Four Pees

Switch

App

FourPees

Create file from template

Four Pees nv

Kleemburg 1
9050 Gentbrugge
Belgium
p +32 9 237 10 00
f +32 9 237 10 01
info@fourpees.com
www.fourpees.com

Table of Contents

1. Introduction

What if you want to create a relatively complex text file in Switch? Say... something like an XML file, or a JDF, or a complex JSON file. Yes, you can write a custom script for this, but even with a script, doing this is still very messy and a lot of hard and painstaking work. If only there would be a way to make that easier...

Wait a minute, when you're generating HTML files for a web site, isn't that very much the same problem? How do those systems deal with that? Turns out that smart people invented something called a templating engine. You provide a template and can then insert variables in that template in a structured way. Modern templating engines even allow you to create loops, insert conditionals and so on.

This Switch app uses a templating engine – Embedded JavaScript templating, or EJS – to do exactly this. The app has different ways to specify where the template is and to specify input variables for the template. It builds the output from the template injected with these variables and then outputs it according to what you need in the rest of the workflow. The rest of this document describes how to use the app and explains the different properties.

1.1. Four Pees ~ feel the good flow

Four Pees was founded in 2007 in the bustling university city of Ghent in Belgium - one of the leading innovation hubs in Europe, specifically for anything print related. Our goal: automation that flows smoothly. Automation that not only makes work easier but also helps companies thrive. In short: **hassle-free solutions to headache-inducing operational challenges**.

Now more than a decade later we provide solutions that streamline entire print and packaging productions all across the globe and we have sales operations in Belgium, The Netherlands, The United Kingdom and France. We also rely on a network of 40+ partners worldwide to give you the best service possible.

With our wide range of customizable automation solutions, we allow you to focus on what you do the best: design, create and produce. **We persist where others give up**. Our team loves a good challenge and will never stop looking for a solution. Let us take care of the technical kinks, all you need to do is...feel the good flow.

1.2. Getting hold of us

Does this app not live up to your expectations? Do you need something slightly different, or do you have an automation project you want to discuss with us? Please get in touch with us! Send an email to info@fourpees.com and we can have a conversation.

For technical questions about this script, please use the support@fourpees.com email address. You'll get a confirmation message and we'll be with you before you can say "Automation".

For any other questions, you can simply use the same email address (we talk to each other internally so we'll find the right person to help you! Or see all contact possibilities you have here: https://www.fourpees.com/en/contact.

1.3. Versions

The following is a short version overview:

- version 1: initial version of the app.

2. How to use

This app is quite flexible in how it can be used in a Switch workflow. Of course, it's goal in life is to create a text file based on a template and variables, but it still allows quite a bit of flexibility in how it does that. This chapter shows three possible way of using it so you can find some ideas. After that, it's up to your own flexibility (and of course we can help with that!).

2.1. Inserting a template result into the flow

Perhaps the most straightforward way of using the app, is by inserting it into a workflow as shown here. In this example:

- A JSON file containing order information enters the workflow and hits the app. The app property that specifies where the variables are says "Incoming job".
- The template is read from a fixed file using the template property of the app.
- And the result property is set to "Send to outgoing connection".

What is important of course is that the incoming JSON file stops at the app and is then discarded. The output of the app is the text file with the email template and inserted variables.

2.2. Using a dataset

What if you want to just let your incoming job pass through the app? That can be done by using a dataset. In the above example:

- It's still a JSON file that enters the workflow and hits the app. So the variables property is again "Incoming job".
- The template is also still a fixed file.
- But now the result property of the app is set to "Attach as dataset".

In this case, the output of the app is the original job – which is unaltered – except that it now has a new dataset attached to it. If for example you'd want to send an email later in the flow, this could be ideal as the "Mail send" tool allows pickup up the mail content from a dataset.

When you're attaching the dataset, you can also specify what type of dataset it should be. The app supports XML and JDF datasets; anything that is not one of these two formats should be set to "Opaque".

2.3. Writing to a fixed file

The third example is again quite different, and it could for example be used to write log information into a CSV file. In this case, a regular job enters the flow and hits the app, and that job is also sent to the output folder of the app (in the screen grab, the “To next task” folder).

But now the app writes the result of the template completion to a fixed file. The app lets you specify both the folder and the file name for this file, so you could easily write a new file every day. And as the app lets you choose between overwriting an existing file or appending to the end of it, both writing a one of file and adding to an existing log file is completely supported.

If you download these example flows from the page on the Enfocus AppStore for this app, you’ll see that the CSV template can even be created in such a way that you can write out a file with a header. The app provides additional variables so your template can know whether it’s the first time you write to this particular log (and then it can add the header), or whether this will be appending (in which case it skips writing the header).

3. Template engine

A template engine is a text processing engine that ingests a template (a text file) and outputs a new text file. The template offers to capability to do text replacement, ranging from simple substitutions of a single variable, over conditional processing to looping. While text replacement engines are mostly used for web development, and the creation of HTML pages, they can easily be used for any text-based file format.

This app uses the EJS template engine. Some usage examples are included below, but much more documentation can be found here: https://ejs.co. If you create the templates using Visual Studio Code, there are a number of Visual Studio Code plug-ins to help you, ranging from syntax coloring to quickly creating the substitution codes you'll need. Remark that EJS uses JavaScript as part of its template syntax, which makes it rather elegant to use. And no, for simple substitutions, you really don't need to speak JavaScript.

3.1. Simple variable replacement

The simplest use case is to have a template with some variables, that are then replaced. This would allow you for example to write something like:

```
My favorite animal is a <%- favoriteAnimal %>.
```

If you feed this template variables, and among the variables there is one called `favoriteAnimal`, whatever the value of that variable is would be inserted into the output string.

```
My favorite animal is a penguin.
```

You could make the output into HTML by using HTML tags in your template around the text:

```
<h2>My favorite animal is a <%- favoriteAnimal %>.</h2>
```

Or if you prefer your files in XML format:

```
<messages>
      <message>My favorite animal is a <%- favoriteAnimal %>.</message>
</messages>
```

Remark that everything in your template will remain untouched, except for the EJS tags (which normally start with "`<%`"). And while these examples show HTML and XML, you can of course do the same in a MarkDown file, a CSV, a JSON…

3.2. Conditional processing

You can use the incoming variables to "sometimes" output part of a template, again based on the variables you feed your template:

```
<% if (favoriteAnimal) { %>
      My favorite animal is a <%- favoriteAnimal %>.
<% } else { %>
      I don't have a favorite animal.
<% } %>
```

The result of this template (given that there is a correctly named variable) will be a single line of text. Which of the two lines is output depends on the result of the "`if`" condition.

3.3. Outputting repeating information

The biggest challenge for a simple text creation app is information that repeats, and especially so if you don't know ahead of time how much data there will be. With a template this can be handled quite nicely:

```
<% items.forEach(item => { %>
      Ordered product:
      <%-item.name-%>
      Quantity: <%-item.quantity-%>, total price: <%-item.price-%>€
<% }) %>
```

The template snippet above assumes our variables include "items". Using standard JavaScript, the template outputs information for each object in the items list.

3.4. Injected variables

Any variables that you define in a JSON file or using the key / value mechanism in the app properties, will be available for use in your template. For example, if your JSON looks as follows:

```
{ name: "Pingo" }
```

In your template you will be able to use the variable "`name`" and it will be replaced by its value. The app tries to make this process a bit easier for you, by also injecting job and switch related information automatically, without you having to do anything.

3.4.1. Standard variables

By default, the app injects an object called "`fourpees`" into your variable set. Even if you would pass an empty JSON file or no key / value pairs, the "`fourpees`" object would still be available. In this object the following properties are defined:

- `info.id`: the unique ID for the job
- `info.name`: the full name of the job
- `info.nameproper`: the name of the job without extension
- `info.extension`: the extension of the job
- `info.isfile`: `true` if the job is a single file, `false` otherwise
- `info.isfolder`: `true` if the job is a job folder, `false` otherwise
- `privatedata`: an object containing a key for each private data field associated with the job and its value.
- `app.appendingtoexistingfile`: either `true` or `false`, see the "New file" section below.

In the template, you can use this information just as you would use other variables you inject in your JSON object. For example:

```
<%-fourpees.info.name-%>
```

Is going to be replaced by the name of the job (including extension). Likewise:

```
<%-fourpees.privatedata.fpNumberOfPenguins-%>
```

Is going to be replaced with the private data field called "`fpNumberOfPenguins`" as defined for the incoming job.

3.4.2. New file or appending?

In some cases, you might want to have a different template output depending on whether this is the first time you're writing to the output file or you're simply appending to an already existing file. A good example would be a CSV file with headers. When creating the CSV file, you want to write both the headers and the incoming data row. When appending to the existing CSV file, you only want to write the incoming data row.

The "`app.appendingtoexistingfile`" variable is meant exactly for this. It will be set to "`false`" in all case, except when the properties of the app ask to append to a file (rather than overwriting it), and the file actually already exists. If you want to use the app to write a log file for example, this could be done like this:

```
<% if (fourpees.app.appendingtoexistingfile === false) { %>
      // Write headers
<% } %>
// Write data
```

The template can in this way do anything that needs to happen only when the output is freshly created.

3.5. Examples

On the Enfocus AppStore you can download an archive with small examples of how to use this app in actual workflows. Go to the page for this app and look at the bottom (next to the link to the documentation file you're reading now).

3.6. Need help?

For simple purposes templates are not that difficult to create. If your project is more complex, so might the corresponding template be. Don't hesitate to get in contact with us (using info@fourpees.com) if you want to see whether we can help you to complete your project.

4. Properties

The following properties can be used to change the behavior of the app.

4.1. Template

In order to function, the app needs an EJS compatible template, which is specified using the "**Template**" property. The following possibilities are supported:

- **Incoming job**: the EJS template is the incoming job. In this case the option "**Incoming job**" cannot also be selected for the "**Variables**" property (the incoming job can't be both an EJS template and a JSON with variables).
- **File**: the EJS template is specified in an external file. In this case an additional option "**File**" allows specifying the location of the template file on your system.
- **Dataset**: the EJS template is attached to the incoming job as a dataset. An additional "**Dataset**" property now allows setting the name of the used dataset.

4.2. Variables

- **Incoming job**: the variables are specified as a JSON file which is the incoming job. In this case the option "**Incoming job**" cannot also be selected for the "**Template**" property (the incoming job can't be both an EJS template and a JSON with variables).
- **File**: the variables are specified as a JSON file. In this case an additional option "**File**" allows specifying the location of the JSON file on your system.
- **Dataset**: the variables are specified as a JSON file which is attached to the incoming job as a dataset. An additional "**Dataset**" property now allows setting the name of the used dataset.
- **Text**: sometimes your variable needs are simple. In that case you can use this option. It allows you to specify an unlimited number of variables and their values in a simple key / value format. Select this option and use the additional "**Text**" property. It contains an example of how to specify the variables in the proper format.
- **JSON as text**: if simple key / value pairs don't cut it, but your needs are still simple enough, you can also specify a JSON object directly in the "**JSON as text**" property of the script. You can use Switch variables in your JSON definition of course.

4.3. Result

This app injects the specified parameters into the template used. The result of this is a "blob" of text created by the EJS template engine. The "Result" properties decide how this result is going to be used. It supports the following values:

- **Send to outgoing connection**: the template result is written to a file, which is then sent to the outgoing connection of the script. The original job is discarded. When this is selected, additional properties appear:
    - **File name**: the full name (including extension) of the output file
- **Write to file**: the template result is written to the specified file. If there is an outgoing connection, the original job is sent to that outgoing connection. If not, the original job is discarded. When this is selected, additional properties appear:
    - **Folder**: the folder to which to write the result file. This folder must exist.
    - **File name**: the full name (including extension) of the output file
    - **Mode**: "Overwrite" if any file with the same name is going to be overwritten, or "Append" if the app will append to an existing file (if it exists).
- **Attach as dataset**: the template result is attached as a new dataset to the original job, which is then sent to the outgoing connection. When this is selected, additional properties appear:
    - **Dataset**: the name of the dataset the result will be written to.
    - **Type**: the type of the dataset you want to attach.

5. Tips, tricks & pitfalls

This chapter describes more advanced topics, limits and implementation limits, tips and tricks.

5.1. Text encoding

The text file that is created always uses UTF-8 (Unicode) encoding. As the text you would input in Switch normally also will be UTF-8, this shouldn't pose any issues. But knowing the encoding of the created text file may be important to determine how you're going to handle this file further down in the Switch flow. You'll have to make sure it's interpreted as Unicode at that point as well.

5.2. What does "text file" mean?

In the context of this app, "text file" can be interpreted as ***any file that is written in a textual format*** (as opposed to a binary file). This means that this app could just as well be used to write a CSV file, JSON, XML, RTF or any other text-encoded file. But the app has no built-in mechanism to make your life easier if you decide to write these files – you'll have to make sure you follow the proper rules.

5.3. Separating logic and view

While reading about this app, you could consider that it might be much easier to simply write a custom script that outputs the file content you need. This is not a wrong thought, but it really depends on what you are doing.

Writing a custom script that outputs a complex XML or JDF for example quickly becomes very tedious, and a big part of that is because you mix the workflow logic and the view, or representation, of the data that you want. If you can have a custom script or workflow that generates a JSON object (which is easy in JavaScript) and then use this app to do the conversion of that JSON data to the actual XML or JDF file you want, that will simplify the logic in both places. Your script can now focus on the logic, which will make it smaller and easier to maintain, and the template you create in this app can focus on what exactly you want to output.

The question is never, can I do this with a custom script. The real question is: "what is the easiest way to do this". Now, and six months from now when the customer (which may be you yourself) comes back with their feature requests.