Task Factory Pack Data Transform

Last Modified on 30 September 2020

📢 Task Factory users running version 2020.1.4 or older (released prior to May 27, 2020): **There's an important Task Factory update.** Please visit here for more details.

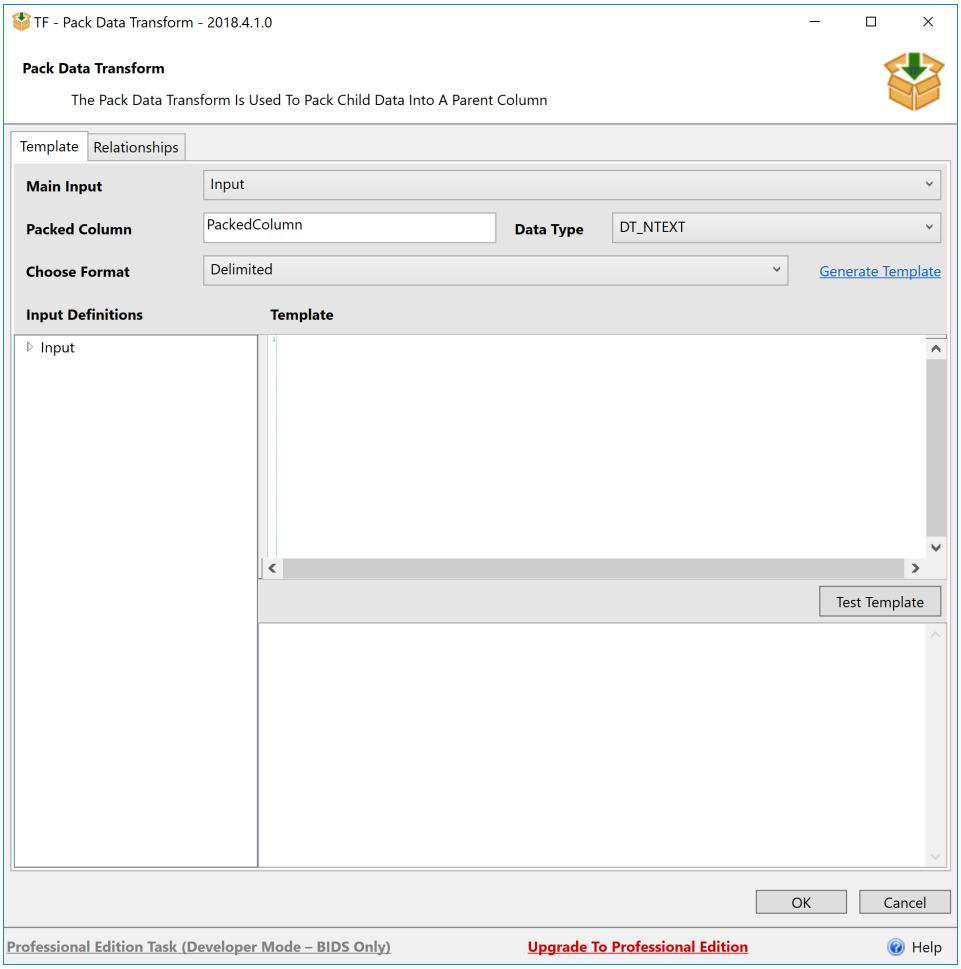
Pack Data Transform

Note:

- Pack Data is available for SQL Server versions 2012 and higher.
- When using multiple sources, all inputs must be sorted.

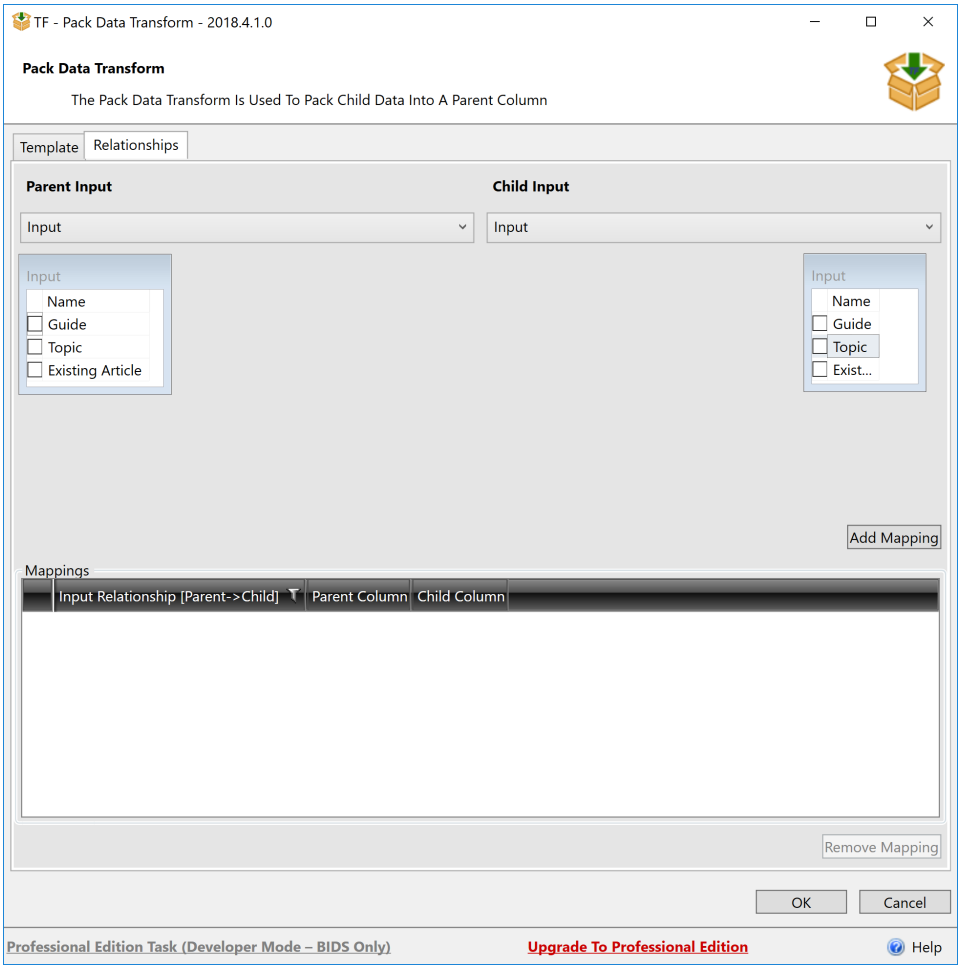
Transform Icon	Transform Description
	The **Pack Data Transform** is used to create a single column of delimited, XML, or Json data (determined by a user-defined template) from a single or multiple input columns.

Template Tab



Option	Description
Main Input	User selects the parent input.
Packed Column	User-defined name of the column to be output with packed data.
Data Type	User can select between two data types: DT_NTEXT or DT_TEXT.
Generate Template	Users can choose the packed data format (Delimited, XML, or Json) and the component automatically generates a template to be used for the packed output. These templates can be adjusted and customized by the user by deleting information not used or drag-and-dropping columns from the **Input Definitions** window.
Input Definitions	List of columns from the inputs.
Template	The design pane where users can view and customize the template used to generate the packed output.

Relationships Tab



Option	Description
Parent Input	Selects an input to identify as the parent.
Child Input	Selects an input to identify as the child.
Update Mapping	After connecting the two input keys, select this button to update the component with the defined relationship.
Remove Mapping	Selecting this button removes a selected relationship highlighted in the **Mappings** pane.

Pack Data Transform Generating Templates

The Pack Data Transform uses dotliquid as the templating engine which can be confusing for some users. The purpose of this page is to help users understand how the syntax is used within the component. This should also help users learn how to use the pre-configured templates as well as create their own within the component.

First, it's important to understand that you can create any delimited, Xml, or Json formatted template. To quickly generate one based on the input(s), select the **Generate Template** hyperlink. The Template window populates based on the selected format and input(s). Columns can also be added manually by dragging from

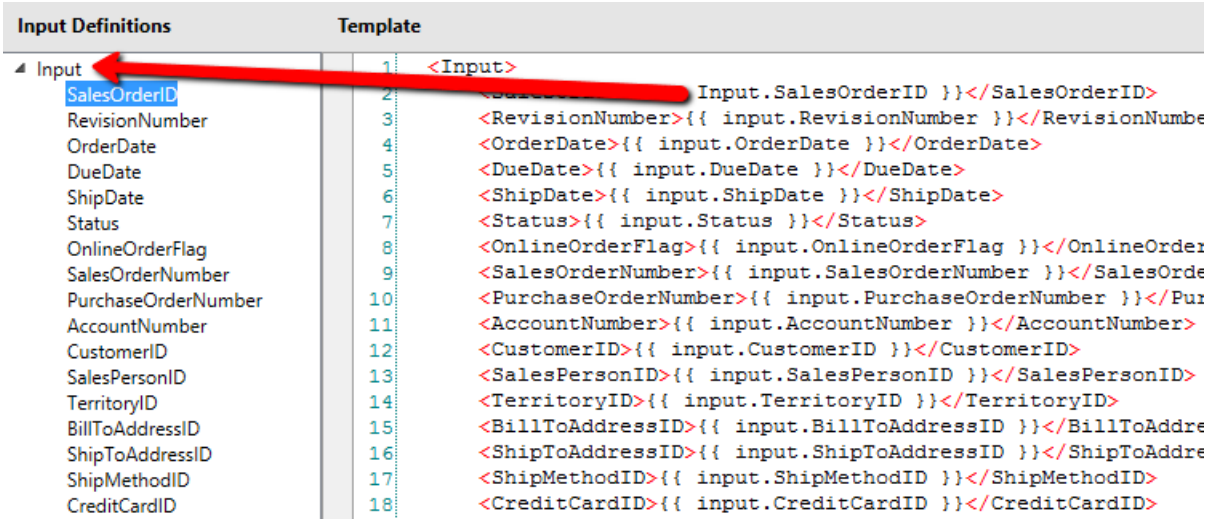
the **Input Definitions** window and dropping to the Template window.

Syntax

Data replacements are in the format of {{inputname.columnname}} Example: Everywhere the user sees {{Input.SalesOrderID}}, it will be replaced with the row data for SalesOrderID. It's further broken down as follows:

- Input = the name of the input on the left side of the UI (the top node of the Input Definitions window).
- SalesOrderID = column name from that input.

As you can see in the example below, the SalesOrderID input on the left corresponds to the {{Input.SalesOrderID}} in the XML on the right:



Adding IF statements

If conditional statements act the same as other languages (except uses the dotliquid syntax.) Users can add If statements by contributing the following:

```
{% if forloop.index > 1%},{% endif %}
```

IF Statements Example

```
{% if row1.CarID == 123 %} Write any text here {% endif %}
```

This reads **If CarID in row1 is equal to 123,** then **Write any text here** is added to the output. See the table below:

CarID	Info
123	Write any text here
456	Data
789	Data

CarID	Info

As you can see, the Info column added **Write any text here** because it matched the condition that **CarID = 123**.

> **Note:** Spaces must surround equality/inequality operators used in If statements.

Comments

Comments can be added between objects Simply use the following tags:

```
{% comment %} {% endcomment %}
```

Comments Example

```
{{input.SalesOrderID}}{% comment %} add comma between objects - this comment tag can be removed fro
m template {% endcomment %}
```

Loops

Loops can only be used with more than one input. In some cases, users may have to loop through more than one row that shares the same ID. To use loops in the Pack Transform, add the following:

```
{% for row1 in input1.Rows -%}
```

Row1 is the name of the loop iterator and is used to access any data from the rows in input1.

Loop Example

```
{{ row1.CarID }}
{{ row1.CarName }}
{% endfor -%}
```

Using more than one input with a parent – child relationship

The tables below are used to establish the parent-child relationship.

Parent Input has a single row with two fields. The parent is named **Input** (seen in the Input Definitions window).

OwnerID	Name
1	Jane Fields

Child input has two rows with two fields. The child is named **Input1** (seen in the Input Definitions window).

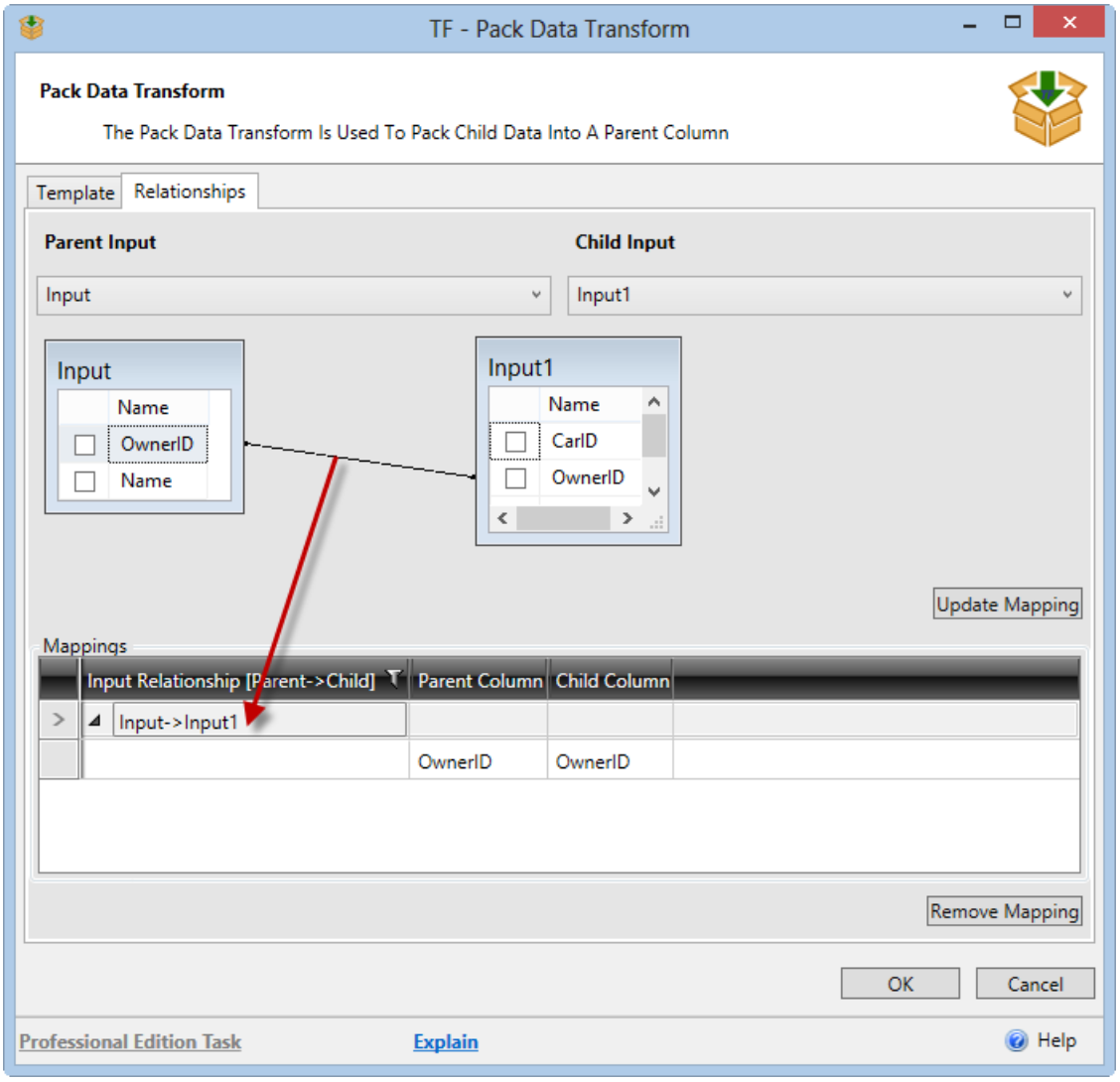
OwnerID	CarID	CarName
1	1	Maxima
1	2	Explorer

There is a relationship defined between the two inputs on OwnerID from Input and Input1.

Users can use the generate template once the relationship has been defined.

When using multiple inputs, a relationship between the two is needed in the relationship tab. To add this relationship, complete the following steps:

1. Select the Relationships tab.
2. Select the Parent Input in the first selection box.
3. Select the Child Input in the second selection box.
4. Drag the parent column key to the child column key that links the tables.
5. Select the **Update Mapping** button.
6. (Optional) Return to the Template tab and select the **Generate Template** hyperlink to auto-generate your template.



Multiple Input Example

We want to output an xml template defined as:

```
{{ input.Name }}
{{input.ID }}

{% for row1 in input1.Rows -%}
         {{ row1.CarID }}
         {{ row1.CarName }}
{% endfor -%}
```

In this example, row1 is the name of the loop iterator and is used to access any data from the rows in input1.

> **Note:** Templates with columns not contained upstream display a warning. Please ensure the column is available upstream or is not misspelled to dismiss the warning. This does not prevent the component from executing.

Additional Information: As noted previously, the Pack Data Transform uses dotliquid as the templating engine. Documentation on the syntax can be found here:

- Liquids for Designers
- Liquid Basics Introduction