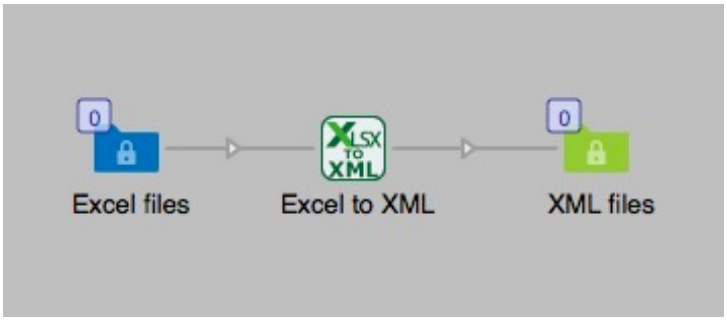
Excel to XML v4

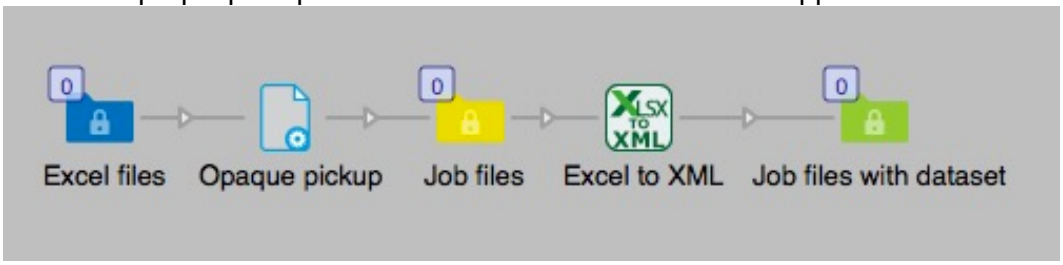
Description
Excel to XML will let you submit an Excel file in the format .xlsx to a Switch flow were it will be converted to XML and/or metadata sets. It will accept Excel files with multiple sheets. You have different options of output either as XML-files or to pickup the Excel data and embed as a dataset in the same way as the XML-pickup element works.

Version adds two Private Data sets



To just output XML-files that later can be imported in to InDesign for automatic production of documents from the XML-data works without the Metadata module. This works in a simple situation where you just import the XML-file to InDesign as long as you don't need any information in the XML for controlling InDesign.

You can also pickup the data in the Excel document and embed it in a job file that you send in alongside with the Excel document. In this case you need the Metadata module and the element Opaque pickup in the flow before the Excel to XML app.



Compatibility
Switch 13 update 1 and higher. Windows or Mac OSX.

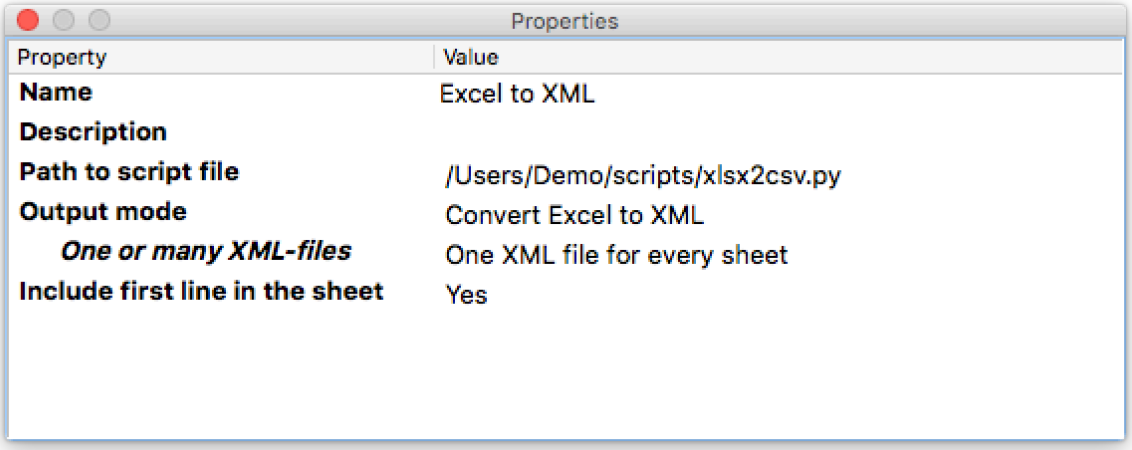
Compatibility third-party applications
This app uses Python to process scripts, for Windows Python is included in the app and for Mac OSX it uses the Python that is in Mac OSX.
You don't need to install any other applications for this app but the Python script itself needs to be downloaded and stored on your system. The conversion from Excel xlsx to csv is performed by this open source Python script. Due to licensing you have to download that script yourself and put in a folder of your choice. Then in the properties you can select this script file. Without it the app will not work.
It is tested and runs with Python version 2.7x.

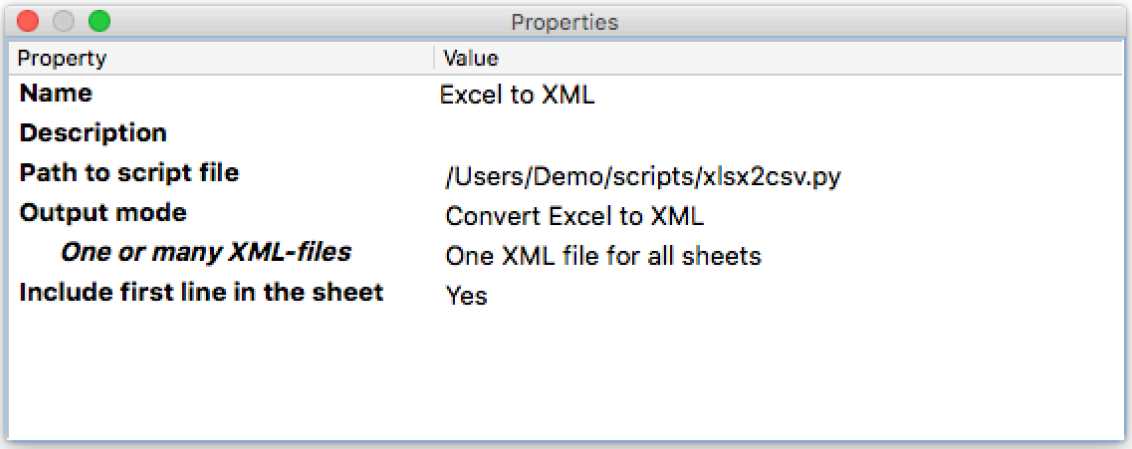
Connections
Excel to XML can have several input connections but there is only one outgoing connection. No settings are available of the outgoing connection.

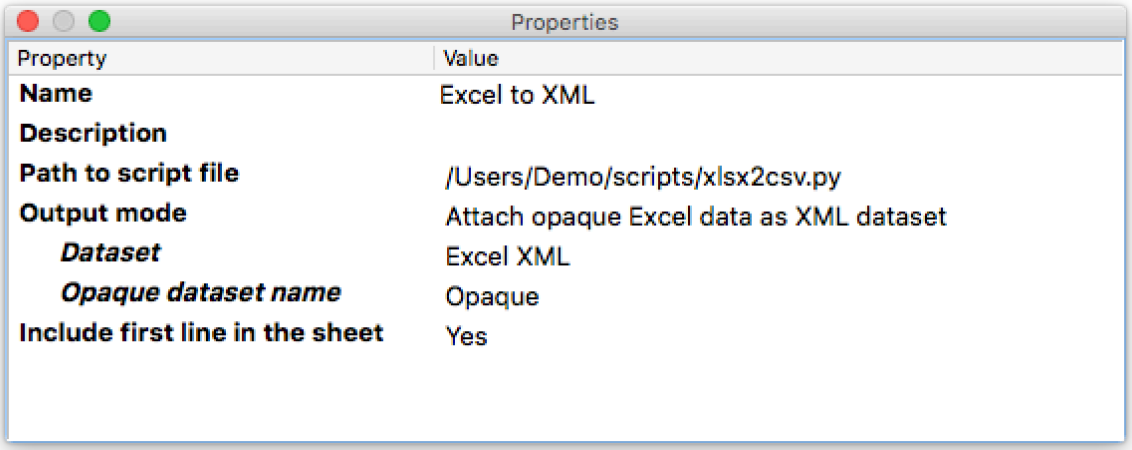
Properties detailed info
If you are using this app to just output XML files you just have to set the properties in the way you like. If you want to send Excel files along job files you need the element “Opaque pickup” that are included in the Metadata module. The reason for the “Opaque pickup” is to be able to pair a job file with the data in the Excel file.

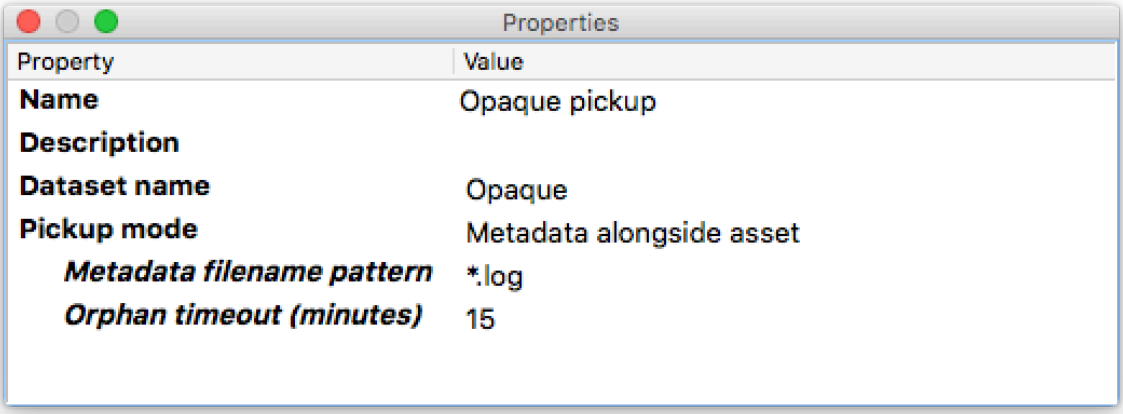
Flow element properties

- Path to script file
  - Select the Python script file named “xlsx2csv.py”
- Output mode
  - Convert Excel to XML.
  - Attach opaque Excel data as XML dataset.
- Convert Excel to XML
  - One XML file for every sheet.
  - One XML file for all sheets.
- Attach opaque Excel data as XML dataset
  - Dataset, the name of the dataset that you can use in variables in Switch.
  - Opaque dataset name, it is important that this property has the same name as the setting in the Opaque pickup elements “dataset name” property. If these two are not the same the job will fail.
- Include first line in the sheet
  - Yes, in this case the value of each column header will be used as a node tag in the XML, if the column header is “first_name” it will be like this: <col name=”first_name”>John</col>
  - No, then the first line is not a header and the values of the first line will be treated as all other rows in the Excel file.









In the Opaque pickup element you have to set the properties as in the image above. Pickup mode must be “Metadata alongside asset”. And the “Metadata filename pattern” must be set to *.xlsx. In this property pane you can set the Dataset name for the Opaque pickup. It must be exactly the same here as you set in the Excel to XML property “Opaque dataset name”. If not the job will fail.

Extra information
If you use this app to output multiple XML files for later import in to InDesign for automatic production of documents you will need to adapt the XML to something useful for InDesign. To do that you have to use the Saxonica configurator and an XSLT-file. This will give you the possibility to produce business cards, tickets or product labels very quickly. You can do this without the Switch Metadata module.
To better process the resulting PDF-files that you have made with the XML-files from the app there is now two Private data that will help you.

- Private data key: ***SheetId***
    - This key will give you the name of the Excel sheet where the XML-data comes from.
- Private data key: ***NumberOfRecords***
    - This key will give you the number of records that are picked up from the Excel sheet.

When you later in the flow will need to assemble the produced single PDF-files from each record these two Private data keys are needed.
In the Assemble job properties you will use the scheme “Custom” and then use the SheetId key as a job identifier and the property “Number of files” from the key “NumberOfRecords”

The XML structure for the multiple XML-files will have the X-path structured per each row and each column as follows: /csv/row/col
Here is an example of an XSLT-file that can be used with the XML-files you get from the Excel to XML app. Each XML-file will have the name of the Excel workbook sheet.

Note
Be aware that this might lead to overwriting XML-files where the default sheet name is used.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">
    <xsl:output method="xml" indent="yes"/>
    <xsl:template match="/">
        <xsl:apply-templates/>
    </xsl:template>
    <!--Splits the XML-file generated in to single XML-files, one for each row-->
    <xsl:template match="/csv">
        <xsl:for-each select="row">
        <!-- Selects which column value to use as filename for the resulting single XML-file. -->
        <!-- In this case column 2 is used. -->
            <xsl:variable name="filename"><xsl:value-of select="col[2]"/>.dita </xsl:variable>
            <xsl:result-document href="{col[2]}.xml" method="xml">
                <excel-row>
                <!--List the columns from the Excel-file in the order they will have in the resulting XML-file.-->
                <!--The name can be set to anything you like, for example the header used in Excel.-->
                    <col name="Column 1">
                        <xsl:value-of select="col[3]"/>
                    </col>
                    <col name="Column 2">
                        <xsl:value-of select="col[4]"/>
                    </col>
                    <col name="Column 3">
                        <xsl:value-of select="col[5]"/>
                    </col>
                    <col name="Column 4">
                        <xsl:value-of select="col[6]"/>
                    </col>
                    <col name="Column 5">
                        <xsl:value-of select="col[2]"/>
                    </col>
                </excel-row>
            </xsl:result-document>
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

If you choose to attach the Excel data to a job file as a dataset the XML structure will be as follows: /workbook/sheet/csv/row/col were each sheet node will have the name of the sheet.

Third party information
This app uses a Python script that is published under GPL2 licenses. The script can be downloaded from here: https://github.com/dilshod/xlsx2csv and it comes as a package, the only file you need is the ***xlsx2csv.py*** that you place in a folder of your choice on the same system as your Switch server. In the properties you then select this file for the app to work.

The app requires Python 2.7 to run the above mentioned script, for Windows it is embedded in the app, in Mac OSX Python 2.7 is already installed in the system. Python 2.7 license can be read here.

New in version 2
Python script is no longer embedded due to licensing issues.

Better XML if first line is header.

New in version 3
App can now handle a one column CSV-file.

New in version 4
Two options to use Private data.