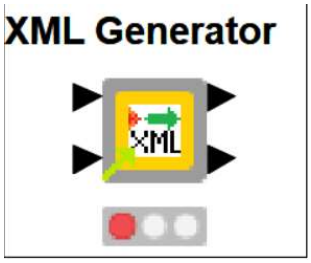


KNIME XML Generation Components

A quick guide to the (*experimental)* *XML Generation "toolkit"* by @takbb

doc version: v1 - 04 Nov 2022

Background and History

Generating XML is something that a lot of us find ourselves doing from time to time. With the prevalence of other data formats, XML is perhaps somewhat less popular than it once was but still there is still the need to be able to generate it and I have personally found it challenging to find a quick and easy way to generate it. In theory it ought to be straightforward but unlike tabular data, XML is structured and so some additional effort will always be required to generate it.

Many relational databases do provide a means for generating XML directly from SQL queries, but these often require a special syntax that is not standard across platforms and is not straight-forward. Sometimes the data is not in a relational database at all, and we do not have the time to go learning a whole new syntax for that one-off or very occasional task?

A tool such as KNIME allows us to think about tabular data without necessarily requiring an in-depth knowledge of the data source or even the destination format. I can write Excel spreadsheets without any knowledge whatsoever of the inner workings of XLSX files (which ironically are really XML plus a few bells and whistles wrapped in a ZIP file) or write to CSV without having to worry about embedding the delimiters or the quotation marks. It therefore makes sense that we should be able to generate XML without requiring vast knowledge of the workings of XML format itself.

A few years ago, before I discovered KNIME, I needed to be able to create XML files out of a database, and being a JAVA developer, I wanted my solution to be portable so that it would work with *any relational database that I could connect to through JDBC.* I wrote a small JAVA application for my own use that I configured using a "properties" file. It mostly worked for the XML I needed to create and occasionally I would need to tweak it a little where I required greater functionality.

Back in mid-2021, having discovered KNIME, I had a need to generate some XML and I found myself going down the same path of trying to find a "simple" (abstracted) solution. I looked at the facilities that KNIME offered for creating XML, and read some tutorials and examples, but the same thoughts from a few years earlier came back to me; I really didn't want to be hand-crafting the XML creation for every piece of new XML I had to generate. I found I was spending too much time with the "node-per-xml-element" route or having to think about how KNIME was generating the XML whereas what I really wanted to do was say "here is the data. Here is the structure I want…make it so". I'm lazy and get bored easily when it comes to mundane tasks; I want to just "crank the handle" and move on.

So, I investigated the possibility of porting my old java application into KNIME. At first, I considered Java Snippets but soon realised that python was the better solution because with python snippets you have access to the entire data table in one go, whereas Java Snippets work on a one-row-at-a-time basis which wasn't really compatible with what I wanted.

The downside of using python is that it means it isn't a totally "out of the box" solution, but it greatly simplified the implementation, and I considered it a price worth paying. As it turned out, I didn't actually port the *java code* into python. I ported the *idea* of how it worked and wrote it from scratch. Over time I have added some refinements. This is the result! Enjoy 😊

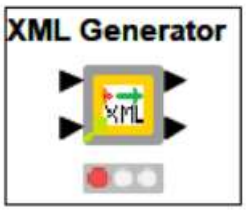
What are the XML Generation Components?

The XML Generation Components started off life as a Table Creator Node and a python script. The Table Creator node was where I entered the "properties" describing the xml to be built, and the python script acted on it. After some experimentation, the python script became the XML Generator component, and the Table Creator took on a life of its own, being replaced by several components written to help create the "Control Table", and to help modify it.

The "**XML Control Table**" is what I call the repository of configuration information that is used to describe the XML structure in simple, easy-to-process terms, and maps the XML elements to the different data items on the input data table.

Before getting into the detail of HOW the components work, I thought I'd take time to introduce them. I may modify these or add others in future as I discover shortcomings.

A word of warning though. These are "**experimental**". What this means is that I have put them together and tested them on a very limited set of sample data. They have worked for the tests I have done with them, but my time is limited and there may be situations where they don't work as expected. You are welcome to use them, but please… please… make sure you test that they are doing what you expect, and need. If you discover any bugs or have suggestions for improvement, please let me know on the KNIME Community Forum.

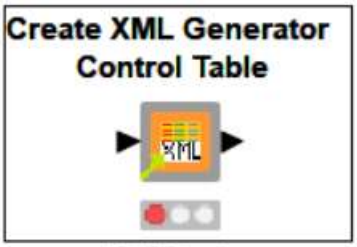


The **XML Generator** is ultimately the "brains" behind the whole operation. It takes two inputs. On the upper port is the table from which it will source the data that is to be turned into XML.

The lower input port is the Control Table that it will use as the "instructions" for how to do it.

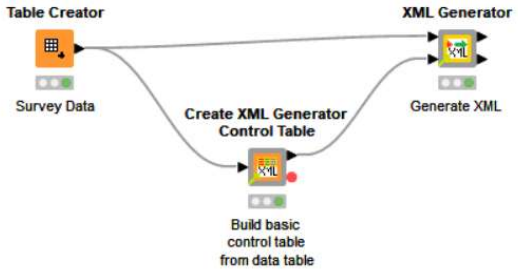
The upper output is the generated xml as an xml data type.

The lower output port is xml generated on a per-row basis as a String data type.



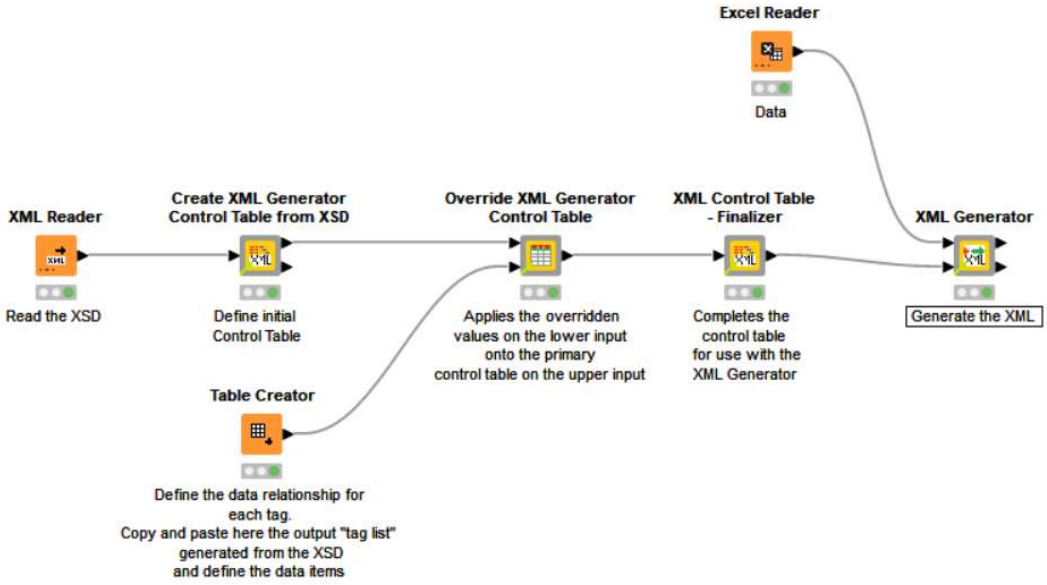
The **Create XML Generator Control Table** component was the first component written to replace the original manually configured "Table Creator". Its purpose is to take away some of the pain of supplying the Control Table. It cannot take away *all* the manual effort though. It isn't a mind-reader 😉

What it does is generate a basic xml control table for a give input data table. So, it reads in a data source and outputs a control table. In its most basic usage, as pictured here, this will generate basic XML. But unless basic row-level XML is what you are after, it is just the starting point.



The **Create XML Generator Control Table from XSD** component takes the alternative approach to building the initial Control Table. It attempts to use an XSD instead of the raw data to generate the Control Table. Unlike the basic Generator though, it knows nothing about the actual data table and so requires an additional manual step to tie the data to the control table before it can be used to generate XML.

The most basic workflow to generate XML with this component is as follows:



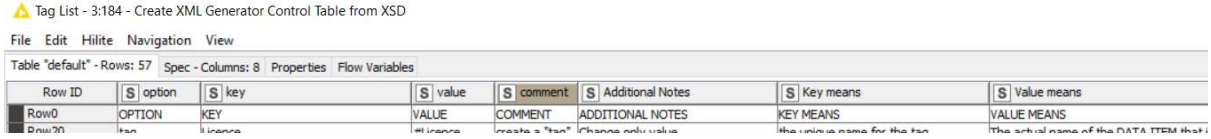
Once again though, there may be additional items that you would wish to configure although because this has used the XSD, it understands the required structure and so "out of the box" the above configuration will give a more advanced xml output than the basic xml generated using the "**Create XML Generator Control Table"** component.

There are two output ports on this component. The upper port is the Control Table that should be passed onto the next XML Generation component.

The lower port is an output of the "tag" entries created by this component. It is anticipated that this component should be followed by an **Override XML Generator Control Table** component which takes as its second input a partial control table (usually from a Table Creator as shown above).

To assist with populating this Table Creator, the lower port of the Control Table from XSD component supplies the starting point for the data for this Table Creator component:

1. View the output from this and copy and paste it into a Table Creator. Then modify the value column (third column) accordingly. There is no need to change the names of this table as the **Override XML Generator Control Table** works *positionally* rather than by column title, to reduce the effort required.



The **Create XML Generator Data Map Table from XML** parses a data mapping XML document and from it generates the data mapping portion of the Control Table. The data mapping control table is the "tag" rows of the control table which define, for each tag, the name of the data item in the input data table that is to be used to source the data for the given element or attribute.

This XML document takes the following form:

```
<ROOT_ELEMENT>
 <ELEMENT_1>
   <ELEMENT_2>DATA_NAME_FOR_ELEMENT_2</ELEMENT>
   <ELEMENT_3>
      <ELEMENT_4>DATA_NAME_FOR_ELEMENT_4</ELEMENT>
      <ELEMENT_5>DATA_NAME_FOR_ELEMENT_5</ELEMENT>
   </ELEMENT_3>
 </ELEMENT_1>
</ROOT_ELEMENT>
```

And from this would generate the following control table:

Option	Key	Value
tag	ELEMENT_2	DATA_NAME_FOR_ELEMENT_2
tag	ELEMENT_4	DATA_NAME_FOR_ELEMENT_4
tag	ELEMENT_5	DATA_NAME_FOR_ELEMENT_5

Note that no tags are created for elements that have no data items. Those tags will be required, but it is expected that they will be generated by the **Create XML Generator Control Table from XSD** component.

Data names may be optionally enclosed within $ symbols.
e.g.

```
<ELEMENT_2>$DATA_NAME_FOR_ELEMENT_2$</ELEMENT>
```

These $ symbols will be removed and play no part in the name of the data item.

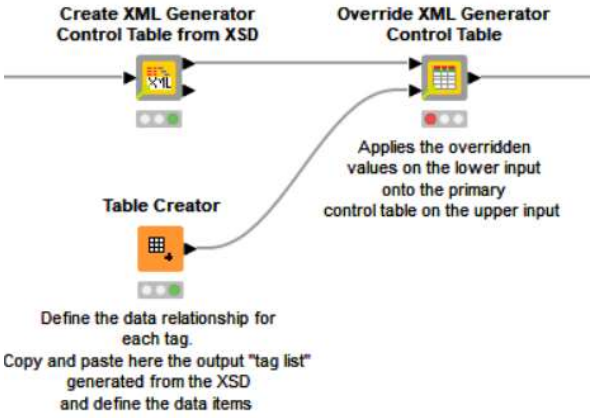
The **Override XML Generator Control Table** is, in reality, a glorified Concatenate Node. It does a bit more than that, but in essence this is what it does.

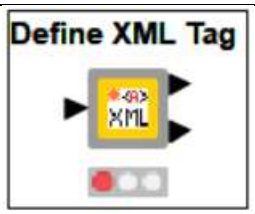
The upper input is the control table that has been generated using one of the Control Table generator or modification nodes.

The lower input is a manually edited control table containing the changes that are required. Typically, such changes are supplied via a manually edited Table Creator and contain the "data mappings" of XML Elements (known as "tags") to their data items.

A "tag" is a term used here to represent a control table "artefact" which represents an XML item to be output: a grouping element, data element or attribute.

The output from this component is simply an updated control table. You will see this component commonly used with the **Create XML Generator Control Table from XSD** component as follows:





The **Define XML Tag** component is used, as you may have guessed to define (create) a new "Tag". A tag is an item that will be used to define an XML Group Element, XML Element or an XML Attribute.

For this component to function, it requires the presence of a flow variable created by the **Create XML Generator Control Table** which lists the different data table columns from the input data source.

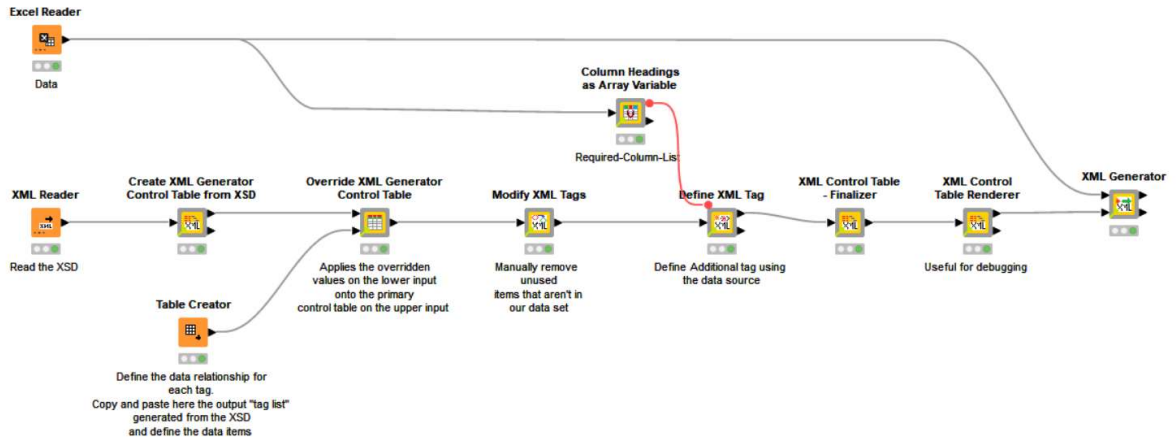
As such, the **Define XML Tag** component cannot be used if the control table has been created just with the **Create XML Generator Control Table from XSD** as its purpose is to allow the manual creation of new tags that are based on data items in a situation where there is no XSD.

This makes sense if the purpose of generating the XML is to match the XSD as in this situation you would not expect to be creating tags for elements that *aren't* in the XSD.

However, if you are perhaps using the XSD merely as a starting point and have additional data items to be added from a separate data table, this can be achieved by manually creating a flow variable named *"**xml-gen-query-col-list**" as a (String Array)* upstream of this component.

This flow variable should contain the names of all the data columns for which you are generating the XML. This could be done also be achieved by using my "**Column Headings to Array Variable"** component available on the KNIME HUB.

Connect the data table to the input data port and connect the output flow variable port to the **Define XML Tag component**. Configure it to create a variable called ***xml-gen-query-col-list*** *and all should be good, as per this example:*

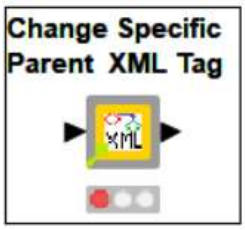


You will notice that there are two output ports on the **Define XML Tag** component. The upper port is the modified Control Table to be passed downstream to the next XML Generation component. The lower port is purely for information (or curiosity) and shows some details of the Control Table entries that have been created by this component. Do not rely on this lower port giving any specific format, as it may change in future, and is really there for debugging purposes.

The **Modify XML Tags** component is used to make a change to one or more existing tags. It is also used to delete tags which may be necessary sometimes if, for example, the Table Creation components automatically create tags from the data source, or XSD, but you don't actually want these to generate elements in the resultant XML.

The **Change Parent XML Tag** component is used to move tags from one "parent" element to another within the XML hierarchy.

This component does not allow you to specify the individual tags to be moved, but instead moves all tags that are currently "children" of one element, so that they become children of a different element. This may be necessary where, for example, the control table has been created based on a data source, but we are manually creating the necessary hierarchy "groupings" and wish to move tags together to a new group in the hierarchy.



The **Change Specific Parent XML Tag** component is used to move a defined tag or set of tags from one "parent" element to another within the XML hierarchy. This gives finer control than the **Change Parent XML Tag** component and will be more likely used once the hierarchy has generally been configured but there are some "fine tuning" adjustments to be done. For bulk moving, though, where you don't want to have to look to find all the "children" of an element, the Change Parent XML Tag can be more convenient.

The **XML Control Table - Finalizer** component is the component to be placed just before the **XML Generator** component. When generating the Control Table, and performing the various configuration steps, there are features of the control table that need to be set in a certain way or else the generator doesn't give the expected results.

When developing the components, I commonly found that a few things would not be "quite right" when it came to the XML generation, and I eventually realised that it would be better to have a "finalizer" that checked for common issues and corrected them rather than try to explain all the inner workings, and waste time trying to remember them.

So, if you find the xml generator isn't outputting correctly, try adding the Finalizer.

It is recommended anyway that the finalizer always be present, as over time, it might add additional tweaks to ensure newly functionality works with older control tables, should changes be necessary.

The finalizer also does a little housekeeping such as removing row duplication in the control table that may have occurred through manual editing or concatenation.

The **XML Control Table Renderer** component is not strictly a necessary component for generating XML. It does not make changes to the control table and its presence in a workflow will not affect XML generation.

What it does though is assist with manual debugging of the control table.

The input port is the control table, and the upper output port is the same control table passed through unchanged, so you can leave it attached in a flow without it breaking anything.

The lower data port, though, is there for debugging and is a "hierarchy" generated from the control table to show how the XML generator will "perceive" the XML to be generated. If you view the output of the second data port, you might, for example, notice that elements are missing, and you can resolve many issues quickly without running the full generation. This lower port is purely for manual "observation", and you should not assume it is in any particular format for processing, as it may be updated in future versions.

The **Set Group Parent Identifier XML Tag** node allows you to specify the identifying tag of a group tag's parent. This allows repetition of items within the group until such time as the group's parent changes. This component generates "breakonchange" tags.

So that ends the quick tour of the components and hopefully gives a flavour for how they interact and their purpose.

Some of the components have configuration options, but many do not. Please read the component help built into each component for assistance with configuration.

The primary configuration for the XML generation as a whole though is the “Control Table”, and this is discussed in the next section.

The XML Control Table

The configuration of the XML Generator is held in what is called “The XML Control Table”. This is effectively a registry of the properties that define the XML to be generated. Every line consists of 3 active properties, “Option”, “Key” and “Value”. There may be other columns to the right, such as “Comment” and “Additional Notes” but these are there purely for information purposes and are not processed by the XML Generator or other components.

The Control table is effectively the “language” of the XML Generator. The “Option” is similar to a command, or instruction. They Key is typically the name of a tag, and a Data Item name, another tag name or a value to be used by the Generator to fulfil the stated instruction.

e.g. This is the first few lines of the control table

OPTION	KEY	VALUE	COMMENT
header	Root	SURVEY_RESULTS	specify the root data element *Only the Value should be changed*
header	Row	record	specify the element name at row level *This should not be changed*
header	RowGroup	row_group	specifies the tag name for the Row *This should not be changed*
display	row_group	PERSON	The display name for each “row” *Only the value should be changed*
parent	row_group	record	The top level parent *This should not be changed*
tag_type	row_group	group	defines top level grouping *This should not be changed*

The “header” options are not generally used other than as the first three instructions and they are created by the Control Table Generator. They define internal information used by the Generator.

These options define the “Root”, the “Row” and the “RowGroup” which are special items required internally for the generator to operate, it requires that XML follow a basic hierarchical structure with a single “root” node, and then have each “row” or “record” contained within a “grouping” node. This grouping node is defined as the “row_group” tag.

Special “internal” values in the Control Table

option	key	Notes on usage
header	**Root**	The **value** of this defines the name of the “root tag” which will define the first XML element to be output. This is user-changeable.
header	**Row**	This is always set to “record” and should not be changed
header	**RowGroup**	This is always set to “row_group” and should not be changed. “row_group” is the internal name for a tag that groups every “row” that is to be output in the XML.
display	row_group	The **value** of this line defines the XML ELEMENT name that will be output for the row_group. This value is user-changeable.
parent	row_group	This is always set to “record” and should not be changed. This tells the generator that the “row_group” tag is a child of the internal “record”.
tag_type	row_group	This is always set to “group” and should not be changed. It tells the generator that row_group is an XML Grouping element

All other values are configurable by the user. All option and key names are case-sensitive.

Primary Control Table Options

option	key	Notes on usage
tag	**tagname**	The **value** of this defines the case-sensitive name of the data item (column name) in the input data that will be used to provide a value for the XML element or attribute when it is output. If no data item is associated with the tag, it should be given a value beginning # which will then be ignored by the generator. The tagname given here must be unique. It will be used as the internal name for the tag and may be referenced by other options in the Control file. If you have multiple XML elements that require the same output name, ensure they each have different (unique) tagnames, but give them the same “display” value.
display	**tagname**	The **value** of this defines the name given to the element or attribute when it is output in the xml. So if an element tag is given a display value of PERSON, the resultant element would be output as <PERSON> This is case sensitive, and you must ensure the value given conforms to XML naming standards for elements or attributes.
tag_type	**tagname**	This takes one of the following values which defines the type of output produced: element – the resultant output will be an XML Element, e.g. <PERSON> group – the resultant output will be an XML Element that groups one or more other XML Elements attribute – the resultant output will be an attribute on the parent XML Element
parent	**tagname**	The value given will be the name of another tag which will be defined as the immediate ancestor of this tag when the XML is generated.

Additional Control Table Options

The remaining options/instructions are not always necessary and are designed to change the behaviour of a given tag during generation

option	key	Notes on usage
breakonchange	**tagname**	The purpose of this option is to define the circumstances under which a "group" element should "close" in the xml and then "reopen" as a new group. The value is the name of another tag that is to be monitored for a change of value. When the value changes, the grouping element will close and reopen as a new group. Typically the identifying element is the ID of the "parent" of this group. So in a list of employees, it might be the ID of the department so that then all employees in the same department will be listed together. For this to work, the data set must be sorted prior to calling the XML Generator so that "grouped rows" occur contiguously.
ifblank	**tagname**	The value to be output if the value of the tag is blank.
ifmissing	**tagname**	The value to be output if the value of the tag is missing.
ifzero	**tagname**	The value to be output if the value of the tag is zero.
rowcounter	**tagname**	**Not yet implemented**. May be implemented in future. This would generate a row counter for each item output. A close approximation to this functionality could be achieved in KNIME, add a row counter into the source data and creating a tag for it in the Control Table.
numeric	**tagname**	If Y, defines that the resultant output is a numeric
integer	**tagname**	If Y, defines that the resultant output is an integer
hideifmissing	**tagname**	If set to Y, this suppresses output of the element or attribute in the XML if the value is missing
hideifblank	**tagname**	If set to Y, this suppresses output of the element or attribute in the XML if the value is blank
comment	**tagname**	Supply a value of "#". This makes the tag into a "comment". This will generate a comment in the XML which does not represent any data value to be processed. Useful for debugging, such as inserting source ID values into the resultant data A. To include the value of one or more other tags in the comment, place the tagname(s) in curly braces: e.g This is the current value of person_id: {person_id}
literal	**tagname**	**Not yet implemented**. May be implemented in future Makes the tag a "literal" value, so the same value will be output on every occurrence. The "value" to be entered for this option is the value that is to be output. If this functionality is required, in KNIME, add a literal value to the input data source.

Note that both the option names and tag names are **case-sensitive**

Output order of elements

When the XML Generator generates the XML, it navigates the **Root** tag, then the **row_group** tag, and then traverses the "parent" hierarchy in the order in which tags are added to their parents. Therefore, the output order of the XML elements is defined by the order in which the "parent" options appear in the control file. To make a element A appear before element B, ensure that in the Control Table, that element A's parent tag appears before element B's parent tag.

Comments and Feedback

This "toolkit" of components for XML Generation is a "work in progress", and as such please fully test that it works for you. I am aware that there is some functional improvements that could be made over time, and I cannot be certain that it will work for all situations as whilst there are a wide variety of scenarios, this generator cannot necessarily deal with all of them.

Feedback and suggestions are welcome. If you find that it doesn't work for your scenario, then please contact me @takbb on the Hub section of the KNIME community forum.

I would also be pleased to hear of any successes with using it, as it is real-world scenarios that motivate me to continue working on it.

I would also greatly appreciate it if you were able to publish sample workflows using these components as this will both take the burden off me for providing demonstration workflows, and potentially assist others with theirs.

Thank you!
takbb

November 2022