# Tutorial 3 – Creating a fork and two branches: a pipeline to process chromatogram files followed by two distinct pipelines for DNA assembly

## Introduction

In this tutorial we will describe the specification of three distinct pipelines. The first one, described by the configuration file `trace_file_to_xml.cnf`, processes raw chromatogram files, similarly to what was seen in Tutorial 1. However, instead of assembling the sequences, it will stop at step 15, creating a snapshot of all the processing steps that have been performed along the pipeline. The component `outsave.pl` will generate a flatfile database (`final_snapshot.xml)` that will be used to feed two other distinct pipelines, using the `upload_xml.pl` component. The result will be the same as continuing the processing from the step 15, where the snapshot has been generated. This will be an example on how we can implement forks in a pipeline. Configuration files `xml_cap3.cnf` and `xml_phrap.cnf` will specify pipelines that represent the "branches" of the fork. These pipelines will upload the `final_snapshot.xml` file and submit it to the steps 16 to 18, described in Tutorial 1. The exception is that the pipeline specified by the configuration file `xml_phrap.cnf` will perform the assembly using the program Phrap instead of CAP3. For this task, a new component will be used, named `assembly_phrap.pl`. In this way, using different pipeline branches, we can compare the results of two distinct assembly programs.

The following steps constitute the pipe specified by the file `trace_file_to_xml.cnf`:

1. Uploading trace files and performing base calling and quality evaluation;
2. Masking primer sequences;
3. Masking vector sequences;
4. Filtering low quality sequences;
5. Saving sequences invalidated by the quality filter;
6. Trimming the bases that present a low Phred quality value and those that are masked;
7. Filtering sequences considered too small;
8. Saving sequences invalidated by the size filter;
9. Filtering mitochondrial sequences;
10. Saving sequences invalidated by the contaminant filter;
11. Filtering ribosomal sequences;
12. Saving sequences invalidated by the contaminant filter;
13. Saving sequences not previously invalidated by any filter;
14. Generating a report of all filtering steps;
15. Creating an XML snapshot recording all the processing steps that were performed;

The steps below constitute the pipe specified by the file `xml_cap3.cnf`:

16a. Uploading an XML file created by a previous pipeline;
17a. Assembling the valid sequences using CAP3;
18a. Generating an HTML page with graphical reports;
19a. Generating a complete graphical report.

The steps below constitute the pipe specified by the file `xml_phrap.cnf`:

16b. Uploading an XML file created by a previous pipeline;
17b. Assembling the valid sequences using Phrap;
18b. Generating an HTML page with graphical reports;
19b. Generating a complete graphical report.

We have previously constructed the pipelines for this tutorial using CoEd, EGene's graphical configuration editor. The EGene's configuration files (`trace_file_to_xml.gen`, `xml_cap3.gen` and `xml_cap3.gen`) and their counterpart text files (`trace_file_to_xml.cnf`, `xml_cap3.cnf` and `xml_cap3.cnf`) can be found at the `config_files` directory. In order to run the pipelines, go to the `/examples/xml_pipe` directory. This directory contains the subdirectory `chromat_dir`, which presents a set of trace files, and the file `primer_table.txt`, composed by a list of the primers used in the sequencing.

To run the first pipeline, you should type the following command:

```
bigou.pl -c ../config_files/trace_file_to_xml.cnf
```

If everything goes well, you should now find the following additional files in this directory:

```
filtered_by_quality.fasta
filtered_by_size.fasta
filtered_by_mitochondria.fasta
filtered_by_ribosome.fasta
good_sequences.fasta
filtering_report.html
final_snapshot.xml
```

Now run the second pipeline that will perform assembly with CAP3. Type the following command:

```
bigou.pl –c ../config_files/xml_cap3.cnf
```

If everything goes well, you should now find the following additional files in this directory:

```
redundancy_report_cap3.html
report_graphic_simple_cap3.html
```

and the following additional directories:

```
assembly_cap3_dir/
complete_report_cap3/
images_cap3_dir/
```

Finally, run the third pipeline that will perform assembly with Phrap. Type the following command:

```
bigou.pl –c ../config_files/xml_phrap.cnf
```

If everything goes well, you should now find the following additional files in this directory:

```
redundancy_report_phrap.html
report_graphic_simple_phrap.html
```

and the following additional directories:

```
assembly_phrap_dir/
complete_report_phrap/
images_phrap_dir/
```

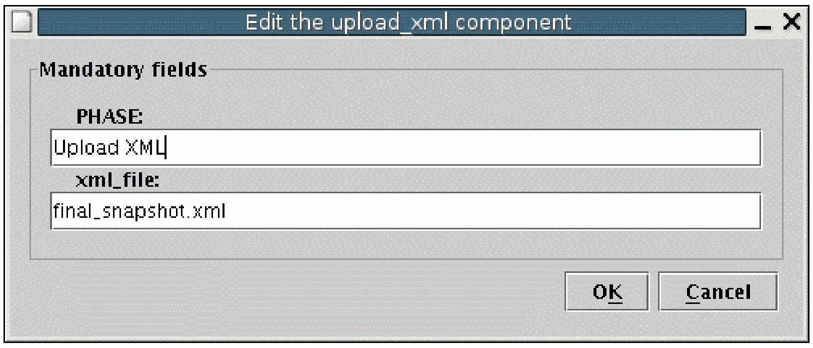
The first pipeline, specified by the configuration file `trace_file_to_xml.cnf`, is identical to the steps 1 to 18 already seen in Tutorial 1 and will not be commented here. The last step (15) creates a snapshot recording all the processing steps that were performed. This file can be used as an input in subsequent pipeline, as we will see below.

The second pipeline, specified by the configuration file `xml_cap3.cnf`, contains the following steps:

**16a. Uploading an XML file created by a previous pipeline**

Configuration parameters in the .cnf file:

```
#======================================================================
PHASE=Upload XML
program = upload_xml.pl
#----------------------------------------------------------------------
xml_file = final_snapshot.xml
#======================================================================
```


Edit the upload_xml component

Mandatory fields

PHASE:
Upload XML

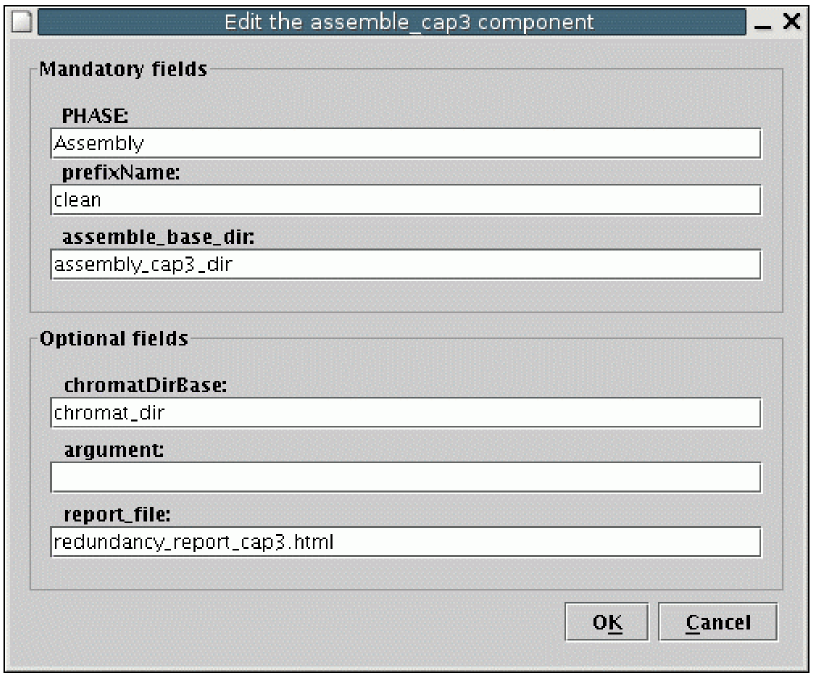
xml_file:
final_snapshot.xml

This step uses the component `upload_xml.pl` to upload an XML file created by a previous pipeline, and located at the directory specified by the user. The only argument to this component is the name of this XML file (in our case `final_snapshot.xml`). It is assumed that `bigou.pl` is run while the shell is in the directory that contains the file `final_snapshot.xml`. Alternatively, the user can specify a complete path for the file (e.g. `/home/test/final_snapshot.xml`).

### 17a. Assembling the valid sequences

Configuration parameters in the .cnf file:

```
#=====================================================================
PHASE=Assembly
program = assemble_cap3.pl
#---------------------------------------------------------------------
prefixName = clean
assemble_base_dir = assembly_cap3_dir
chromatDirBase = chromat_dir
assemble_phds = /dev/null
report_file = redundancy_report_cap3.html
#=====================================================================
```


Edit the assemble_cap3 component

Mandatory fields

PHASE: Assembly

prefixName: clean

assemble_base_dir: assembly_cap3_dir

Optional fields

chromatDirBase: chromat_dir

argument:

report_file: redundancy_report_cap3.html

In this step, we will assemble the valid sequences using the program CAP3. The component that performs this task is `assemble_cap3.pl`. The component `assemble_cap3.pl` does not alter any sequence, but rather generates new files. The parameters were already discussed in Tutorial 1 and will not be commented here. At the same way the other steps of this pipeline, which were already introduced in Tutorial 1, will not be covered here.

The third pipeline, specified by the configuration file `xml_phrap.cnf`, contains the following steps:

### 16b. Uploading an XML file created by a previous pipeline

Configuration parameters in the .cnf file:

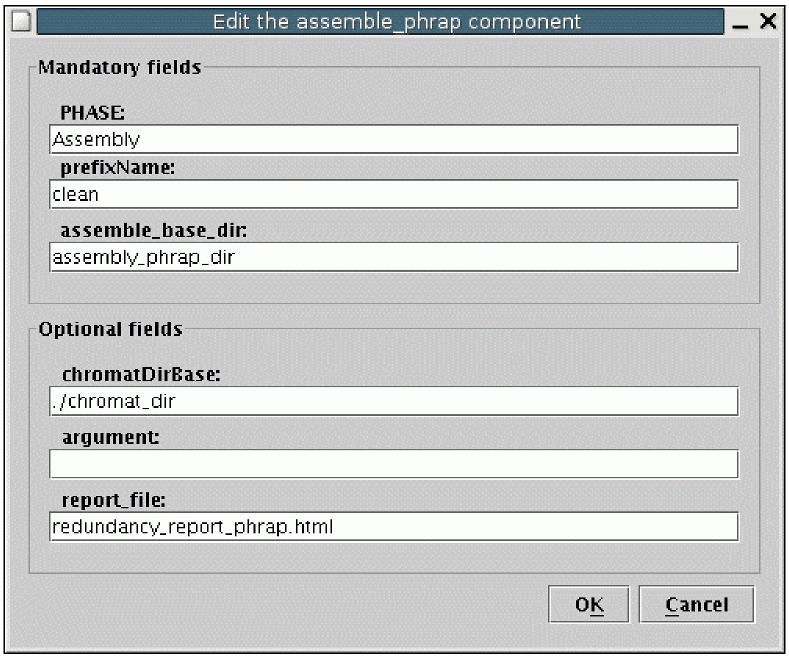
```
#======================================================================
PHASE=Upload XML
program = upload_xml.pl
#----------------------------------------------------------------------
xml_file = final_snapshot.xml
#======================================================================
```

This step is identical to step 16b, discussed above. It will permit the upload of an XML file created by the previous pipeline.

**17b. Assembling the valid sequences**

Configuration parameters in the .cnf file:

```
#======================================================================
PHASE=Assembly
program = assemble_phrap.pl
#----------------------------------------------------------------------
prefixName = clean
assemble_base_dir = assembly_phrap_dir
chromatDirBase = ./chromat_dir
assemble_phds = /dev/null
report_file = redundancy_report_phrap.html
#======================================================================
```


Edit the assemble_phrap component

Mandatory fields

PHASE: Assembly

prefixName: clean

assemble_base_dir: assembly_phrap_dir

Optional fields

chromatDirBase: ./chromat_dir

argument:

report_file: redundancy_report_phrap.html

In this step, we will assemble the valid sequences using the program Phrap. The component that performs this task is `assemble_phrap.pl`. The component `assemble_phrap.pl` does not alter any sequence, but rather generates new files. The parameters are identical to those used for the componente `assemble_phrap.pl. They` were already discussed in Tutorial 1 and will not be commented here. At the same way the other steps of this pipeline, which were already introduced in Tutorial 1, will not be covered here.

If you followed this tutorial without problems, you should now be able to compare the assemblies performed by both CAP3 and Phrap programs. They can be accessed in the subdirectories `assembly_cap3_dir` and `assembly_phrap_dir`, respectively.

© Copyright 2004: Alan M. Durham and Arthur Gruber