Talend User Components tXmlJaxb*

Purpose

This bundle of components is dedicated to work with XML documents in the most flexible and unlimited way.
The idea behind these component is to work with JAX-B classes within Talend and without the need of an external Java project to build these classes. As source of the structure information can serve XSD or WSDL files (also with referenced XSD or WSDL).

Following components exists:

Component	Purpose
tXmlJaxbModel	This component builds and loads the JAX-B annotated classes
tXmlJaxbInput	Provide the attributes (also from the embedded objects) to a normal Talend schema
tXmlJaxbOutput	Creates an object instance and set their member variables.
tXmlJaxbSave	Renders the final XML tree pretty formatted as String to a file or schema column

Because of the objects are never duplicated they can be read and written within the same job.

Talend-Integration

You find these components in the studio in the palette under XML.

Component tXmlJaxbModel

This component reads the xsd- or wsdl from a file and builds the java classes and loads them immediately into the JVM. It is only once necessary to put this component on your job for one xsd or wsdl. All generated classes are also available for the embedded Talend jobs. For services it a supposed to put this component in the tPreJob chain to generate the classes right at the start of the service and not at the first request.
This component detects if a model is in the same JVM already loaded and does not repeat the class generation and loading.

Basic settings

Property	Content
XSD or WSDL File	Setup here the XSD or WSDL file for which you want to build the classes.
Target dir for generated classes	Set here the directory in which the classes will be generated. It is a good practice to use a sub folder under the directory for the XSD or WSL file. This helps to prevent losing the knowledge about the origin of the classes. The classes generated here can savely be deleted.
Create jar file	Option allows to build a jar file from all generated classes. This way you can this component a bit as development tool and use the generated jar in a different job or service without the need to build the classes again.
Jar file path	The path the jar file to be generated.

The generate process prints usage information for the generated classes. This helps to indentify the later necessary class names and attributes.

This is an example of the output. Please do not mix-up the term Context here. The word here means indeed the JAX-B context and not the Talend context.

```
JAX-B Contexts start ##################################
class de.cimt.customer.Company - is ROOT
    name type: java.lang.String
    CUSTOMER type: java.util.List
    NAME type: java.lang.String
    customer type: java.util.List

----------------------------
class de.cimt.customer.Customer - is ROOT
    address type: java.util.List
    FIRSTNAME type: java.lang.String
    title type: java.util.List
    NAME type: java.lang.String
    lastname type: java.lang.String
    firstName type: java.lang.String
    LASTNAME type: java.lang.String
    ADDRESS type: java.util.List
    POADDRESS type: de.cimt.customer.Customer$Poaddress
    name type: java.lang.String
    TITLE type: java.util.List
    ID type: de.cimt.customer.Idvalues
    id type: de.cimt.customer.Idvalues
    age type: javax.xml.datatype.XMLGregorianCalendar
    AGE type: javax.xml.datatype.XMLGregorianCalendar
    poaddress type: de.cimt.customer.Customer$Poaddress

----------------------------
class de.cimt.customer.Customer$Address
    CITY type: java.lang.String
    city type: java.lang.String
    street type: java.lang.String
    houseNumber type: java.util.List
    HOUSENUMBER type: java.util.List
    ID type: int
    STREET type: java.lang.String
    id type: int

----------------------------
class de.cimt.customer.Customer$Poaddress
    CITY type: java.lang.String
    city type: java.lang.String
    street type: java.lang.String
    houseNumber type: java.util.List
    HOUSENUMBER type: java.util.List
    ID type: int
    STREET type: java.lang.String
    id type: int

----------------------------
class de.cimt.customer.Address
    CITY type: java.lang.String
    city type: java.lang.String
    street type: java.lang.String
    houseNumber type: java.util.List
    HOUSENUMBER type: java.util.List
    ID type: int
    STREET type: java.lang.String
    id type: int

----------------------------

JAX-B Contexts end ###################################
```

Take note the actual single class Address is now represented by 2 classes because there are 2 different attributes to the the address in the Customer! To allow the automatical assigment of the child class to the parent class attribute it was necessary to build up the classes this way. It simplifys the configuration.
Also important here is the existence of the attributes in capital letters. They are synonyms of the original attributes and helps to simplify the attribute setting. In later versions of these components, this also helps to simply work with database result sets as source.

… and here the corresponding xsd file:

```
<xsd:schema
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xjac="http://java.sun.com/xml/ns/jaxb/xjc"
   xmlns:xasc="http://java.sun.com/xml/ns/jaxb"
   xmlns="http://cimt.de/customer/"
   targetNamespace="http://cimt.de/customer/"
   elementFormDefault="qualified">

   <xsd:element name="company">
          <xsd:complexType>
             <xsd:sequence>
                <xsd:element name="name" type="xsd:string" minOccurs="0"/>
                <xsd:element ref="customer" minOccurs="0" maxOccurs="unbounded"/>
             </xsd:sequence>
          </xsd:complexType>
   </xsd:element>

   <xsd:complexType name="address">
      <xsd:sequence>
         <xsd:element name="id" type="xsd:int" minOccurs="1" maxOccurs="1"/>
         <xsd:element name="street" type="xsd:string" minOccurs="0"/>
         <xsd:element name="_house_number" type="xsd:integer" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element name="city" type="xsd:string" minOccurs="0"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:element name="customer">
          <xsd:complexType>
             <xsd:sequence>
                <xsd:element name="id" type="idvalues" minOccurs="0"/>
                <xsd:element name="name" type="xsd:string" minOccurs="0"/>
                <xsd:element name="address" type="address" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element name="poaddress" type="address" minOccurs="0" maxOccurs="1"/>
                <xsd:element name="age" type="xsd:date" minOccurs="0"/>
                <xsd:element name="title" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
             </xsd:sequence>
             <xsd:attribute name="first_name" type="xsd:string"/>
             <xsd:attribute name="_lastname" type="xsd:string"/>
          </xsd:complexType>
   </xsd:element>


   <xsd:simpleType name="idvalues">
          <xsd:restriction base="xsd:string">
               <xsd:enumeration value="A"/>
               <xsd:enumeration value="B"/>
               <xsd:enumeration value="C"/>
          </xsd:restriction>
   </xsd:simpleType>
</xsd:schema>
```

Return values

Return value	**Content**
ERROR_MESSAGE	If the parsing will fail, the error message goes here.
XSD_FILE	The path to the used wsdl or xsd file. (do not be confused, the wsdl option comes later and the name of this variable still remains unchanged)
MODELCACHE_DIR	The path to the root directory of the generated classes
JAR_FILE	The ful path to the generated jar file if it is relevant.

Component tXmlJaxbInput

This component is used to read values from the JAX-B object.
It can build a hierarchy of components (also with tXmlJaxbOutput) to reflect the XML document structur.
The component reads in a xml source and builds the JAX-B object instances. After that it provide access to these objects and their attributes.

Basic settings

Property	**Content**
Is root	Indicates the component have to read the values from the root node.
Parent object	If it is not the root node, this setting points to the parent object.
Attribute path to loop	This is the attribute of the parent object to address the current object from which we want to get the values. This attribute is actually a lightwight x-path expression. It can deal with / or . to allow to address child objects deeper than member of the parent object.
Attributes	Configure here the attributes you want to read from the jax-b addressed object. **Column:** the schema column **Alternative Name:** You can set here the name of the attribute if it is not the same as the schema column name. If this expression results in an empty or null name, the schema name will be used instead. **Use Column:** Decide here which column you want to fill from the jax-b object. **Ignore missing attribute:** Set this option if the attribute can be missed. The default option is off.
Iterate over value list	With this option you can pick up one list attribute (it must be a list attribute) and let the component iterate over this list. The choosen schema column will be filled with the values from the list and all other schema columns keep their values.
Value list column	Choose the schema column which name actually refers to a list attribute in the xml object. The schema data type must be of the type of the values within the list.
Schema	The default schema editor.

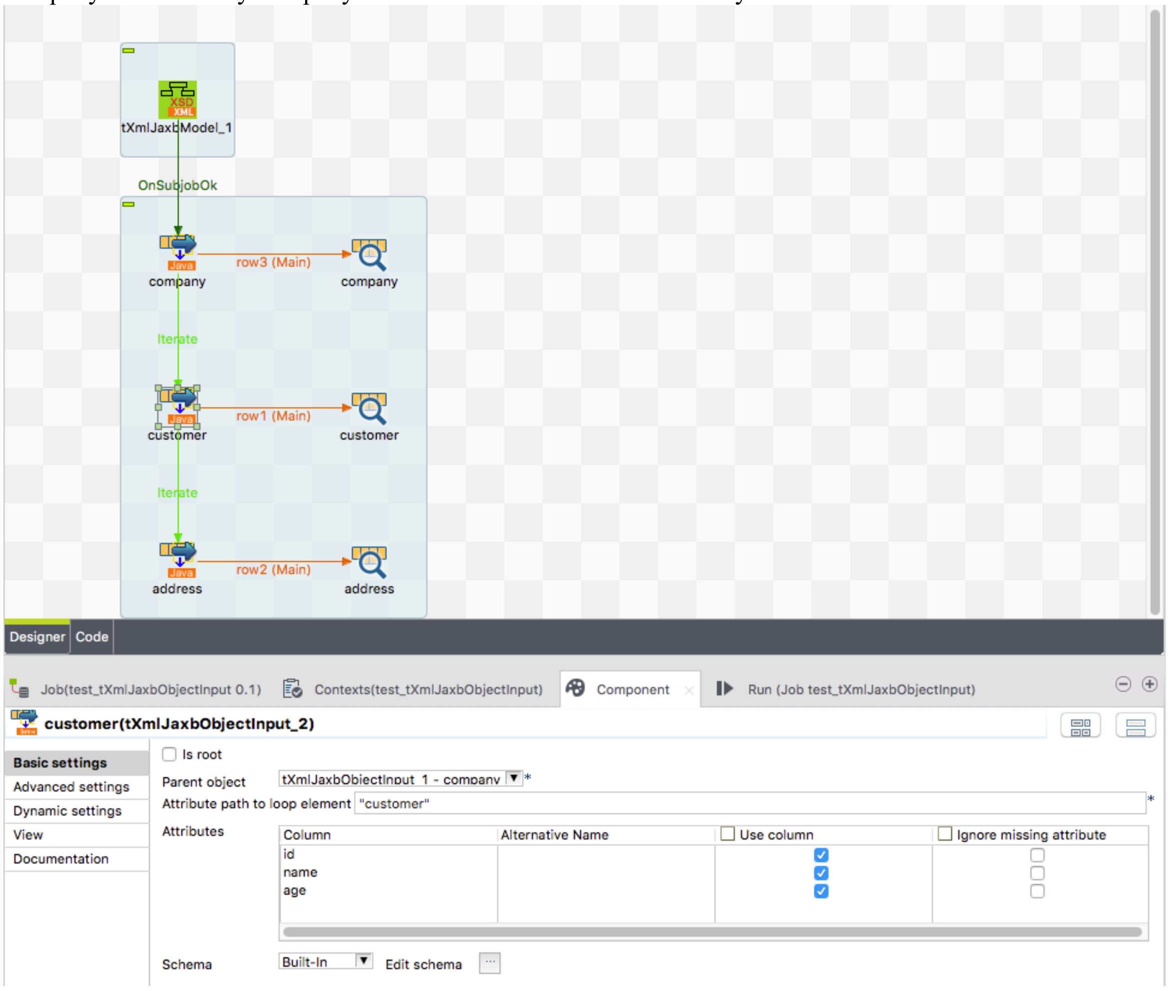
The read object will be detected by the attribute type of the jax-b object. Therefore, it is not necessary to mentioned the addressed object type.

Return values

Return value	**Content**
ERROR_MESSAGE	If the parsing will fail, the error message goes here.
CURRENT_OBJECT	This is the current jax-b object from which the attributes are currently read. This is potentially for every record another one if the parent attribute is a list.
NB_LINE	The number of outgoing rows.
FILENAME	In case of the component reads the content from a xml file the full path to the file will be available here. This is useful in case of you have a bit more complex calculation of the path instead of using a literal.

Scenario: Reading a document with multiple levels:

This shows how to chain the processing with Iterate flows. At first we read the model and after that we read the company and for every company we read the customers and for every customer its addresses.



customer(tXmlJaxbObjectInput_2)

☐ Is root

Parent object: tXmlJaxbObjectInput_1 - company *

Attribute path to loop element: "customer" *

Attributes

Column	Alternative Name	☐ Use column	☐ Ignore missing attribute
id		✓	☐
name		✓	☐
age		✓	☐

Schema: Built-In Edit schema ...

Even such a simple example shows the power of this approach. In a tXMLMap you cannot have a loop within a loop without re-aggregating the parent structure at the end.
Keep in mind how an iterate flow works: At first one plain flow record goes and per such a record one iterate goes. In other words: One iterate per one main flow record. Now take note the address component uses always the current customer object as parent as well as the customer component uses the just current company (it is the root here – not very difficult to use this one ;-)

Component tXmlJaxbOutput

This component is dedicated to create and write into jax-b objects.
It can be chained with other tXmlJaxbOutput or tXmlJaxbInput components and work relatively on top of the current object of the addressed parent component. The knowledge about the parent object and the class this component should be instantiate is quite enough to find the correct member in the parent object to set. If the parent attribute is a list, this component adds the new instance to the list, otherwise it simply set the current member again an again. You as developer must know which cardinality you have here and decide if your input flow have more than one record or not.

One important feature is to automatically assign child objects to their parents. The idea behind this feature is to prevent running queries against the database for every parent to get the children.

Basic settings

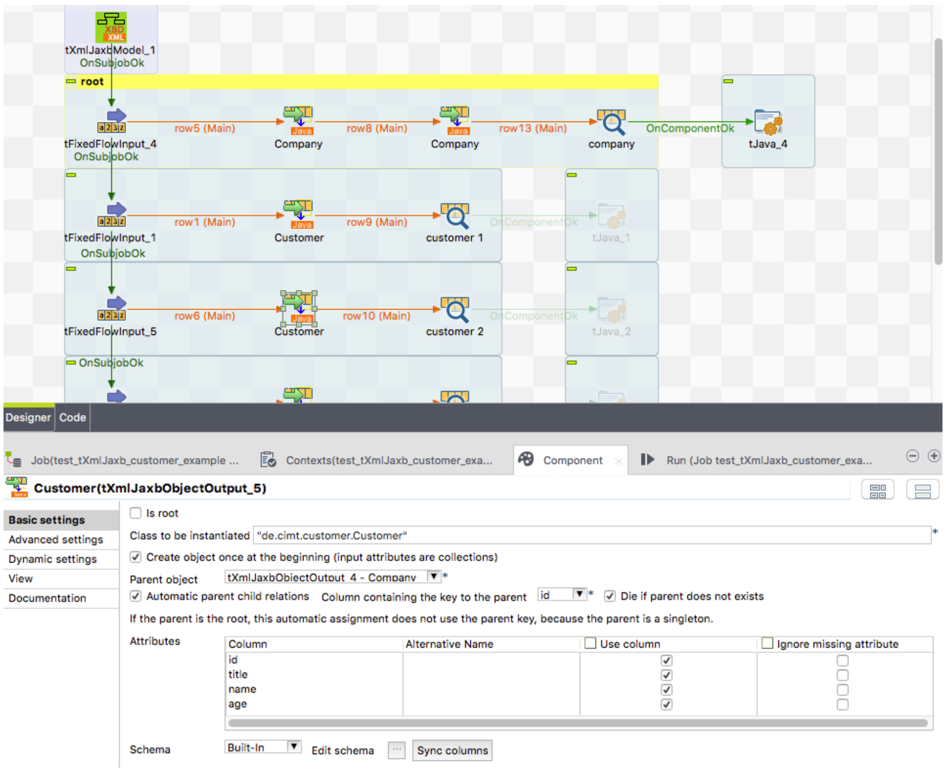
Property	**Content**
Is root	Decide if this object is a root element. In case of root this component does not tries to assign its objects to parent objects.
Class to be instantiated	Set here the full qualified class name you want to instantiate. To get an idea which class you need check the output
Create the output object right at the beginning	With the help of this option, you can write values into arrays.
Parent object	Choose here the component holding the parent object. Which parent-member will be written is quite clear by the class to be instantiate.
Automatic parent child relation	With this option the component assignes the current object to the its matching parent object
Column containing the key to the parent	Choose here the input schema column which contains the foreign key to the parent object. The parent component holds all objects based on its own key field.
Die if parent does not exists	Let the component die if the parent cannot be found based on the foreign key column value.
Attributes	Configure here the attributes you want to write into the jax-b addressed object. **Column:** the schema column **Alternative Name:** You can set here the name of the attribute if it is not the same as the schema column name. If this expression results in an empty or null name, the schema name will be used instead. **Use Column:** Decide here which column you want to write to the jax-b object. Especcally when you use a foreign key column it is very likely this column is not actually part of the jax-b object. Such a column is needed but leaf out for the jax-b object itself. **Ignore missing attribute:** Set this option if the attribute can be missed. The default option is off.
Schema	The default schema editor. Please use the Date pattern to parse the Date strings in the json document. JSON actually does not have a standard date pattern or even such a data type. It depends on string parsing to work with dates and timestamps. The component use a internal default pattern "yyyy-MM-dd'T'HH:mm:ss:SSS" if you do not provide a date pattern in the schema.

Return values

Return value	Content
ERROR_MESSAGE	If the parsing will fail, the error message goes here.
CURRENT_OBJECT	This is the current jax-b object to which the attributes are currently written.
NB_LINE	The number of incoming rows
KEYS_AS_IN_CLAUSE	In case of the automatic assignment from the child objects to their dedicated parents it is supposed to read from the database only the appropriated child objects. To support this, this component provides a comma separated list of all key values as String. This can easily be used to select the matching children in the database.

Scenario 2: Write a complex jax-b document

This scenario shows how to build a multi-level jax-b document. It uses the auto-assignment between children and parents. Therefor it is not necessary to use iteration here.



Customer(tXmlJaxbObjectOutput_5)

☐ Is root

Class to be instantiated "de.cimt.customer.Customer" *

☑ Create object once at the beginning (input attributes are collections)

Parent object tXmlJaxbObjectOutput_4 - Company *

☑ Automatic parent child relations Column containing the key to the parent id * ☑ Die if parent does not exists

If the parent is the root, this automatic assignment does not use the parent key, because the parent is a singleton.

Attributes

Column	Alternative Name	☐ Use column	☐ Ignore missing attribute
id		☑	☐
title		☑	☐
name		☑	☐
age		☑	☐

Schema Built-In

The input components are pure dummy input flows.

Component tXmlJaxbSave

This component is dedicated to provide the XML document as pretty formatted string to any kind of output.

Basic settings

Property	Content
XML Document root	Choose here the JXmlJaxbOutput component which current processing node should be used to render the document output.
Output file path	The path to the output file.
Schema	This schema is only necessary in case you want to write the content in the output flow of this component.
Output schema column	Choose the schema column in which the XML als String should provided.
Pretty Print	The string output will be formatted in a human readable way. Otherwise everything is in a condensed one line String.
Output as fragment	The pragma will be leaf out. This helps to build xml snippets which can be assemebled later to the final document.
Write content to standard out	Does what is says. This is actually I kind of debug option to see the content on the standard output of the job.

Return values

Return value	Content
ERROR_MESSAGE	If something went wrong, e.g. we cannot write into the file, the error message goes here.
OUTPUT_FILE_PATH	The path to the output file if set. This is useful if the path is calculated and you need that in the further processing.