# Apache NiFi Flow Manager

**De macro Ctrl+T zet de taal op Engels.**
**De macro Ctr+L maakt een streepjes lijst.**
**De macro Ctr+G maakt een grid/tabel.**

**Content**

## 1. Introduction

According **Apache NiFi** own website, http://nifi.apache.org/docs.html:
*"**NiFi** was built to automate the flow of data between systems. While the term 'dataflow' is used in a variety of contexts, here it means the automated and managed flow of information between systems"*

The marketing wording (http://nifi.apache.org/index.html) is as follows:

*"An easy to use, powerful, and reliable system to process and distribute data"*

**NiFi** is an acronym, coming from the '*NiagaraFiles*' software project. It is pronounced as "nye fye" (nī fī).

It is not an **ETL** (**E**xtract **T**ransform **L**oad) tool. It listens for (new) data and process accordingly and saves it at the destination system, as long as **NiFi** has support for it. For instance, you read csv data from an **SFTP server**, store it locally, do something with it, and saves it at the other system.

In my case, the target system is MarkLogic. **NiFi** has no built-in support for it. You first install a plug-in and then **NiFi** can store the data at MarkLogic. But data manipulation at MarkLogic has to be done in MarkLogic itself. For that they have a "data-hub" application, their ETL tool.

## 2. Managing data flows

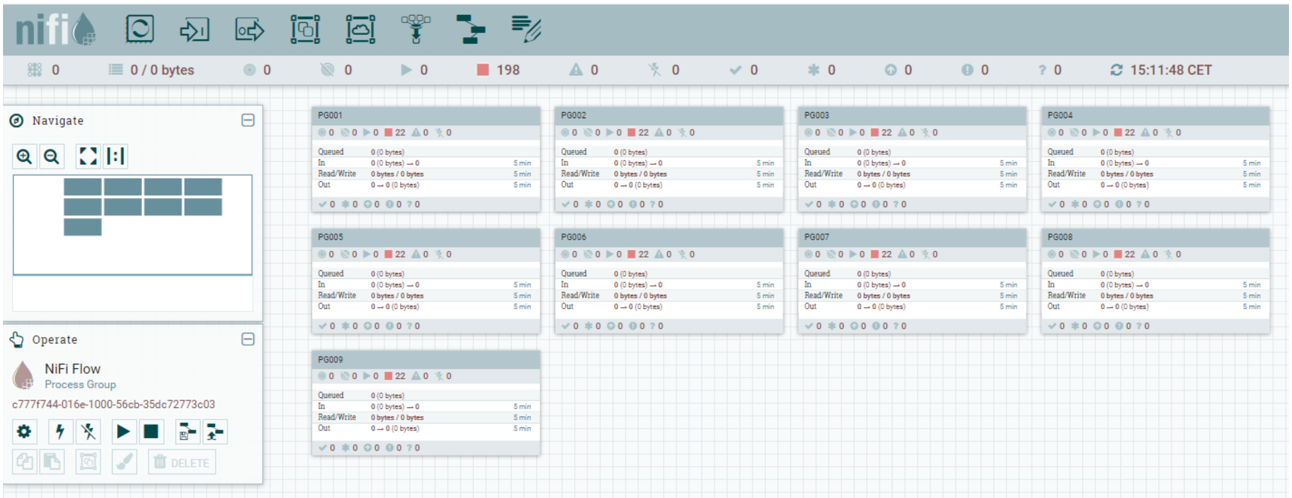
**NiFi** has one canvas which uses instant auto-saving when you edit a dataflow. Everything is recorded in one compressed file: **flow.xml.gz** saved at one particular location. When you have 100 dataflows, it is all in one file. To manage all the **NiFi** flows, an approach is to develop, test and maintain each dataflow as a separate flow, and then later on, merge all the separate flows together in one flow, the *merged flow*, which becomes your production flow. This can be done as long as each flow is encapsulated in a so-called *process group*. *Process groups* can be nested, but as long as a flow has only one main *process group*, you can extract the *process group* from the flow, the *data flow*, and merge them all together into the *merged flow*.

You need one flow, called the *common flow*, where you put all the extracted flow data in. The *common flow* must **not** contain a *process group*. The *common flow* contains all the shared parameters (which can be encrypted, passwords etc.) that can be used be the *data flows*. Thus, it is an empty flow, which is then populated with extracts from the *data flows*, which become the *merged flow*. The *data flow* should be developed as a fully self-supporting stand-alone flow. When you use parameters, you have to maintain them also in the *common flow*. The *common flow* is not overwritten. The *merged flow* is, but you can make new ones.

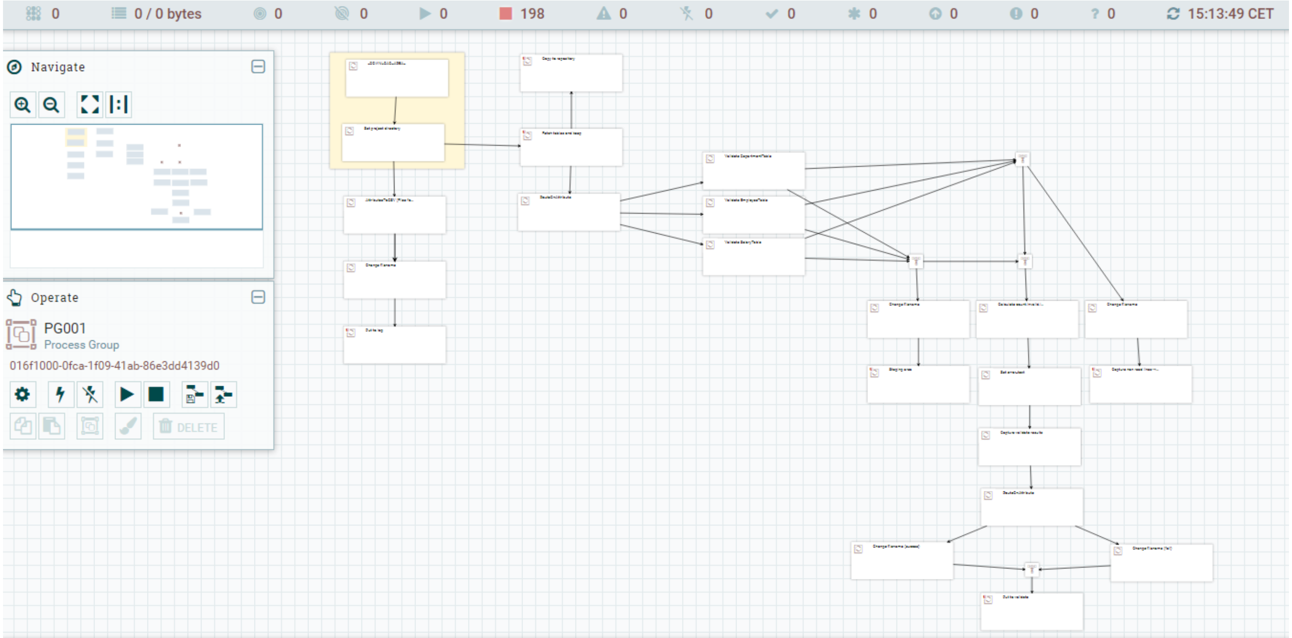
The ***NiFi Flow Manager*** application automates this approach. It is a fully file-driven approach. It operates on the file system. When you have 100 flows, you have 100 **flow.xml.gz** files, all stored at a different subdirectory. When you want to edit a flow, you copy the flow to the *config directory* where **NiFi** read the flows. When you are done you copy back the flow to the subdirectory. Once the *merged flow* is created, you can edit that one to.

This means that you can use a version issue system, for instance GitHub or SourceTree. Al the flows are stored in a central repository, and you have a local copy of it. Many people can work on the same flow.

Next screen-shot shows what is meant by merging. Here you have nine flows. The objects are lined automatically, once merged. Here it is 4 column grid, but that it adjustable.



In this case all flows are the same, and look like this:



It is not so relevant what this is doing, but here you see a draw-back of **`NiFi`**; You need wide monitor screens to show a significant readable part of the complete flow. The above does not tell you much. You have to zoom-in to get a readable canvas. If you develop at a 14inch laptop, you are constantly moving the flow at the canvas to get the processor (that is the name of a block or box) in focus. The GUI does not support scroll-bars. It is very easy to accidently moving boxes and lines etc. There is no Ctrl+Z function to undo that. Also, automatic lining/formatting does not exist. These draw-back are minor things. You get used to it.

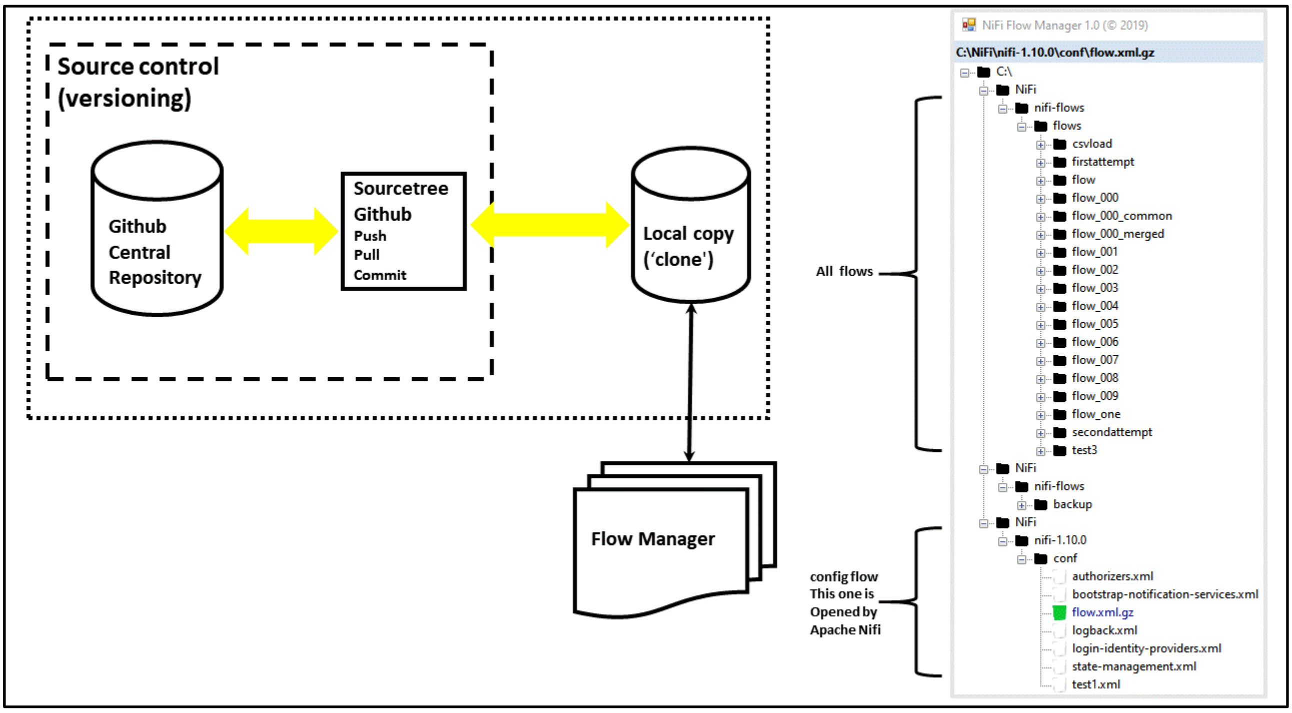
**Note:** When you copy a flow and want to work with the copied flow, you have to first copy the *process group* to a new one and delete the original *process group*. This is needed because every processor/parameter must have a unique processor id. The copying is doing that for you. When a flow has objects (processors) with the same id, Apache **`NiFi`** won't run. In your data flow, containing one flow, this is no issue, but when you are going to merge, and want to run, then you get problems. Thus, when you have copied a flow, running it in **`NiFi`**, then first copy the *process group*, and delete the original process-group, and then start working on the new *data flow*.

**Note:** **`NiFi`** support also flow variables, sort of constants. Don't use them. The official documentation status you have to use parameters instead.

## 3. Architecture

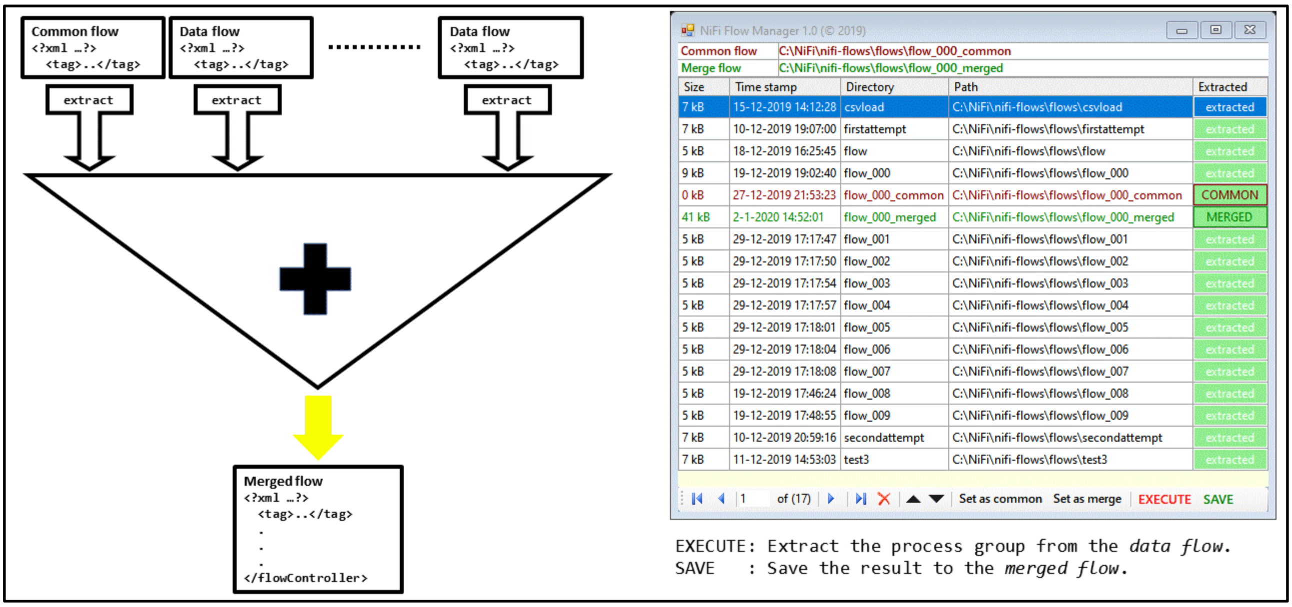
A common development approach is to develop and test, then do acceptance testing, then put into production. In the case of **NiFi** you can argue what do you want to test at acceptance, the whole *merged flow*? Or the *data flow*?

The picture below shows the development architecture. Any acceptance/production environment looks the same, at a different file system. At the right you see the tree view that list the file system. The bottom list the path to the conf(ig) directory, that is where **Apache NiFi** installs itself, which is installation dependent.



```
C:\NiFi\nifi-1.10.0\conf\flow.xml.gz
C:\
  NiFi
    nifi-flows
      flows
        csvload
        firstattempt
        flow
        flow_000
        flow_000_common
        flow_000_merged
        flow_001
        flow_002
        flow_003
        flow_004
        flow_005
        flow_006
        flow_007
        flow_008
        flow_009
        flow_one
        secondattempt
        test3
  NiFi
    nifi-flows
      backup
  NiFi
    nifi-1.10.0
      conf
        authorizers.xml
        bootstrap-notification-services.xml
        flow.xml.gz
        logback.xml
        login-identity-providers.xml
        state-management.xml
        test1.xml
```

Below you see how the merging works. You have selected the flows that are part of the merging, in this case 17. But you can remove flows, and add flows etc. The *common flow* and *merged flow* are mandatory. You can select the merge order. The button tells you the status of the merge. Clicking on it result in viewing the original *data flow* and what is extracted. The merged flow has no original data. You select directories. When there is no *data flow* in that directory, it is skipped. When the buttons are **green**, all fine. Otherwise **red**. This is an extraction status. The *merged flow* might not run due to duplicate processer id, or *data flow* parameters not passed through to the common flow.

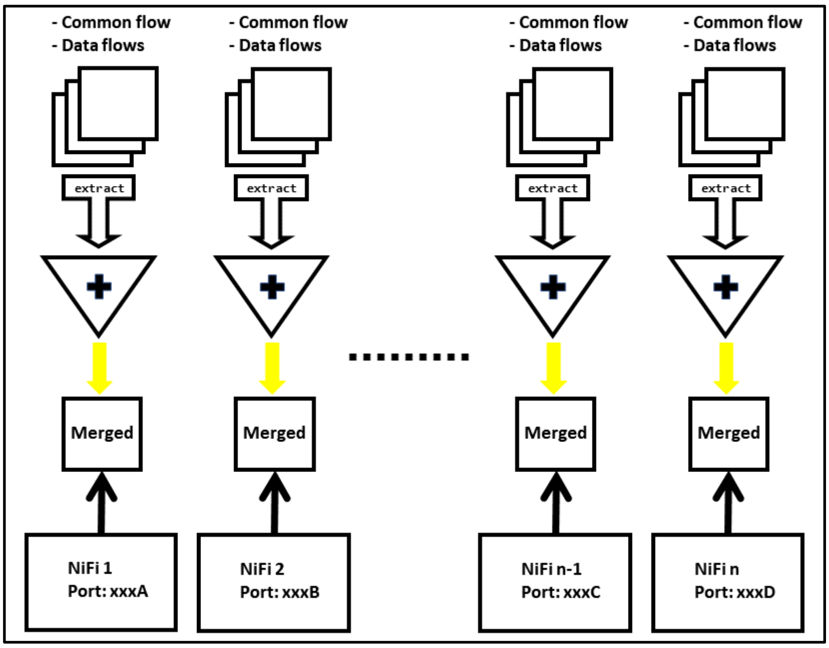


| | |
|---|---|
| Common flow | C:\NiFi\nifi-flows\flows\flow_000_common |
| Merge flow | C:\NiFi\nifi-flows\flows\flow_000_merged |

| Size | Time stamp | Directory | Path | Extracted |
|---|---|---|---|---|
| 7 kB | 15-12-2019 14:12:28 | csvload | C:\NiFi\nifi-flows\flows\csvload | extracted |
| 7 kB | 10-12-2019 19:07:00 | firstattempt | C:\NiFi\nifi-flows\flows\firstattempt | extracted |
| 5 kB | 18-12-2019 16:25:45 | flow | C:\NiFi\nifi-flows\flows\flow | extracted |
| 9 kB | 19-12-2019 19:02:40 | flow_000 | C:\NiFi\nifi-flows\flows\flow_000 | extracted |
| 0 kB | 27-12-2019 21:53:23 | flow_000_common | C:\NiFi\nifi-flows\flows\flow_000_common | COMMON |
| 41 kB | 2-1-2020 14:52:01 | flow_000_merged | C:\NiFi\nifi-flows\flows\flow_000_merged | MERGED |
| 5 kB | 29-12-2019 17:17:47 | flow_001 | C:\NiFi\nifi-flows\flows\flow_001 | extracted |
| 5 kB | 29-12-2019 17:17:50 | flow_002 | C:\NiFi\nifi-flows\flows\flow_002 | extracted |
| 5 kB | 29-12-2019 17:17:54 | flow_003 | C:\NiFi\nifi-flows\flows\flow_003 | extracted |
| 5 kB | 29-12-2019 17:17:57 | flow_004 | C:\NiFi\nifi-flows\flows\flow_004 | extracted |
| 5 kB | 29-12-2019 17:18:01 | flow_005 | C:\NiFi\nifi-flows\flows\flow_005 | extracted |
| 5 kB | 29-12-2019 17:18:04 | flow_006 | C:\NiFi\nifi-flows\flows\flow_006 | extracted |
| 5 kB | 29-12-2019 17:18:08 | flow_007 | C:\NiFi\nifi-flows\flows\flow_007 | extracted |
| 5 kB | 19-12-2019 17:46:24 | flow_008 | C:\NiFi\nifi-flows\flows\flow_008 | extracted |
| 5 kB | 19-12-2019 17:48:55 | flow_009 | C:\NiFi\nifi-flows\flows\flow_009 | extracted |
| 7 kB | 10-12-2019 20:59:16 | secondattempt | C:\NiFi\nifi-flows\flows\secondattempt | extracted |
| 7 kB | 11-12-2019 14:53:03 | test3 | C:\NiFi\nifi-flows\flows\test3 | extracted |

EXECUTE: Extract the process group from the *data flow*.
SAVE   : Save the result to the *merged flow*.

**Apache NiFi** is a webserver application default running at port 8080. You can cluster **NiFi**, but you can also run multiple instances of **NiFi** at different ports. You can install at different physical virtual machines, or at different locations on the same physical machine, all having its own port. When you install at different machines, having different IP addresses, you can use the same port number. This means you spread workload by making the *merged flow* smaller. Instead of 100 flows, you can split it in 4 times 25 flows. Or group per flow complexity. You get something as depicted below.



Here, every **NiFi** instance has its own common flow. You can also use one for all. As far as I can see, you are quite flexible in installing **NiFi** instances. You can also make it as complex as possible. You can get lost in configuration optimisation, or write robust flows who do not break and take the default throughput times for granted which are ok, according the official documentation.

## 4. Terminology and cut-over.

Next summarizes the terms used so far, and a new one.

| | |
|---|---|
| **data flow** | One single *data flow*. |
| **common flow** | A placeholder flow, containing parameters. |
| **merged flow** | A merge of the *common flow* and as many *data flows* you like. |
| **config flow**<br>**executing flow** | This is the flow stored in the *config directory* and that is the one **Apache** is executing. When you want to execute a new *merged flow*, or whatever flow, you first have to copy that flow to this flow/directory. That is called **PUSHING**. You can also copy the *config flow* to another subdirectory, that is called **PULLING**. The application does that for you. Thus, the *config flow* is the *executing flow* and can be any flow. But, in production, it is a copy of the *merged flow*. That is important! |

Here you see a potential draw-back of this approach. Because everything is in one flow, and although you can edit any *data flow* you like, at the moment you have to bring a new *data flow* in production, you have to make a new *merged flow*, easy, but then you have to shut down the existing *config flow*, meaning shutting down **Apache**, and replace it by a new *config flow*, and then re-start **Apache**. This means; one flow change has impact on all other flows. There is a cut-over issue here!

When you have thousand flows, running constantly, reading data every minute, and this cannot stop easily, then you might have an issue. But when data pulling from other systems is less critical, your flows are waiting for data that refresh a couple of time a day, daily, weekly, monthly, or when a slot is missed, source data is still available at next run, then no real issue here. On the other hand, your flows can always break, and have to deal with those situations. This forces you to write good *data flows*!

To give an example. New data comes available at 00:00. **NiFi**, when configurated that way, immediately picks it up and processes it. Assume **NiFi** also throws it away when processed. But **NiFi** is down, because you just have replaced the *config flow*. At 00:01 **NiFi** is running again. Then **NiFi** picks it up at 01:00. This will not stress your system. Assume all flows start at 00:00. Well, now they start at 00:01.

What if flows are in progress? The **NiFi GUI** does not show you the status of a flow, when it is running or not. In **NiFi**, all flows on the canvas are running by default. (You can turn on/off any process group you like, but in the real world they are all running waiting for data.) So, indeed you have no clue what is actually going on, unless you have a protocol that demands that every data flow records it starting and finishing time, busy or not busy status in a separate system, assuming such a status exists for a flow. Thus, a logging protocol, and look there. There is a **NiFi** summary, with a lot of statistical information, no doubt about that. But it does not show you which *process groups* are actually in progress. Again, for constantly running flows, that expect every second/minute data, this is indeed not relevant. But many flows are rather static.

Anyway, this is something to be aware off, and means you have to rely on the robustness of a data flow.

## 5. Practical use

**NiFi** supports templates. The idea is that many *data flows* share the same repetitive steps. You put those steps in a dedicated template, and then import it in your *data flow*. The ***NiFi Flow Manager*** is an extractor application. It is very easy to store any re-usable flow steps in a dedicated *data flow*, call it the *template flow*, and extract it from there in your own *data flow*. You merge your *data flow* with the *template flow* → results in a *merged flow* → becomes your new *data flow*. And then keep the elements you need.

## 6. Conclusion

When using **Apache NiFi**, I was a bit confused by the concept of one flow, and not able to save and opening the flow you want. That is why I came up with this approach. It is Windows Forms C# .NET application. Drive letters are needed to address the file system. All doable on a local machine, I think.

The assumption is that you have a local version of **Apache NiFi** installed, and develop locally. You can also the **Apache NiFi Registry** application.

The ***NiFi Flow Manager*** application is a simple and straightforward approach of maintaining your *data flows*. It is completely file-based and the whole authorization of who is allowed to edit what, is handled by how you set up the git repository or any other version/issue system you are using. The application itself is not aware of it. It simple sees the local file system. And yes, the extracting and merging of flows can all be done manually.

The cut-over issue is something you have to consider as being an issue or not. Otherwise you have to use the development method as described in the official documentation.

## 7. Future Features

- A check on unique processor ids in the *merged flow*.
- The tree view also shows the acceptance and production environment.
- Support for multiple instances of **NiFi**. (Also, a matter of enhanced tree view behaviour).
- Turning on **Apache NiFi** when you have replaced a *config flow*. The replacement is done automatically the turning on not.

Parameter extracting might be handy, but I think this makes things complex. It might be easier to add them to the *common flow* when needed.