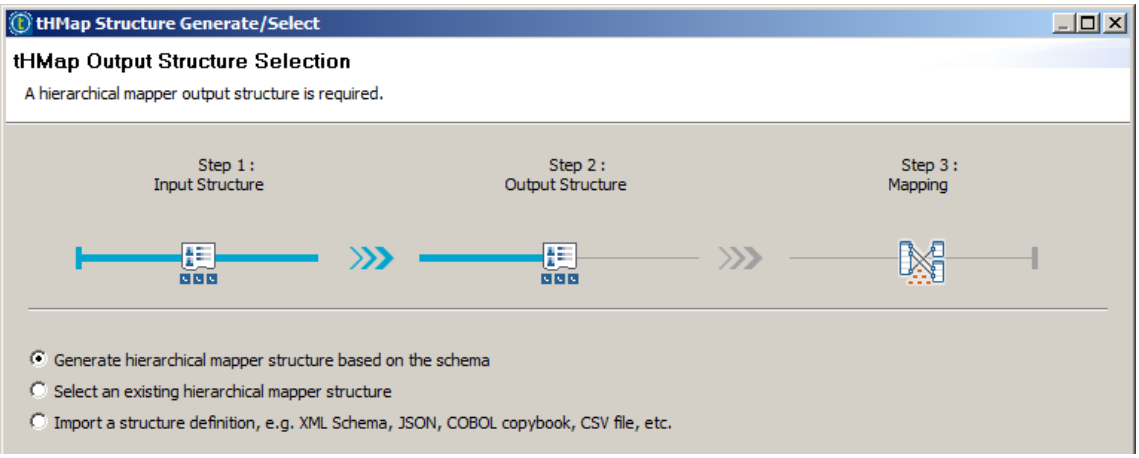
3. Select the **STRU03-RECORD** Structure for the input and click **Next >**.



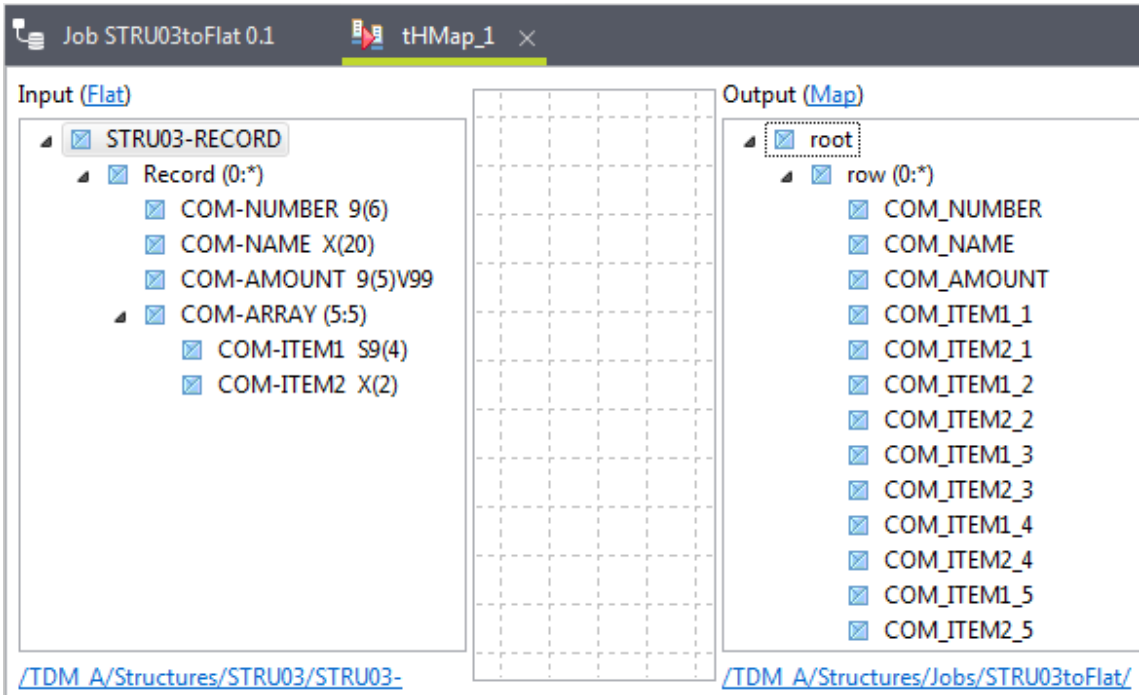
4. Click **Next >** on the confirmation screen again to move to **Step 2**.
5. Keep the default **Generate hierarchical mapper structure based on the schema** option in **Step 2** to reuse the schema defined in the previous section in **tMap**. Then click **Next**.



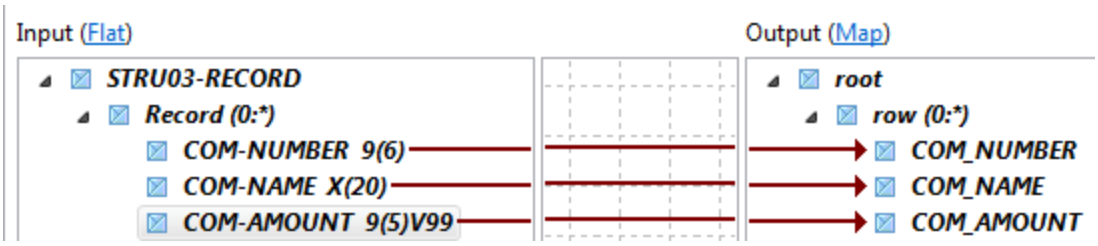
6. Finally, click **Finish** on the last confirmation screen to actually create the Map.



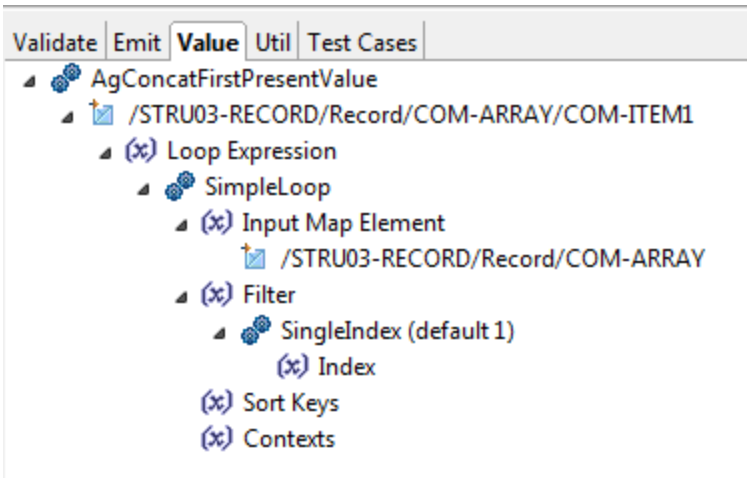
7. Now let's map the elements and customize the mapping. The Data Mapper starts automatically and displays the input and output Structures set with the wizard.



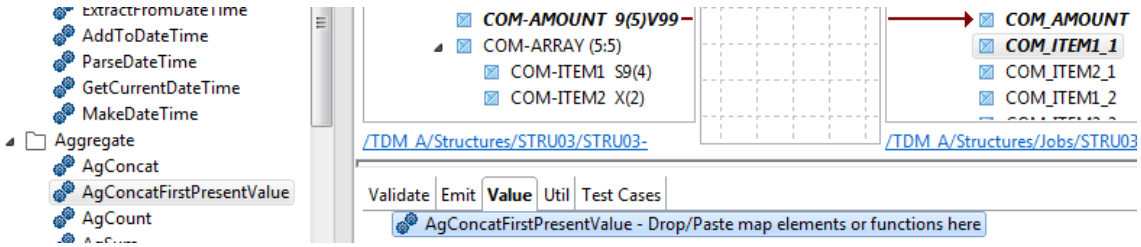
8. First, map the first three non-repeating elements.



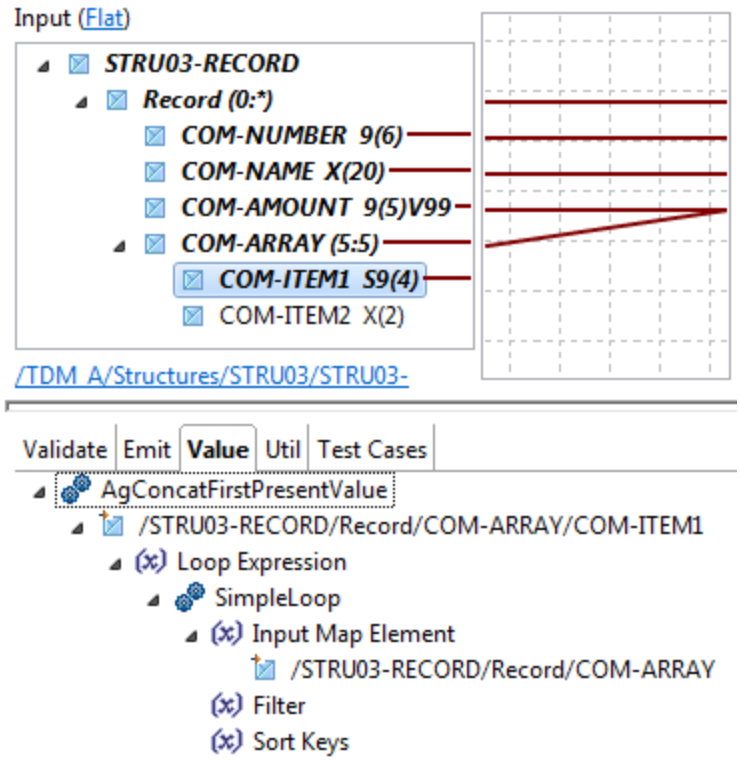
9. The remaining elements will be mapped from the elements under the *COM-ARRAY* Structure which is a looping structure. A direct mapping cannot be performed as the output Structure has been flattened. Thus you need to use an **AgConcat** or **AgConcatFirstPresentValue** Function and then apply a **SingleIndex** filter. For example, here is what the *COM-ITEM1* element looks like:



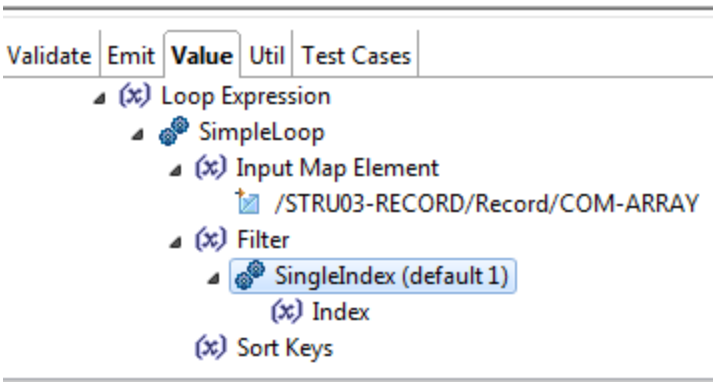
10. Drag a **Functions > Aggregate > AgConcatFirstPresentValue** to the **Value** tab of the first unmapped output element *COM_ITEM1_1*:



11. Drag *COM-ITEM1* from the **Input** area and drop it on the Function:



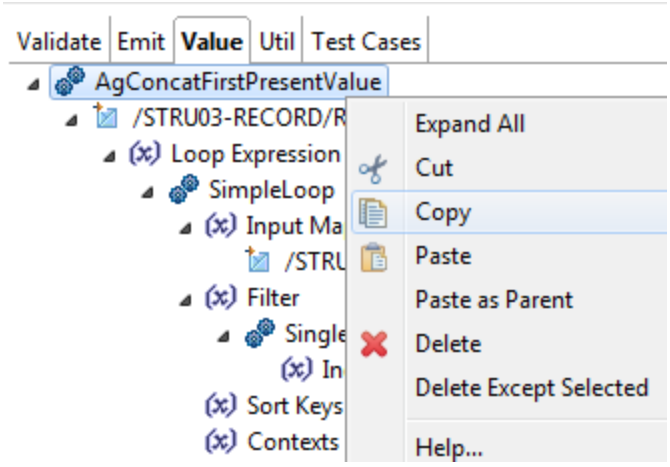
12. Drag a **Functions > Loop > Filters > SingleIndex** and drop it on the **Filter** branch:



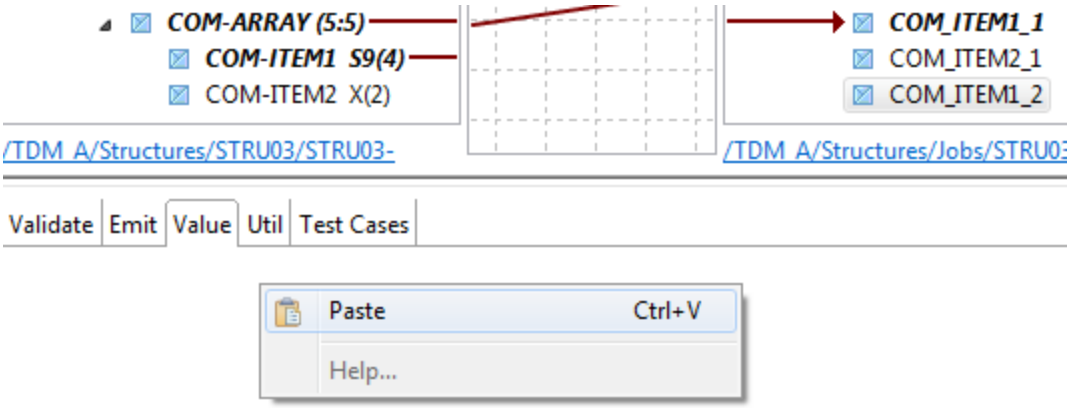
Note that **SingleIndex** is set to *1* by default, which is valid for the first index of the *COM_ITEM1_**1*** and *COM_ITEM2_**1*** elements. For the next elements, it must be set to the right index value: *2* for *COM_ITEM1_**2*** and *COM_ITEM2_**2***, and so on until *5* for *COM_ITEM1_**5*** and *COM_ITEM2_**5***.

13. Now let's copy this Function and apply it to the rest of the output elements whose index is 1, i.e. *COM_ITEM**1**_2*, *COM_ITEM**1**_3*, *COM_ITEM**1**_4*, and *COM_ITEM**1**_5*.

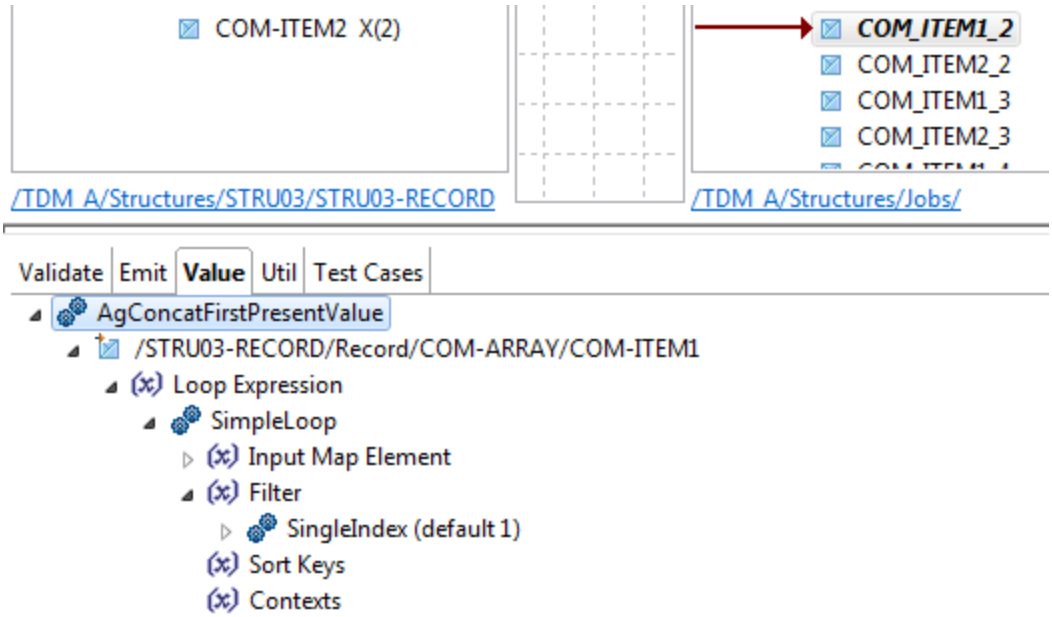
    Right-click *AgConcatFirstPresentValue* and select **Copy**:



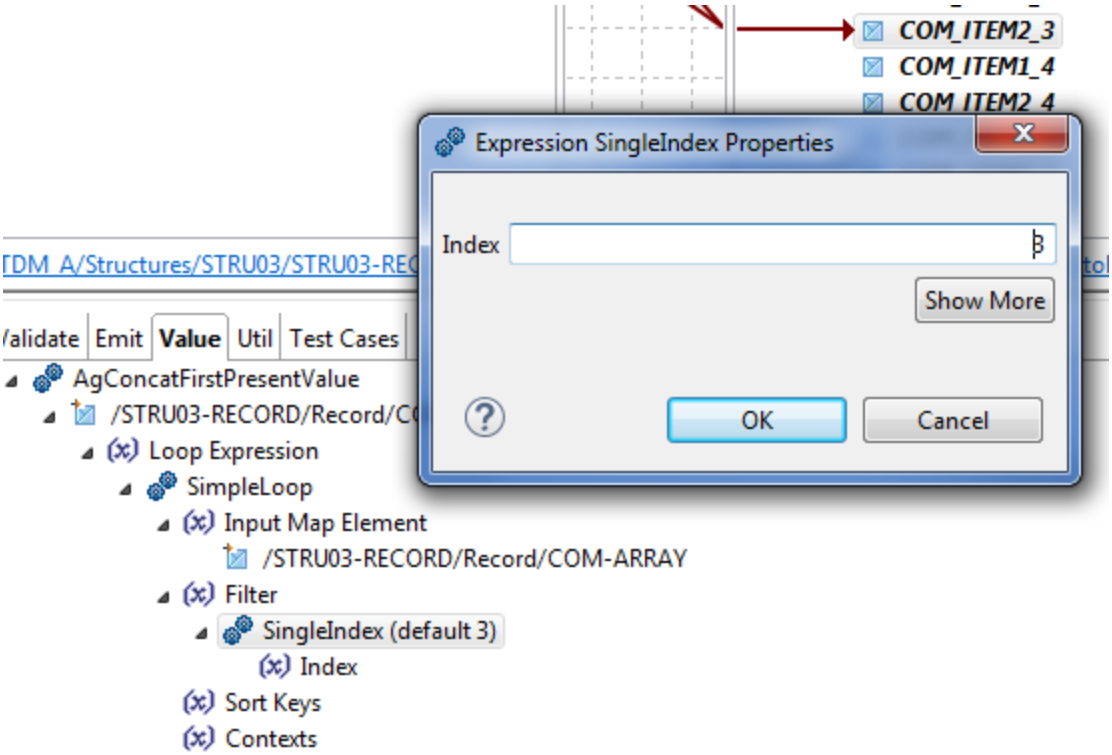
14. Select *COM_ITEM1_2* in the **Output** area to display its **Value** tab. Right-click the empty area and select **Paste**.



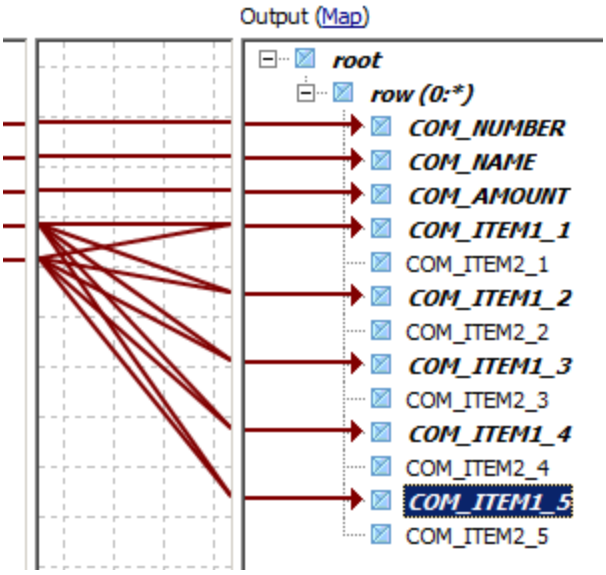
15. Repeat the previous step for all the elements whose name start with *COM_ITEM1_*:



16. Double-click the **SingleIndex** Function of all *COM_ITEM1_* elements and enter the correct index value. Here is an example for the *COM_ITEM2_**3*** element.

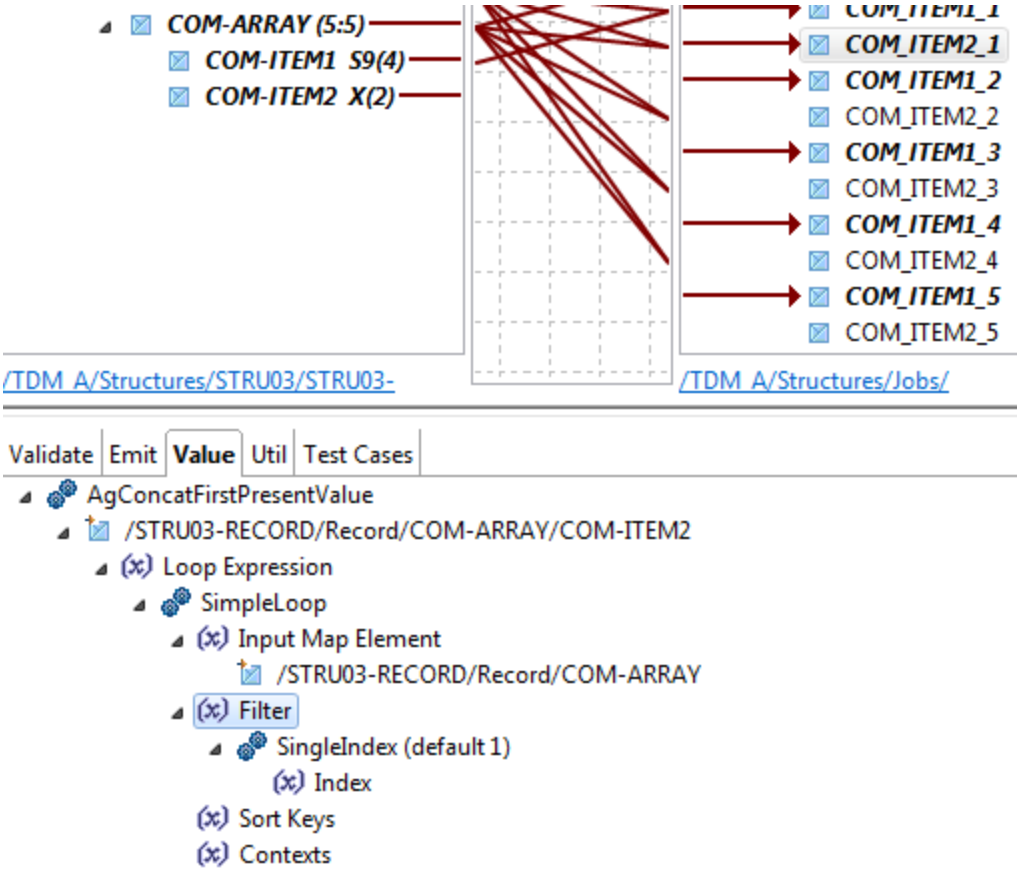


17. When done, all *COM_ITEMS1_* elements should be mapped as illustrated below.

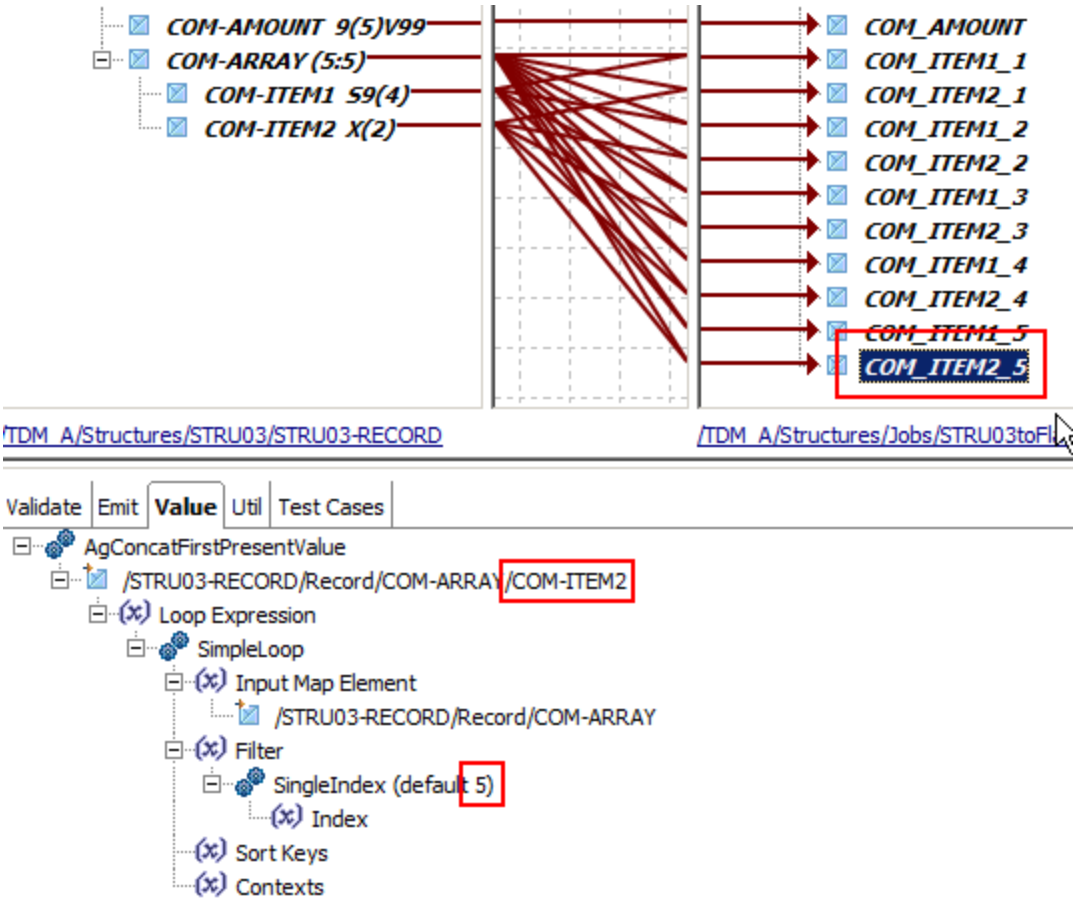


18. Now repeat the previous steps for elements whose names start with *COM_ITEM2_*, for example *COM_ITEM2_1*.
    1. Select *COM_ITEM2_1* to display its **Value** tab.
    2. Drag a **Functions > Aggregate > AgConcatFirstPresentValue** to the **Value** tab.

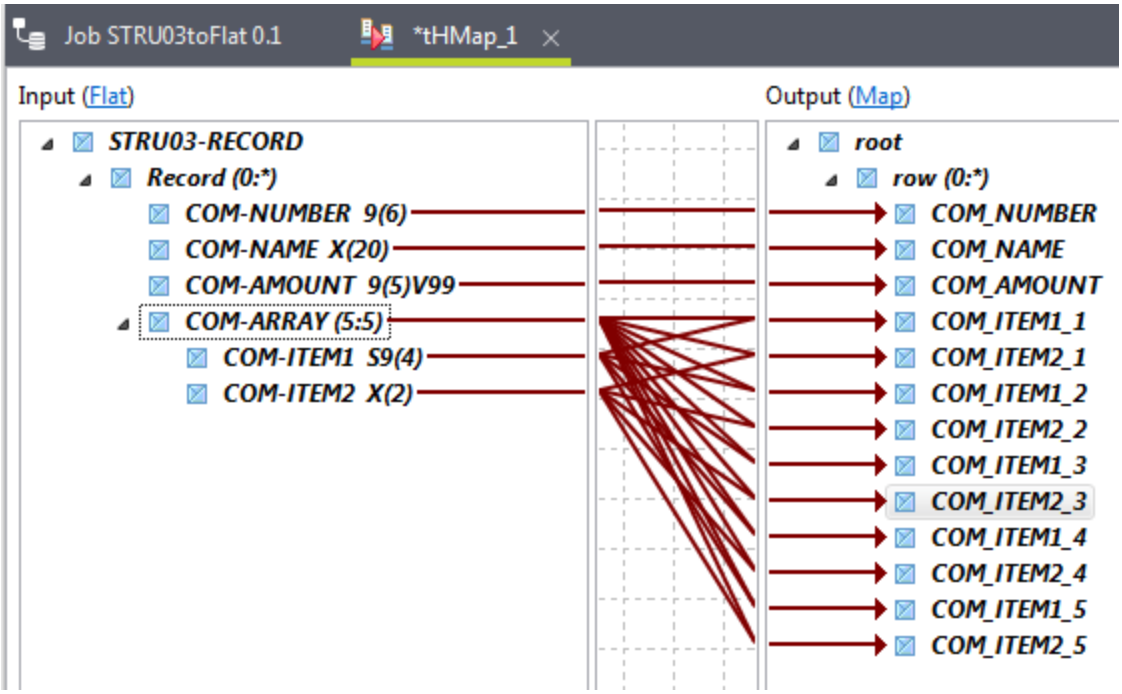
3. Drag *COM-ITEM2* from the **Input** area and drop it on the Function.
4. Drag a **Functions > Loop > Filters > SingleIndex** and drop it on the **Filter** branch.
5. Copy/paste the *AgConcatFirstPresentValue* Function from *COM_ITEM2_1* to all the other *COM_ITEM2_* elements.
6. Double-click the **SingleIndex** Function of all *COM_ITEM2_* elements and enter the correct index value. Here is an example for the *COM_ITEM2_**3*** element.



19. Check that all the *COM_ITEM* output elements use the correct settings, as highlighted below.


COM-AMOUNT 9(5)V99
COM-ARRAY (5:5)
COM-ITEM1 S9(4)
COM-ITEM2 X(2)
COM_AMOUNT
COM_ITEM1_1
COM_ITEM2_1
COM_ITEM1_2
COM_ITEM2_2
COM_ITEM1_3
COM_ITEM2_3
COM_ITEM1_4
COM_ITEM2_4
COM_ITEM1_5
COM_ITEM2_5
/TDM_A/Structures/STRU03/STRU03-RECORD
/TDM_A/Structures/Jobs/STRU03toFl
Validate
Emit
Value
Util
Test Cases
AgConcatFirstPresentValue
/STRU03-RECORD/Record/COM-ARRAY/COM-ITEM2
Loop Expression
SimpleLoop
Input Map Element
/STRU03-RECORD/Record/COM-ARRAY
Filter
SingleIndex (default 5)
Index
Sort Keys
Contexts

20. Once all the elements are mapped with a correct index value, the Map should look like this.



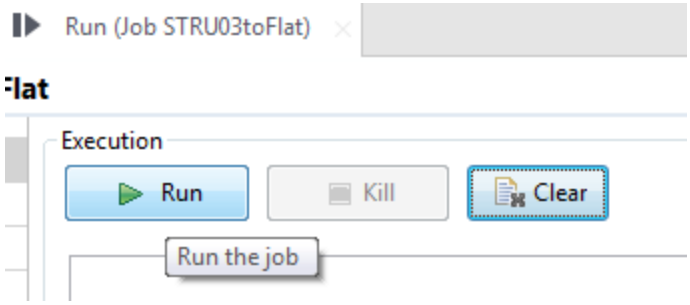
21. Save the changes to the Map and switch back to the **Integration** perspective and the *STRU03toFlat* Job.

Next Step

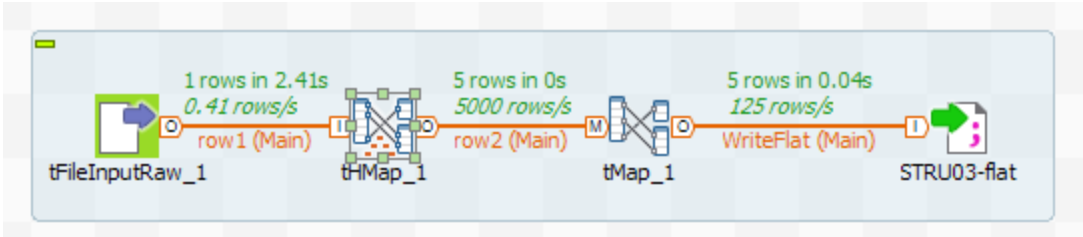
All the settings are properly configured. Let's run the Job and check the output.

Running the Job

1. Go to the **Run** tab and click **Run** to execute the transformation.



2. The Design area and the Output Console should look like:




```
Starting job STRU03toFlat at 03:37 08/03/2016.

[statistics] connecting to socket on port 3942
[statistics] connected
0      INFO [main] oaklandsw.transform.infrequent - Loaded project
TDM_A from development workspace/installation at:
file:/C:/Talend/6.1.1/studio/workspace/TDM_A
[INFO ]: com.oaklandsw.transform.infrequent - Loaded project TDM_A
from development workspace/installation at:
file:/C:/Talend/6.1.1/studio/workspace/TDM_A
118    INFO [main] oaklandsw.transform.infrequent - Map:
/TDM_A/Maps/Jobs/STRU03toFlat/tHMap_1.xml started - time: 76
[INFO ]: com.oaklandsw.transform.infrequent - Map:
/TDM_A/Maps/Jobs/STRU03toFlat/tHMap_1.xml started - time: 76
187    INFO [main] oaklandsw.transform.infrequent - Using X/Query
engine: Saxon 9.1.0.2osJ from Saxonica (Saxon B - Free version)
[INFO ]: com.oaklandsw.transform.infrequent - Using X/Query engine:
Saxon 9.1.0.2osJ from Saxonica (Saxon B - Free version)
[statistics] disconnected
Job STRU03toFlat ended at 03:37 08/03/2016. [exit code=0]
```

3. Finally, open the output file *stru03flat_out.txt* in *C:\StudentFiles* and check the results match the expected output.

```
stru03flat_out.txt
1  000075,Earl Curzon,278.56,0,V0,1,V1,2,V2,3,V3,4,V4
2  000076,John Franklin,139.28,0,V0,1,V1,2,V2,3,V3,4,V4
3  000077,Francis Beaufort,92.85,0,V0,1,V1,2,V2,3,V3,4,V4
4  000078,John Franklin,69.64,0,V0,1,V1,2,V2,3,V3,4,V4
5  000079,Francis Beaufort,55.71,0,V0,1,V1,2,V2,3,V3,4,V4
6
```

Next Step

This lesson is almost over. Head to the Wrap-Up section for a summary of the concepts reviewed in this lesson.

Wrap-Up

In this lesson, you learned how to:

- Create a complex mapping between repeating and non-repeating elements, to convert a loop into a flat sequence
- Use a tHMap and a tMap to perform the transformations from a DI Job, and convert some structured COBOL data to a flat CSV file

Next Step

Congratulations, you successfully completed this lesson. Click the **Check your status with this unit** button below in order to save your progress. Then click **Completed. Let's continue >** on the next screen to jump to the next lesson.

This page intentionally left blank to ensure new chapters start on right (odd number) pages.

LESSON 4

Mapping COBOL to XML

This chapter discusses the following.

Overview

Lesson Overview

Imagine you need to map some complex COBOL data to an XML output. This lab will guide you from the import of the COBOL records to the definition of the mapping.

Objectives

After completing this lesson, you will be able to:

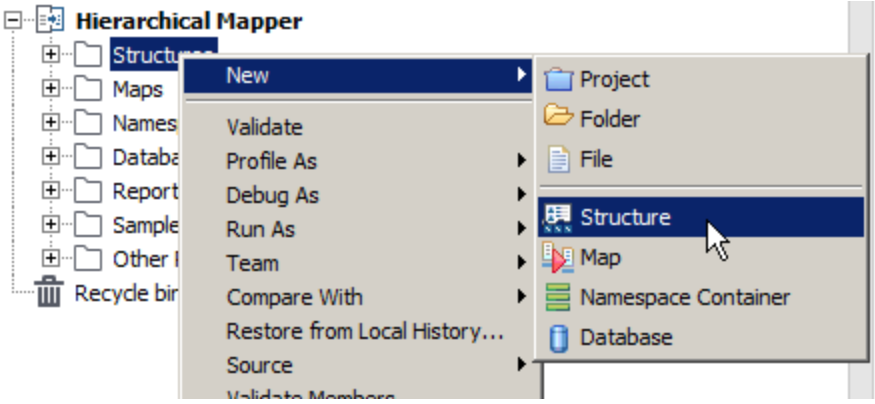
- Import multiple records from a COBOL copybook and create one Structure for each
- Merge several Structures into one outer Structure by defining inheritance manually
- Change the output representation of a Map from Flat to XML

Next Step

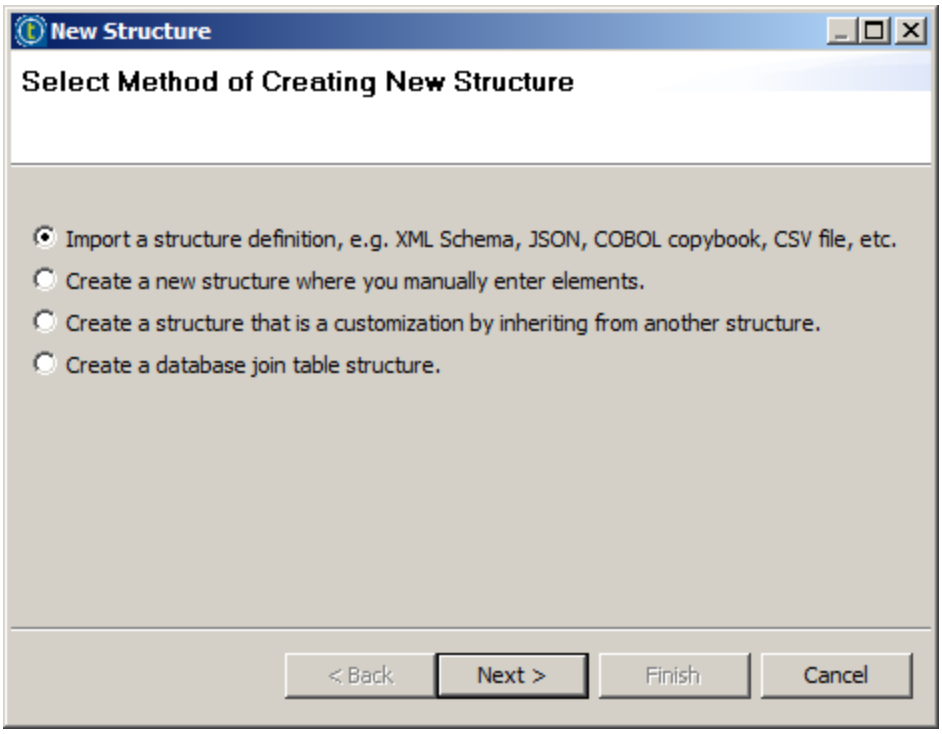
First, let's import the records available in the COBOL copybook and create one Structure for each record.

Importing the COBOL Structures

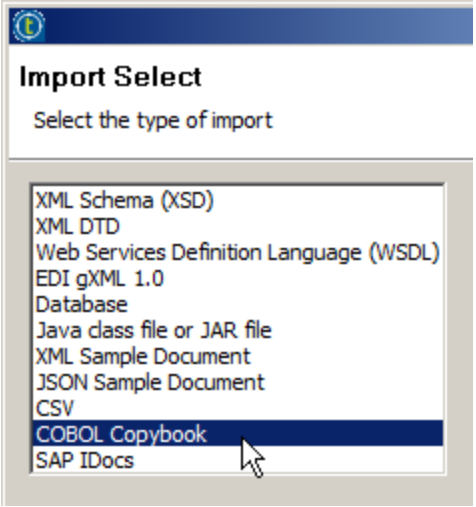
1. Switch to the **Mapping** Perspective.
2. Right-click **Hierarchical Mapper > Structures** and select **New > Structure**.



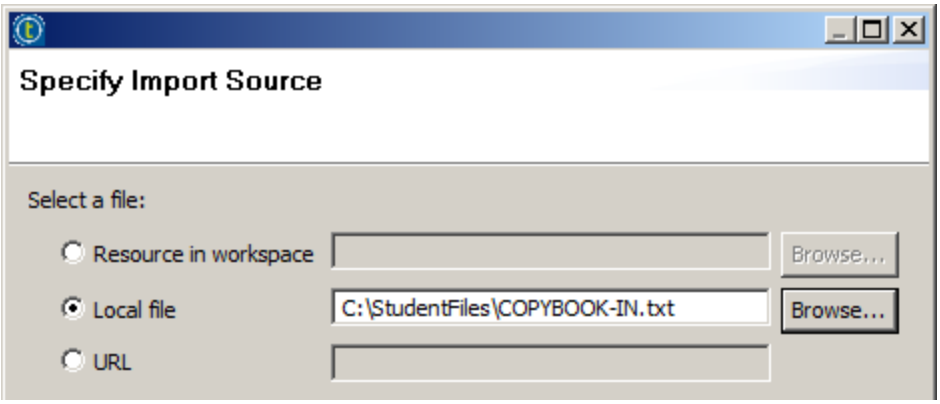
3. Keep the default **Import a structure definition** option and click **Next >**.



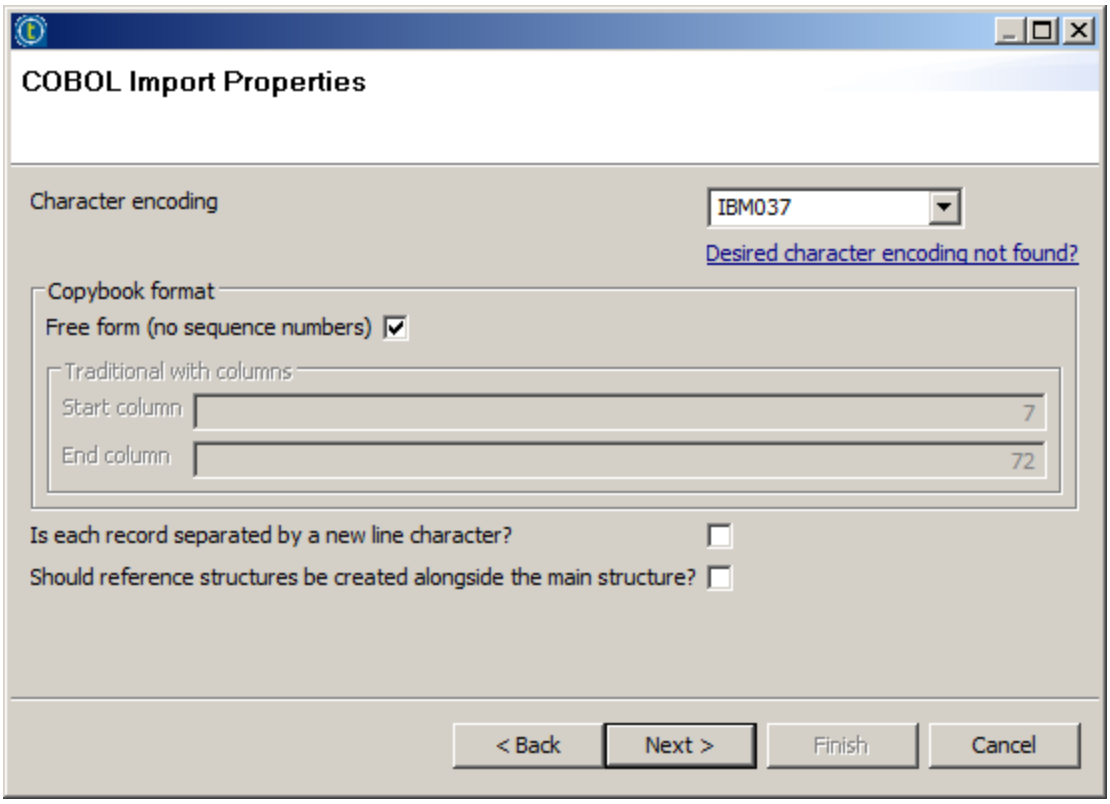
4. Select **COBOL Copybook** and click **Next >**.



5. Select **Local file**, click **Browse...** to select *C:\StudentFiles\COPYBOOK-IN.txt* then click **Next >**.



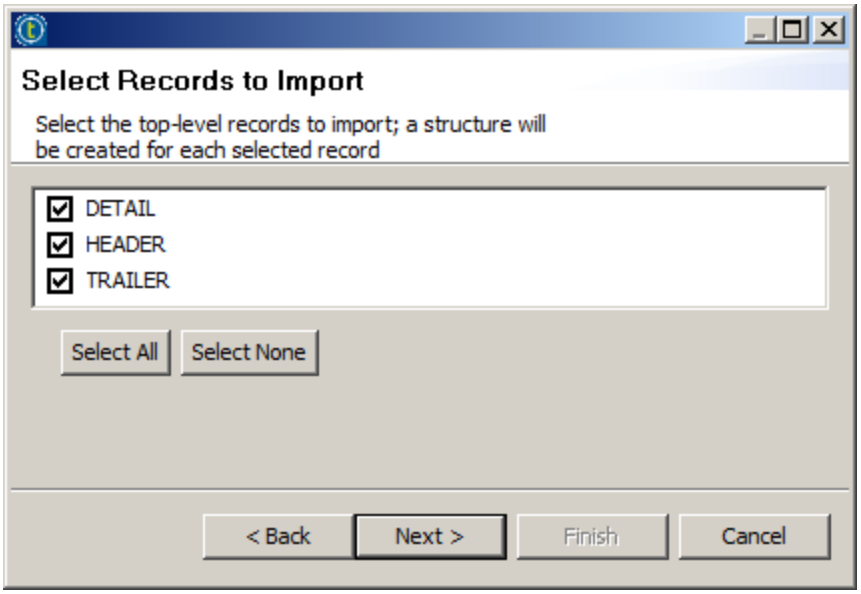
6. Set **Character encoding** to *IBM037* (because the data file is in EBCDIC format, not ASCII), also check the **Free form (no sequence numbers)** box and click **Next >**.



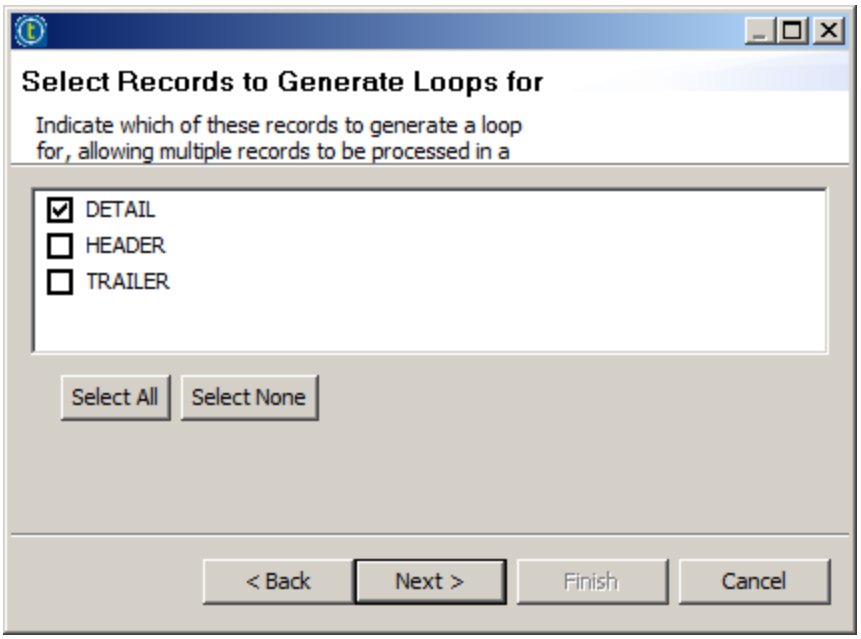
7. Three records should be detected and selected for import, as illustrated below.

   As mentioned in the dialog, a Structure will be created for each selected record. In this case, it means a total of three Structures will be created. As only one Structure can be used on each side of a Map, the next section will focus on creating a single output Structure that inherits from these three, so it still contains all the relevant information included in the initial three records.

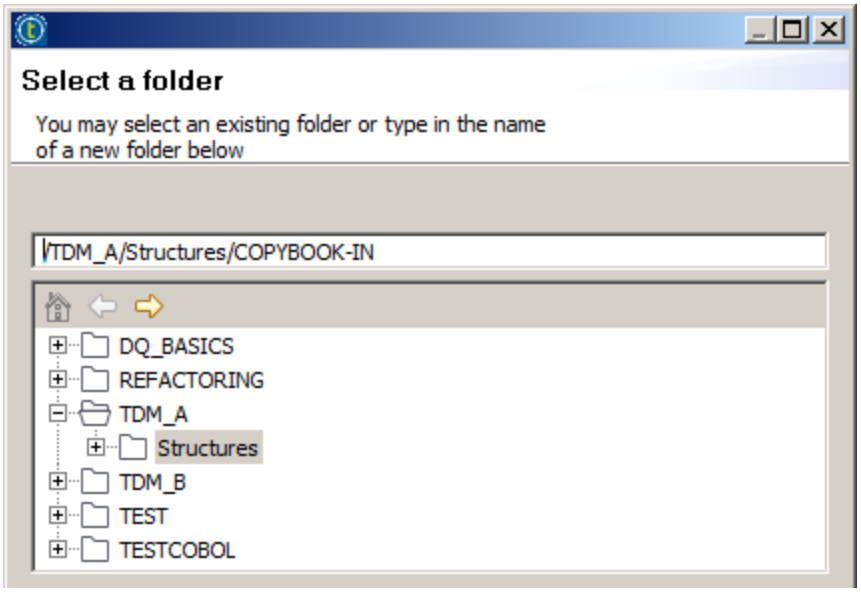
   Click **Next >** to go to the next step.



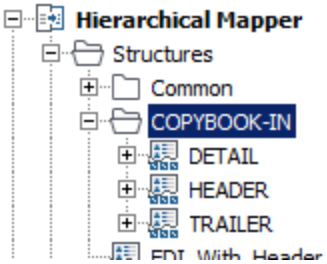
8. Uncheck **HEADER** and **TRAILER** in the next screen, because no loops are necessary for these elements. Only the **DETAIL** record appears multiple times in this file. Then click **Next >**:



9. Keep the default saving location for **COPYBOOK-IN** and click **Next >**.



10. Click **Finish** and check the Structures have been created.



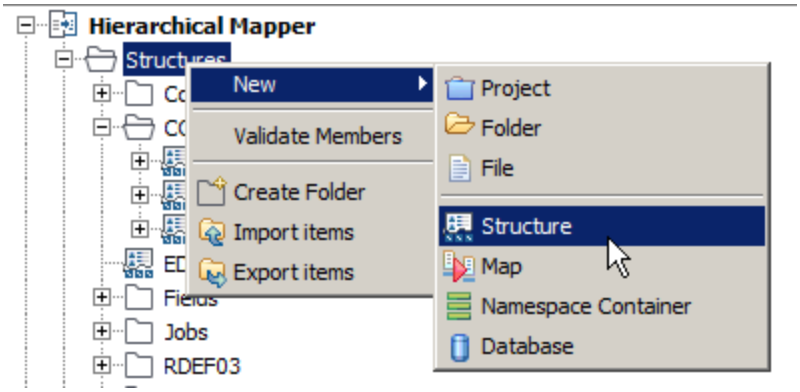
Next Step

Now let's create an outer Structure inheriting from the three Structures just imported.

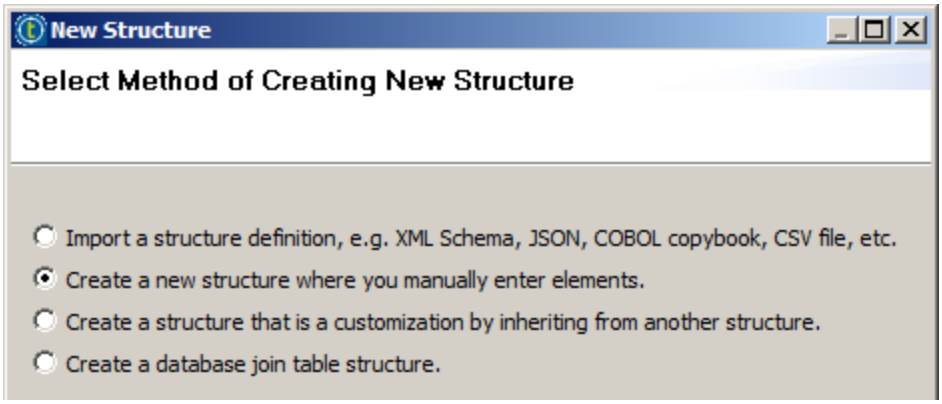
Creating the Outer Structure

A total of three Structures were imported in the previous section. As only one Structure can be used on each side of a Map, this section focuses on creating a single outer Structure that inherits from these three, so it still contains all the relevant information included in the initial three records. This outer Structure will be used as both the input and the output of the Map created in the next section.

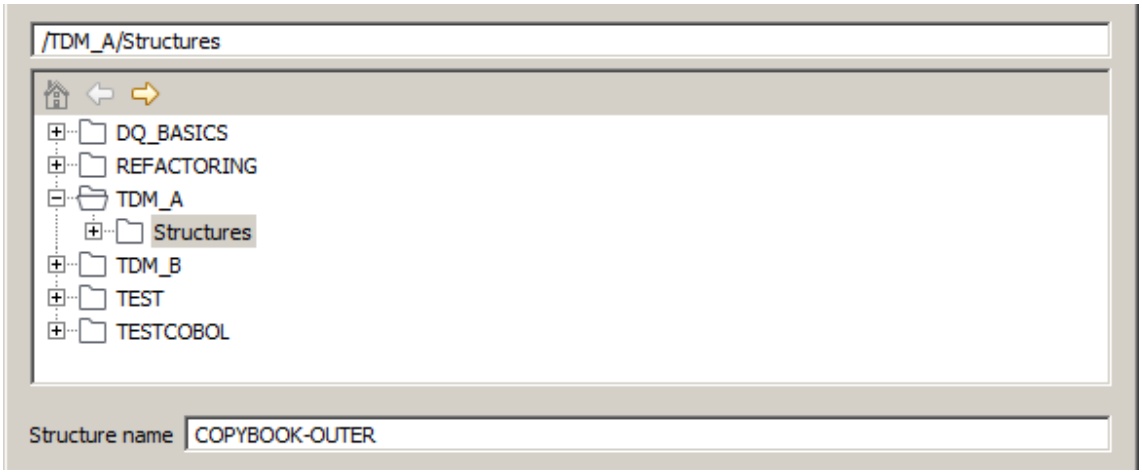
1. Right-click **Hierarchical Mapper > Structures**, then select **New > Structure**.



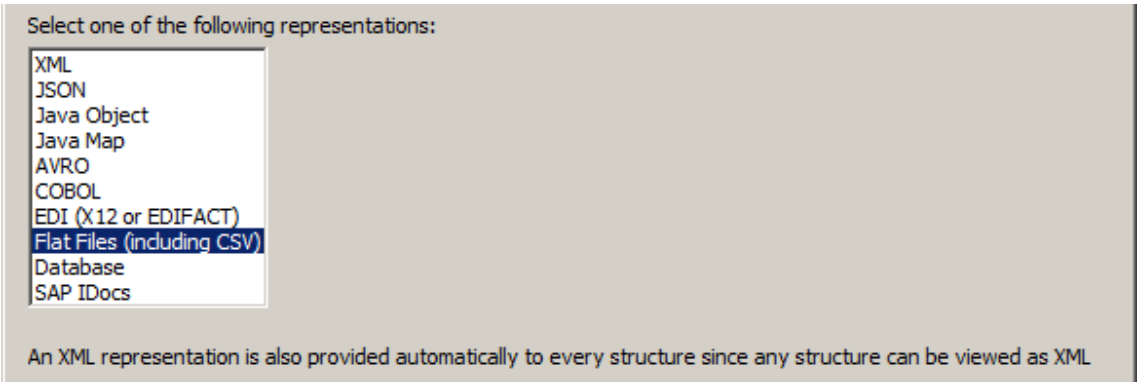
2. Select **Create a new structure where you manually enter elements** and click **Next >**:



3. Name the new Structure **COPYBOOK-OUTER** and click **Next >**.

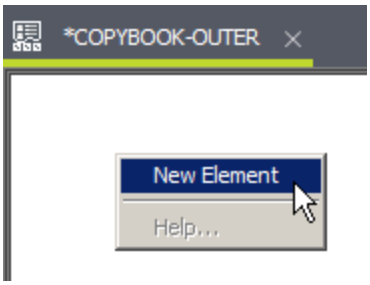


4. In the **Select Representation** step, select **Flat Files (including CSV)** and click **Finish**.

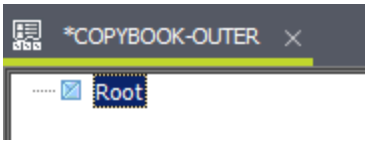


Now let's create the elements manually, and have them inherit from the Structures imported in the previous section.

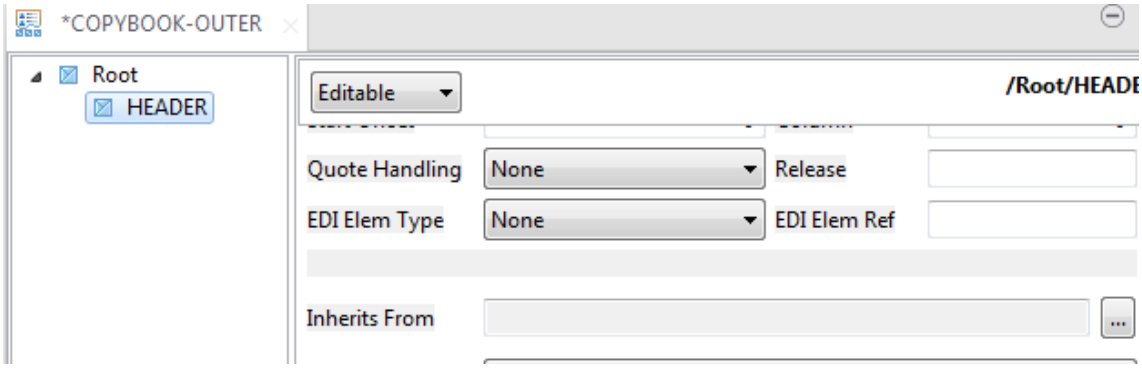
1. Right-click the empty area and select **New Element**.



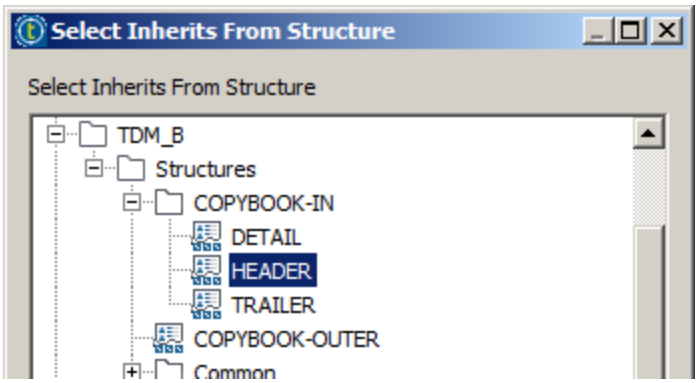
2. Name the new element *Root*.



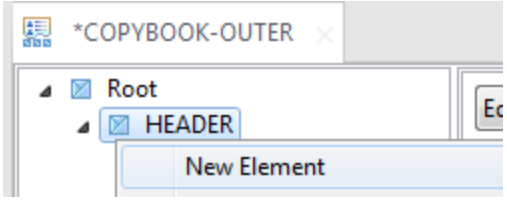
3. Right-click *Root*, choose **New Element** to add a child node and name it *HEADER*. Then click the **...** button to the right of the **Inherits From**field to select an ancestor.



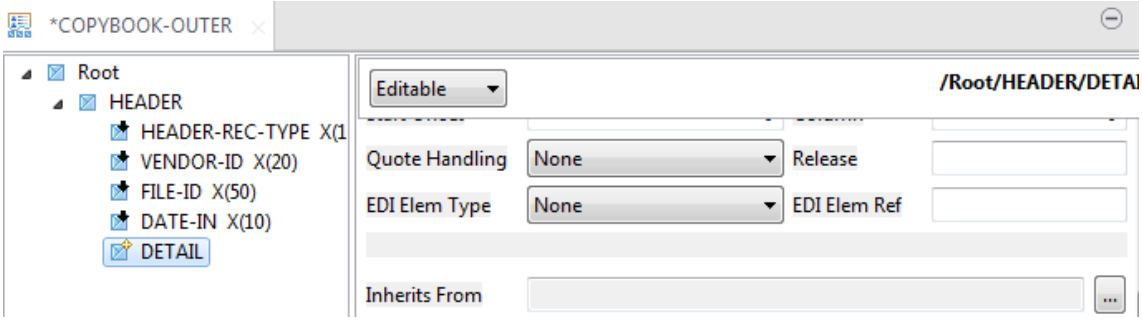
4. Select the *HEADER* structure imported previously and click **OK**.



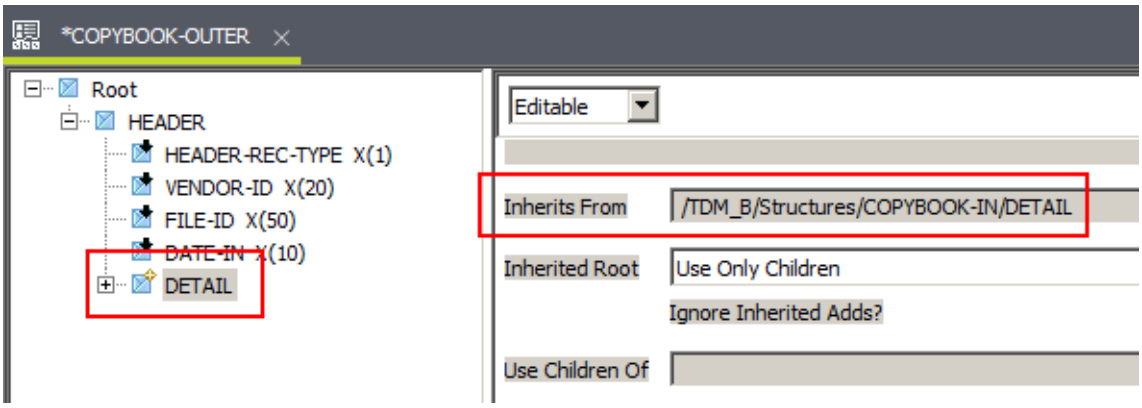
5. Right-click *HEADER* and select **New Element** again.



6. Name the new element **DETAIL**. Click the **...** button to the right of the **Inherits From** field.

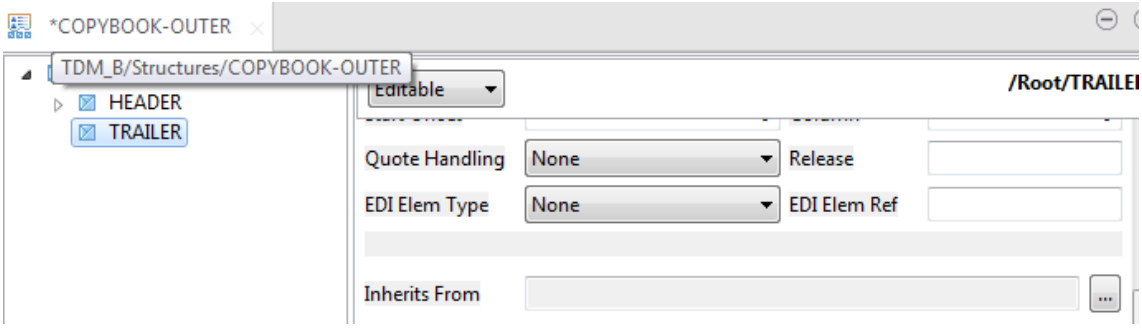


7. Select the **DETAIL** structure imported previously and click **OK**.

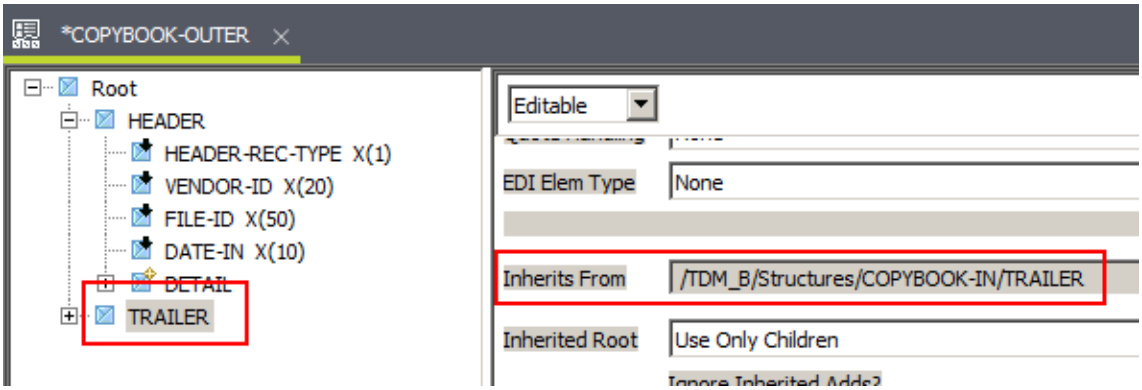


8. Now let's create a final element that will inherit from the last Structure imported previously.

   Right-click **Root**, select **New Element**and name it *TRAILER*.

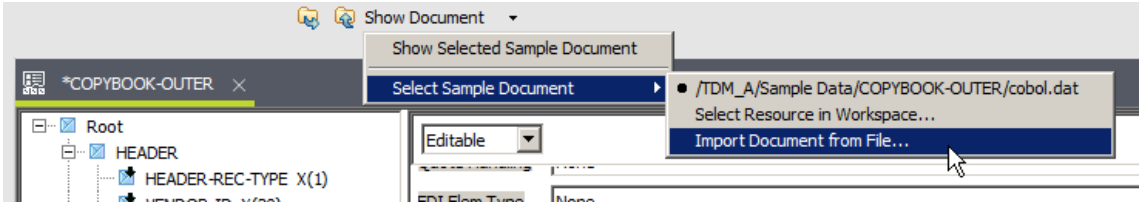


9. Click the **...** button to the right of the **Inherits From** field and select the **TRAILER** Structure:

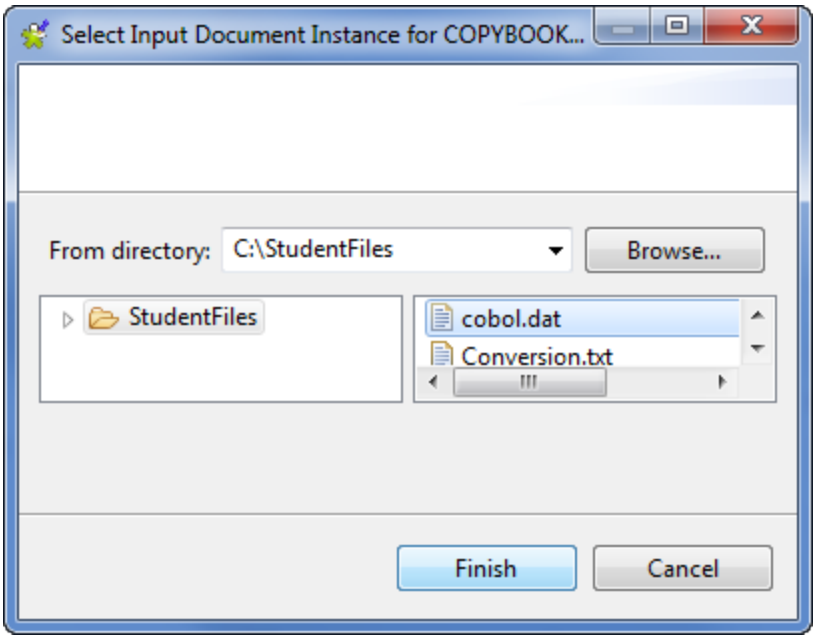


10. The Structure definition is complete. Now let's define a sample document to validate it.

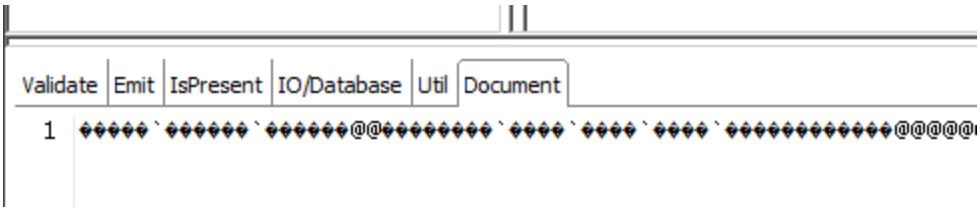
    Expand the **Show Document** menu and select **Select Sample Document > Import Document from File...**.



11. Select *C:\StudentFiles\cobol.dat* in the next screen and click **Finish**.

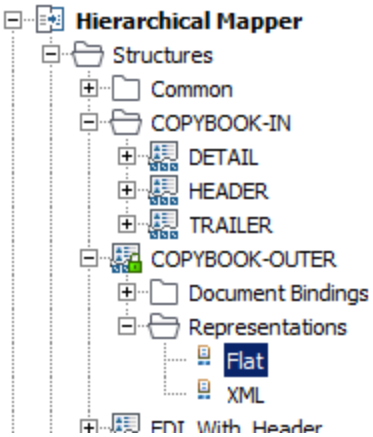


12. Note that the data shown in the **Document** tab will likely be unreadable due to the character encoding for the *COPYBOOK-OUTER* Structure which is set to *Platform Default*. The representation needs to be changed to *IBM-037* for EBCDIC files such as this one.

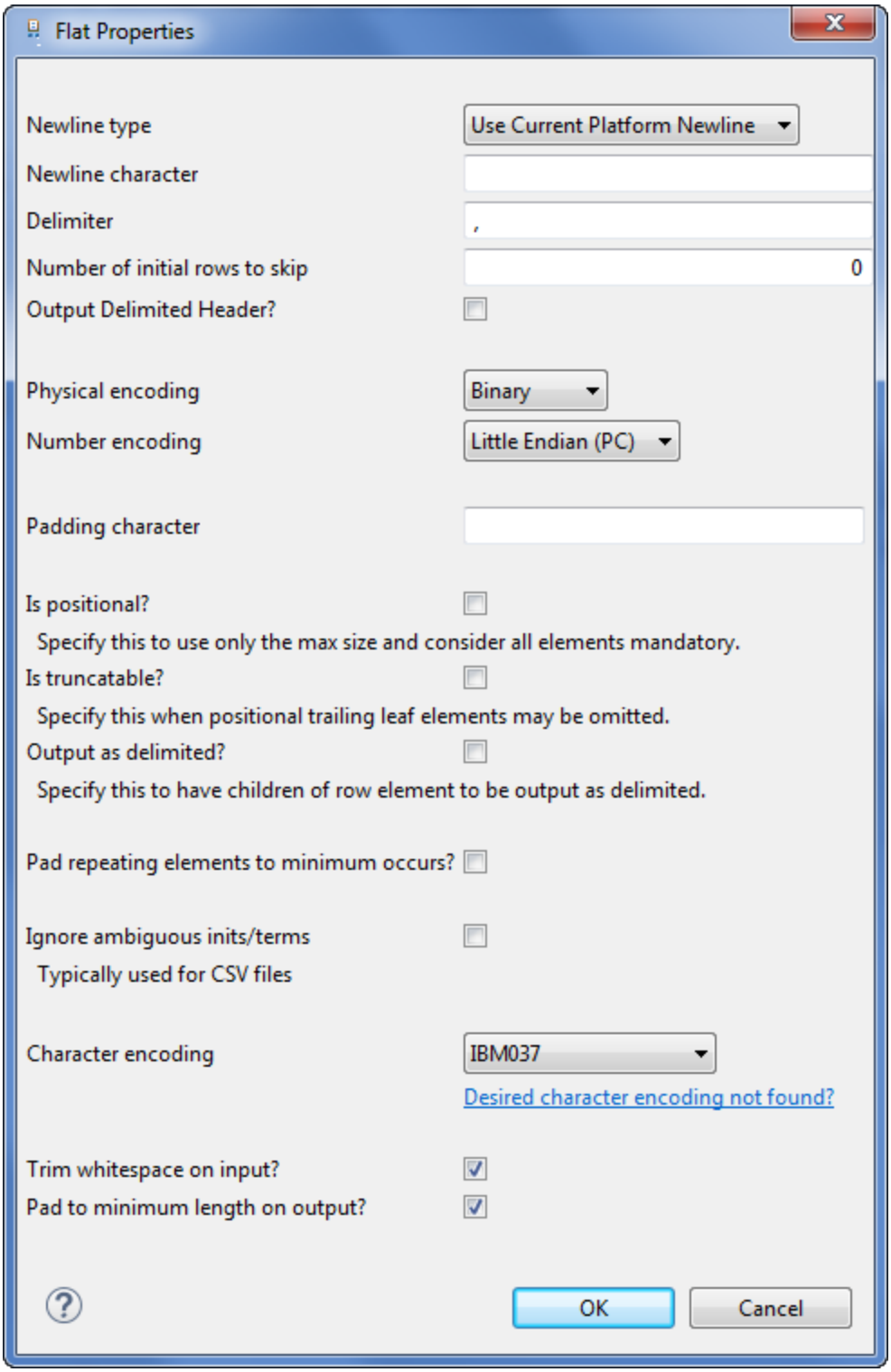


In addition, the physical encoding needs to be changed to *binary* because some fields in the EBCDIC data are packed and stored in binary mode.

Expand **Representations** under **COPYBOOK-OUTER** and double click **Flat** to define its properties.



13. Change **Physical encoding** to *Binary* and **Character encoding** to *IBM037*.



14. The **Document** tab should now be able to display the data is illustrated below.

    The 4 sets of columns on the left are the EBCDIC hexadecimal representation of the data.

    The human-readable data on the right is the ASCII representation of the same data.

For example, the first value *C8* on the left is the hexadecimal representation of the ASCII character *H*.

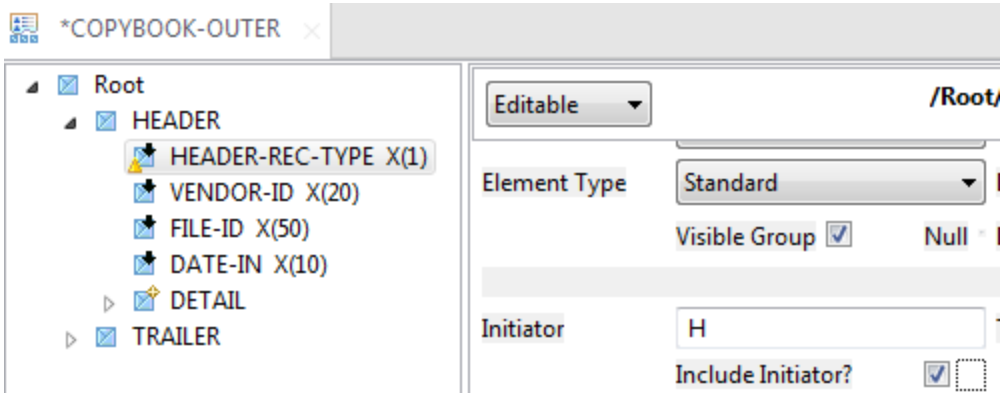
```
Validate | Emit | IsPresent | IO/Database | Util | Document

  0   C8E3C5E2 E360E5C5 D5C4D6D9 60E3C1D3   HTEST-VENDOR-TAL
 10   C5D5C440 40F1F6F9 C6C6C5F8 C160F7C6   END  169FFE8A-7F
 20   C3C460F4 F0F4C260 C1C1C3F5 60F2F1C2   CD-404B-AAC5-21B
 30   F9F1F8F9 C5C4C1F3 F5404040 40404040   9189EDA35
 40   40404040 40404040 40404040 40404040
 50   40C4D9C5 D4F0F1C2 8193A389 94969985    DREM01Baltimore
 60   40404040 40404040 40404040 40404040
 70   40404040 40404040 40404040 40404040
 80   40404040 40404040 40404040 40404040
 90   40404040 40404040 40404040 40404040
 A0   4040C4D9 C5D4F0F2 C2A48686 81939640     DREM02Buffalo
 B0   40404040 40404040 40404040 40404040
 C0   40404040 40404040 40404040 40404040
 D0   40404040 40404040 40404040 40404040
 E0   40404040 40404040 40404040 40404040
 F0   404040E3 F0F0F0F0 F2000000 00400C40      T00002.... .
100   40404040 40404040 40404040 40404040
110   40404040 40404040 40404040 40404040
120   40404040 40404040 40404040 40404040
130   40404040 40404040 40404040 40404040
```

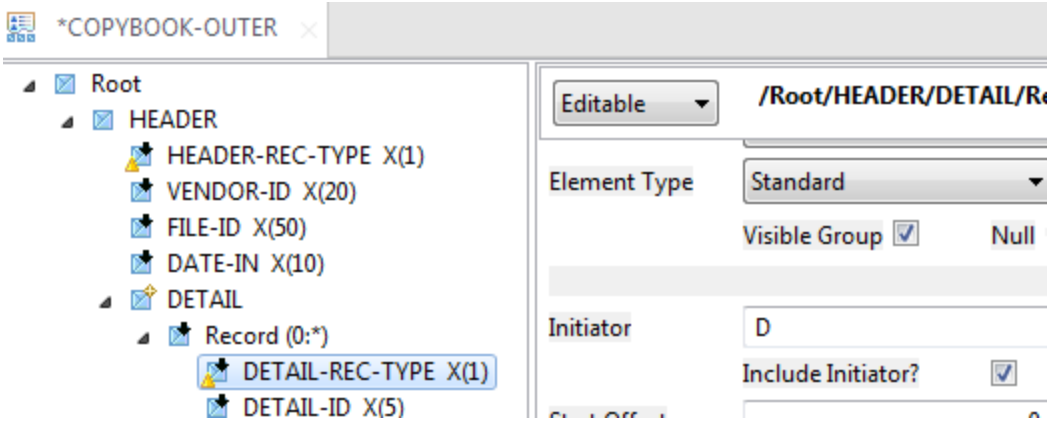
If your data is still not presented correctly, check the representations for the *HEADER*, *DETAIL* and *TRAILER* structures.They all should have a character encoding set to *IBM-037*. Only the TOTAL structure will have a physical encoding of binary because it's the only one containing packed fields.

15. The next step is to set up the Structure so that it is able to identify the different record types based on the first character – *H*, *D* or *T* (for Header, Data and Trailer respectively).

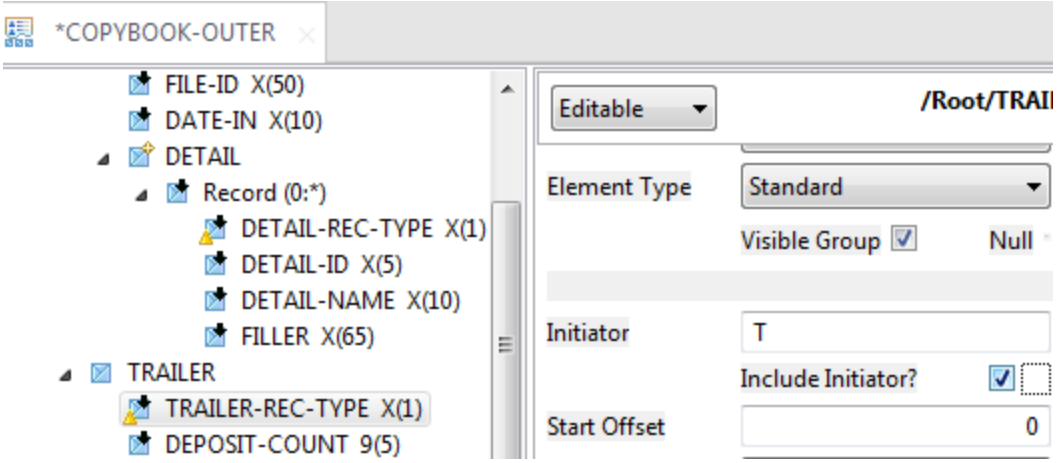
    First, select the **Root > HEADER > HEADER-REC-TYPE** element and enter *H* in the **Initiator** field. Also check the **Include Initiator** option.



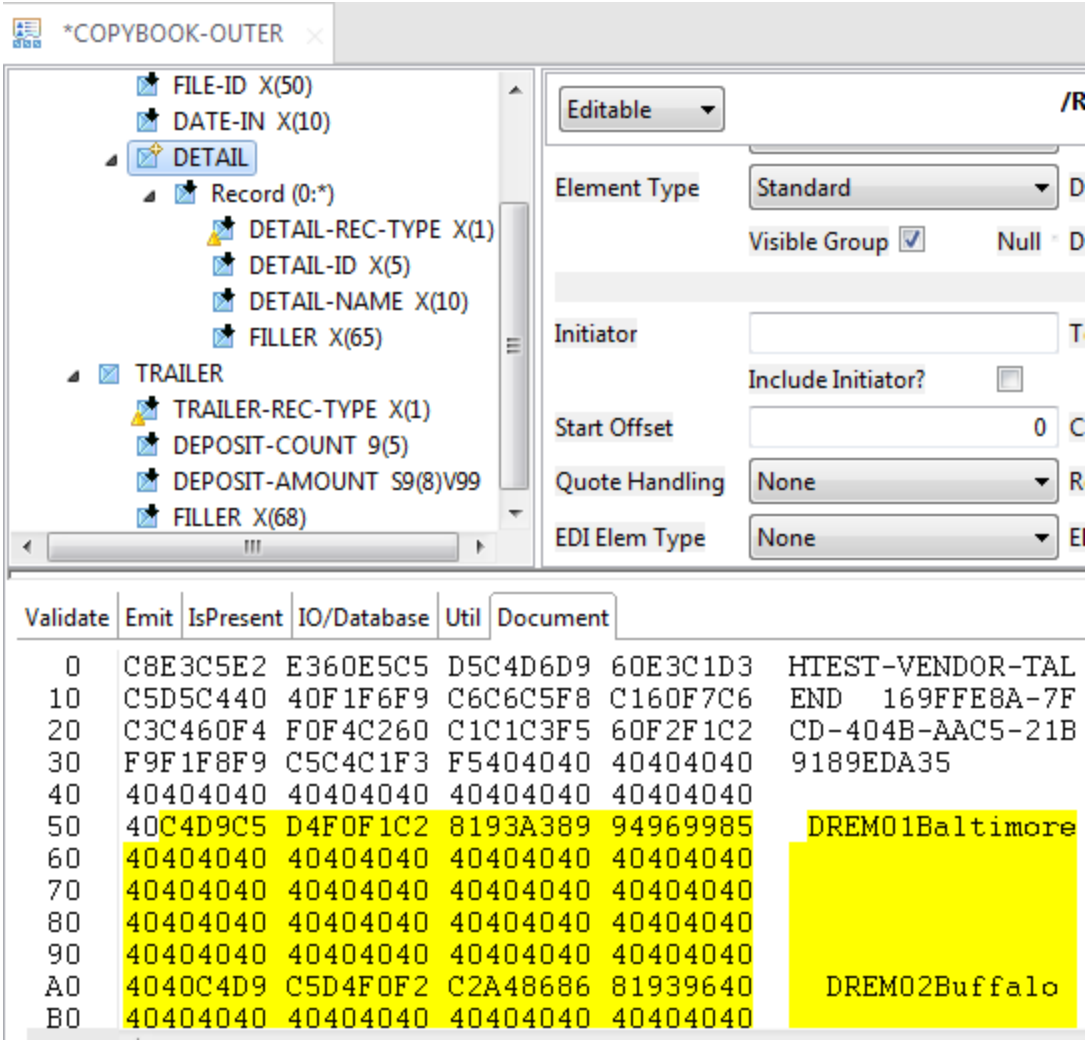
16. Select the **Root > HEADER > DETAIL > DETAIL-REC-TYPE** element and enter *D* in the **Initiator** field. Also check the **Include Initiator** option.



17. Finally, select the **Root > TRAILER > TRAILER-REC-TYPE** and enter *T* in the **Initiator** field. Also check the **Include Initiator** option.



18. Check that the record types are being identified correctly by clicking on each record type element and checking the highlighted data. For example, the detail data should be highlighted in the sample document when you click the *Detail* element:



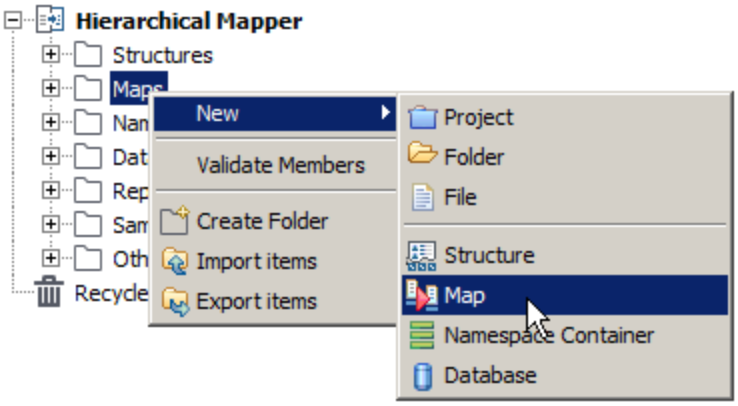
Note that, because of how the Structure is defined, clicking the *Header* element will highlight both the header *and* detail data.

Next Step

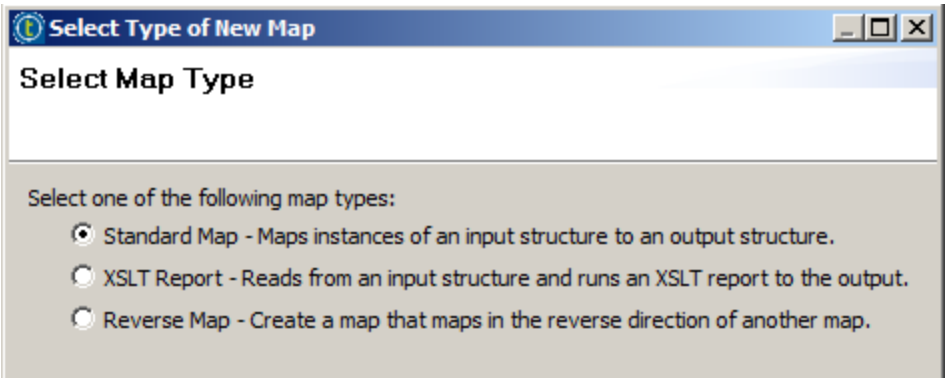
With this Structure created, let's create the Map responsible for converting the COBOL input to XML output.

Creating the Map

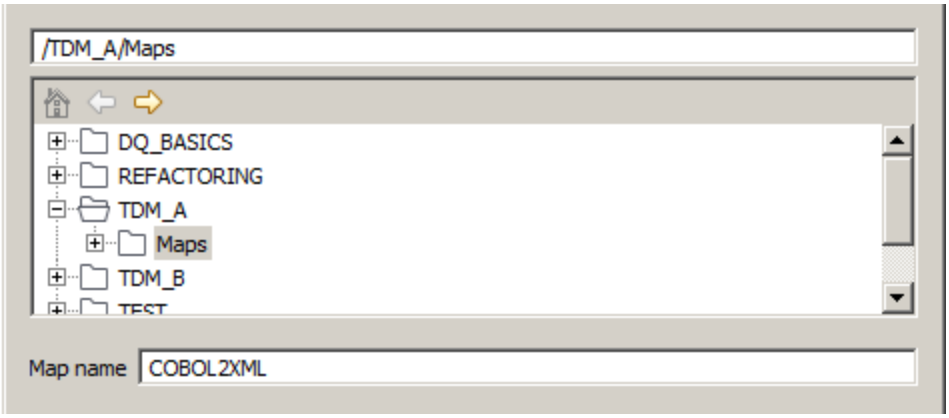
1. Right-click **Hierarchical Mapper > Maps** and select **New > Map**.



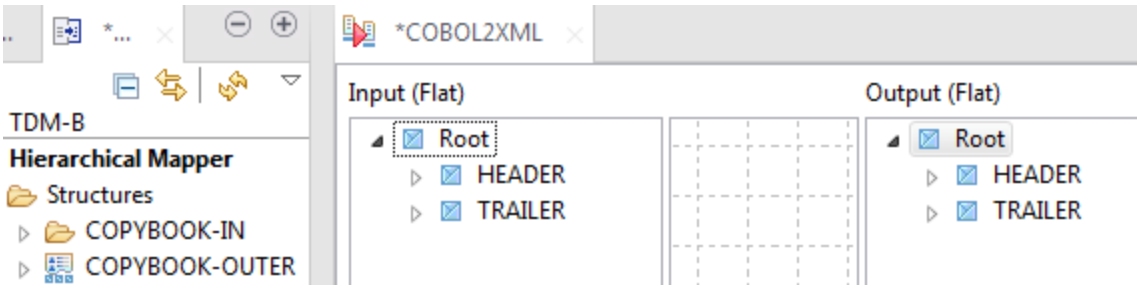
2. Select a **Standard Map** and click **Next >**.



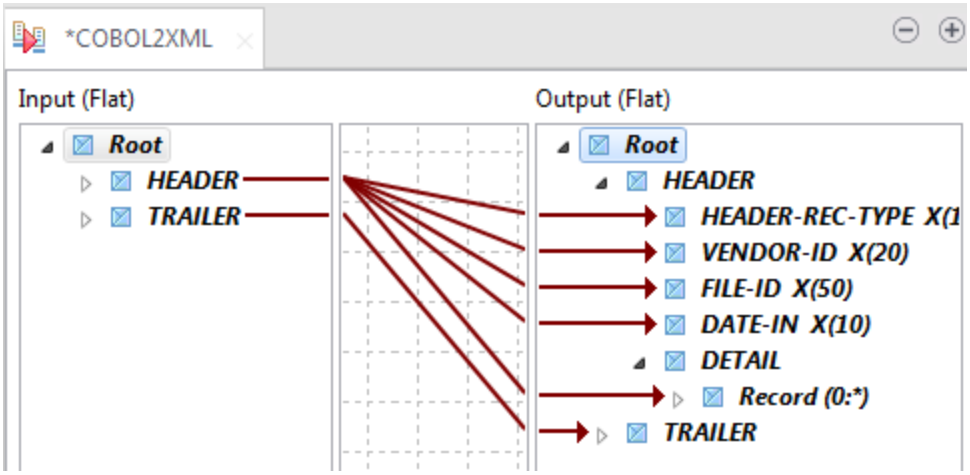
3. Name the new Map **COBOL2XML** and click **Finish**.



4. Drag and drop the **COPYBOOK-OUTER** Structure created in the previous section to both the **Input** and the **Output** areas.

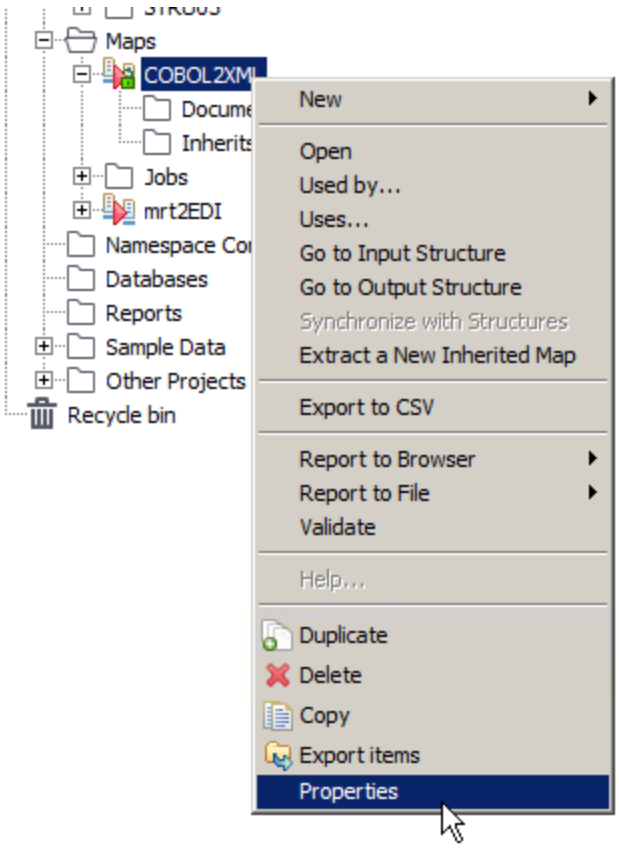


5. Leverage the automap feature by selecting the *Root* element on the **Input** side and dragging it to the *Root* element on the **Output** side.

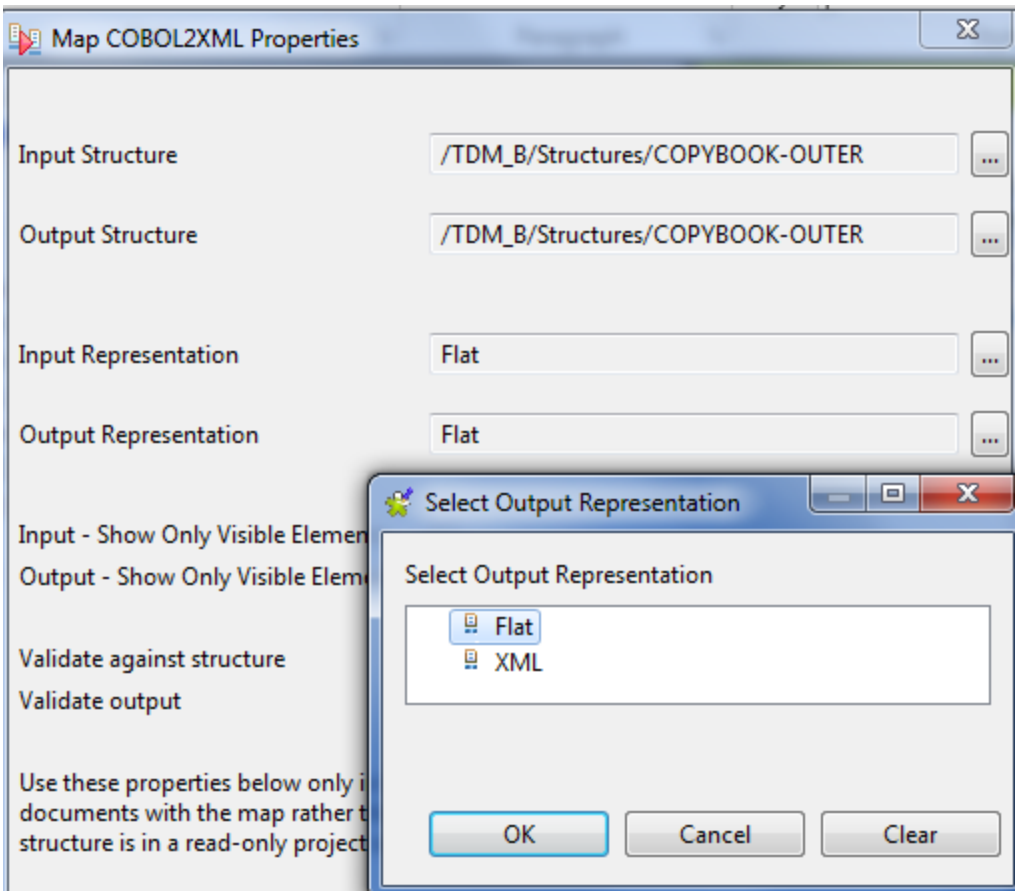


Notice that the **Output** file is defaulting to **Flat**. If you created a DI Job to execute this Map, the resulting file would be an **EBCDIC** flat file. In order to get an XML file, the **Output Representation** setting needs to be changed to **XML**.

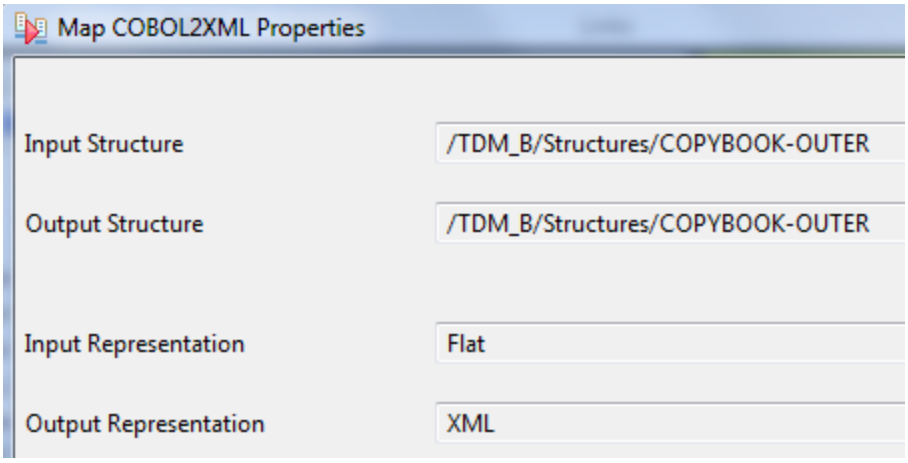
6. Right-click **Hierarchical Mapper > Maps > COBOL2XML** and select **Properties**.



7. Click the **...** button to the right of the **Output Representation** field, select **XML** and click **OK**:

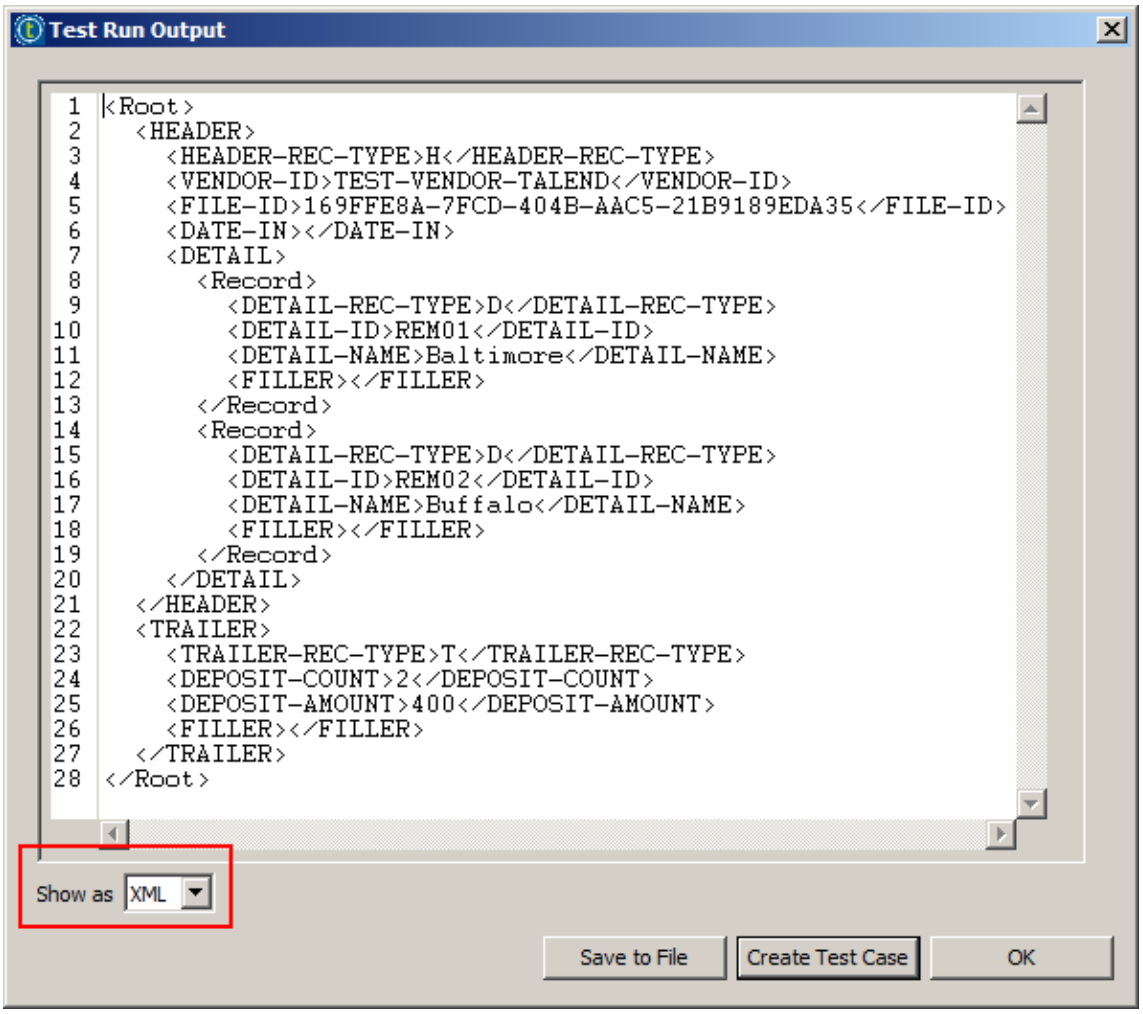


8. Check the Map properties look as illustrated below and then click **OK**.



9. The Map will be updated automatically, and the **Output** area should now be labeled *Output (XML)*.

10. Finally, click **Test Run** and check that the output looks like the following:




```
<Root>
  <HEADER>
    <HEADER-REC-TYPE>H</HEADER-REC-TYPE>
    <VENDOR-ID>TEST-VENDOR-TALEND</VENDOR-ID>
    <FILE-ID>169FFE8A-7FCD-404B-AAC5-21B9189EDA35</FILE-ID>
    <DATE-IN></DATE-IN>
    <DETAIL>
      <Record>
        <DETAIL-REC-TYPE>D</DETAIL-REC-TYPE>
        <DETAIL-ID>REM01</DETAIL-ID>
        <DETAIL-NAME>Baltimore</DETAIL-NAME>
        <FILLER></FILLER>
      </Record>
      <Record>
        <DETAIL-REC-TYPE>D</DETAIL-REC-TYPE>
        <DETAIL-ID>REM02</DETAIL-ID>
        <DETAIL-NAME>Buffalo</DETAIL-NAME>
        <FILLER></FILLER>
      </Record>
    </DETAIL>
  </HEADER>
  <TRAILER>
    <TRAILER-REC-TYPE>T</TRAILER-REC-TYPE>
    <DEPOSIT-COUNT>2</DEPOSIT-COUNT>
    <DEPOSIT-AMOUNT>400</DEPOSIT-AMOUNT>
    <FILLER></FILLER>
  </TRAILER>
</Root>
```

Note that changing **Show as** from **XML** to **Flat** at the bottom left of the pop-up window temporarily reverts the Output Representation setting defined in step 8.

Next Step

This lesson is almost over. Head to the Wrap-Up section for a summary of the concepts reviewed in this lesson.

Wrap-Up

In this lesson, you learned how to:

- Import multiple records from a COBOL copybook and create one Structure for each
- Merge several Structures into one outer Structure by defining inheritance manually
- Change the output representation of a Map from Flat to XML

Next Step

Congratulations, you successfully completed this lesson. Click the **Check your status with this unit** button below in order to save your progress. Then click **Completed. Let's continue >** on the next screen to jump to the next lesson.