bioGUI Documentation

bioGUI

Nov 17, 2022

Contents

Bioinformatics is a highly interdisciplinary field providing tools for researchers from many disciplines. Nonetheless, most methods are implemented with a command line Interface only.

Using *bioGUI*, former command line-only tools can be started from a GUI, making them available to a broader scientific community.

If you are a **user**, it is recommended to start reading directly from *bioGUI: a universal GUI for command line tools*. Please also read through the *User Guide*. We also provide a video tutorial to demonstrate how to install a module with an *install module* and use this module/tool afterwards in *Use an Install Module (video tutorial)*.

If you are a (bioinformatics) tools **developer** and want to learn how to build (install) templates, start reading here: *How does bioGUI work?*.

If the offered windows and execution nodes are not enough and you want to **extend** *bioGUI*, the *bioGUI* *Developer Guide* may be of help for you.

CHAPTER 1

bioGUI: a universal GUI for command line tools

Bioinformatics is a highly interdisciplinary field providing tools for researchers from many disciplines. Nonetheless, most methods are implemented with a command line Interface only. Non-computer affine colleagues may well interpret results from such tools, but installing and starting tools on the command-Line often is a problem. Providing a Graphical User-Interface (GUI) for bioinformatics tools is a step towards routinely applying these command line-only tools, and, thus a more effective interdisciplinary work.

bioGUI is a universal GUI for command line tools making use of Window's newest feature: WSL (Windows Subsystem for Linux), which provides a *native* Ubuntu bash on Windows. *bioGUI* templates are easily scriptable and render a GUI for user input from defined visual components elements. Install modules can install a tool and its template with few clicks from our emph{bioGUI} repository.

Using *bioGUI*, former command line-only tools can be started from a GUI, making them available to a broader scientific community.

1.1 Who is bioGUI for?

With bioGUI, domain experts, who don't want to be bothered with the command Line, are enabled to use high standard bioinformatics tools. bioGUI specifically aims at Windows users, as Microsoft just introduced the Windows Subsystem for Linux (WSL) with its *Bash on Ubuntu on Windows*. This system allows the usage of said sophisticated tools on a regular Windows computer as most people have. With bioGUI one also does not need any knowledge about the command line, because the task of executing a given tools becomes a point & click solution.

1.2 What is bioGUI *currently* not?

It is not about generating a GUI by its own. *Currently* generating template files is a manual business, which is best performed by a tool's developer. For the future, integration of automatic command line Interface to *bioGUI* converters are thought of. However, this could only be realised for a limited number of argument parsers, such as *argparse* for *python*.

CHAPTER 2

User Guide

2.1 How to get bioGUI?

Download bioGUI releases from the github releases. Please note additional information provided in our *User Guide*.

2.2 Install bioGUI

Binary packages/releases for the following operating system are provided on github.

2.2.1 Windows

Extract the zip-File and place the contained bioGUI folder somewhere on your hard drives.

Follow the instructions to setup WSL: *How to setup WSL (Bash on Ubuntu)*.

Please do not put bioGUI in your Program Files directory, as this is specially protected by Windows and may cause problems.

Place bioGUI into a location which does not contain spaces in its name, e.g not `C:\Program Files\bioGUI` **!**

You can put bioGUI for instance into the locations `C:\bioGUI\` **or** `D:\bioGUI\` **!**

If you want to access external drives (USB stick, network drive), you first need to *mount* this drive into WSL. The WSL **Mount Drive** install module will install a script which can do this for you. Make sure to save the template and use the *Mount Drive (WSL)* template to make the drive available to WSL. You need to enter the drive-letter you want to mount (e.g. F), and your *sudo* password.

2.2.2 Linux

Extract the *tar.gz* file and place the contained bioGUI folder somewhere on your hard drives. On Ubuntu you can then execute the `bioGUI.desktop` file, e.g. from your Explorer equivalent, or being in the bioGUI folder, the command-line: `dex bioGUI.desktop` (you may need to install *dex* first: `sudo apt-get install dex`). Alternatively you can also simply execute `sh ./bioGUI.sh`.

2.2.3 Mac OS

Download and open the provided dmg package. You can simply drag and drop the bioGUI.app into your Applications folder. *bioGUI* has been built for Mac OS X 10.14 Mojave. Mac OS may bother your about running an app from a non-signed/verified developer. In the `Preferences->Security` menu you can tell Mac OS to still run *bioGUI*.

In order to use bioGUI, it may be required to install the OSX command line tools as well as brew. For the command line tools, open a *Terminal* and enter `xcode-select --install`. You can get brew from here To test brew, simply run the following code:

```
brew install wget netcat
```

Make sure to have brew in your path.

From the terminal, run:

```
echo "export PATH=/usr/local/bin/:\$PATH" >> ~/.bash_profile
```

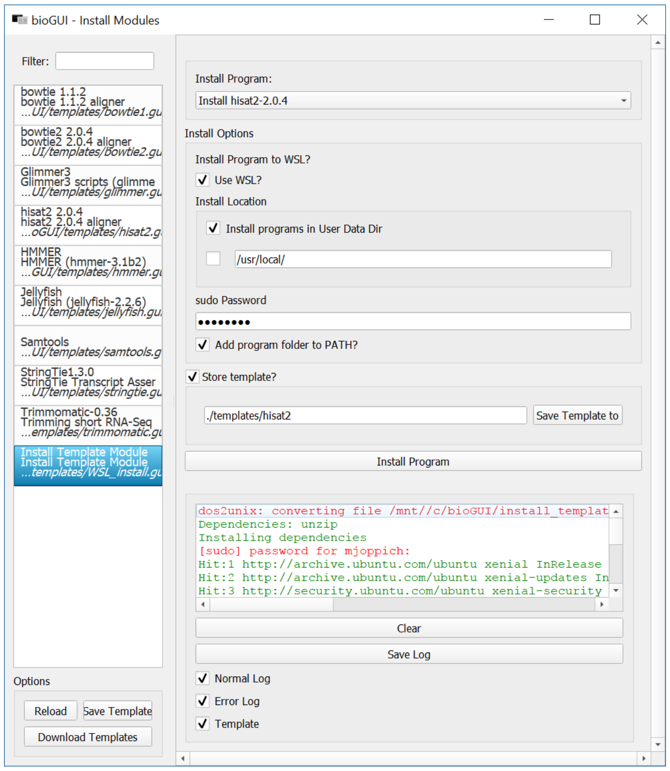
This will tell Mac OS to look for (unix) applications also under `/usr/local/bin/`. This is where brew installs itself.

2.3 Installing new software with Install Modules

After downloading an *Install Module*, the *Install Template Module* will list the downloaded install template (you might need to *reload* once). The screenshot below shows the *Install Template Module*. In the dropdown menu, the *hisat2* install template has been chosen. Since this module is executed on Windows, *WSL* is selected. *hisat2* is supposed to be installed into the *User Data Dir*, which is on *WSL* and linux usually `~/.local/share/bioGUI`. This is the directory where *bioGUI* installs new programs. Advanced users may want to change this settings, but in general, it should be left.

In order to automatically install dependencies, the sudo-password must be supplied. This is the password you set up during the installation of the *WSL* feature. This gives *bioGUI* administrative rights within *WSL* such that it can install dependencies automatically.

Finally the path for the specific *GUI* template has to be set. *bioGUI* by default only searches the template directory besides the executable. *GUI* templates must have the file extension `.gui`. If your entered template name does not end with this extension, the extension is appended to your filename. As a short-cut, if you just enter a name (e.g. mygui), *bioGUI* will save the new template in the template directoy with the filename *mygui.gui*.


bioGUI - Install Modules

Filter:

bowtie 1.1.2
bowtie 1.1.2 aligner
...UI/templates/bowtie1.gu

bowtie2 2.0.4
bowtie2 2.0.4 aligner
...UI/templates/bowtie2.gu

Glimmer3
Glimmer3 scripts (glimme
...UI/templates/glimmer.gu

hisat2 2.0.4
hisat2 2.0.4 aligner
...oGUI/templates/hisat2.g

HMMER
HMMER (hmmer-3.1b2)
...GUI/templates/hmmer.gu

Jellyfish
Jellyfish (jellyfish-2.2.6)
...UI/templates/jellyfish.gu

Samtools
...UI/templates/samtools.g

StringTie1.3.0
StringTie Transcript Asser
...UI/templates/stringtie.gu

Trimmomatic-0.36
Trimming short RNA-Seq
...emplates/trimmomatic.gu

Install Template Module
Install Template Module
...templates/WSL_install.gu

Install Program: Install hisat2-2.0.4

Install Options

Install Program to WSL?
✓ Use WSL?

Install Location
✓ Install programs in User Data Dir
☐ /usr/local/

sudo Password: ••••••••

✓ Add program folder to PATH?

✓ Store template?
./templates/hisat2

```
dos2unix: converting file /mnt//c/bioGUI/install_templat
Dependencies: unzip
Installing dependencies
[sudo] password for mjoppich:
Hit:1 http://archive.ubuntu.com/ubuntu xenial InRelease
Hit:2 http://archive.ubuntu.com/ubuntu xenial-updates In
Hit:3 http://security.ubuntu.com/ubuntu xenial-security
```

✓ Normal Log
✓ Error Log
✓ Template

2.4 Use-case: Windows setup

The binary distribution (zip-files) are targeted for *end-users*: [prebuilt binaries](https://github.com/mjoppich/bioGUI/releases). Download the Windows version.

Make sure that the Windows Subsystem for Linux (WSL) is installed. Please follow the steps on [how to setup WSL](http://biogui.readthedocs.io/en/latest/build_wsl.html).

After downloading the zip-archive, please unzip the archive to a location of your preference. Then simply start the executable (bioGUI.exe on Windows). *Place bioGUI into a location which does not contain spaces in its name, e.g not C:Program FilesbioGUI !* C:bioGUI is fine though!

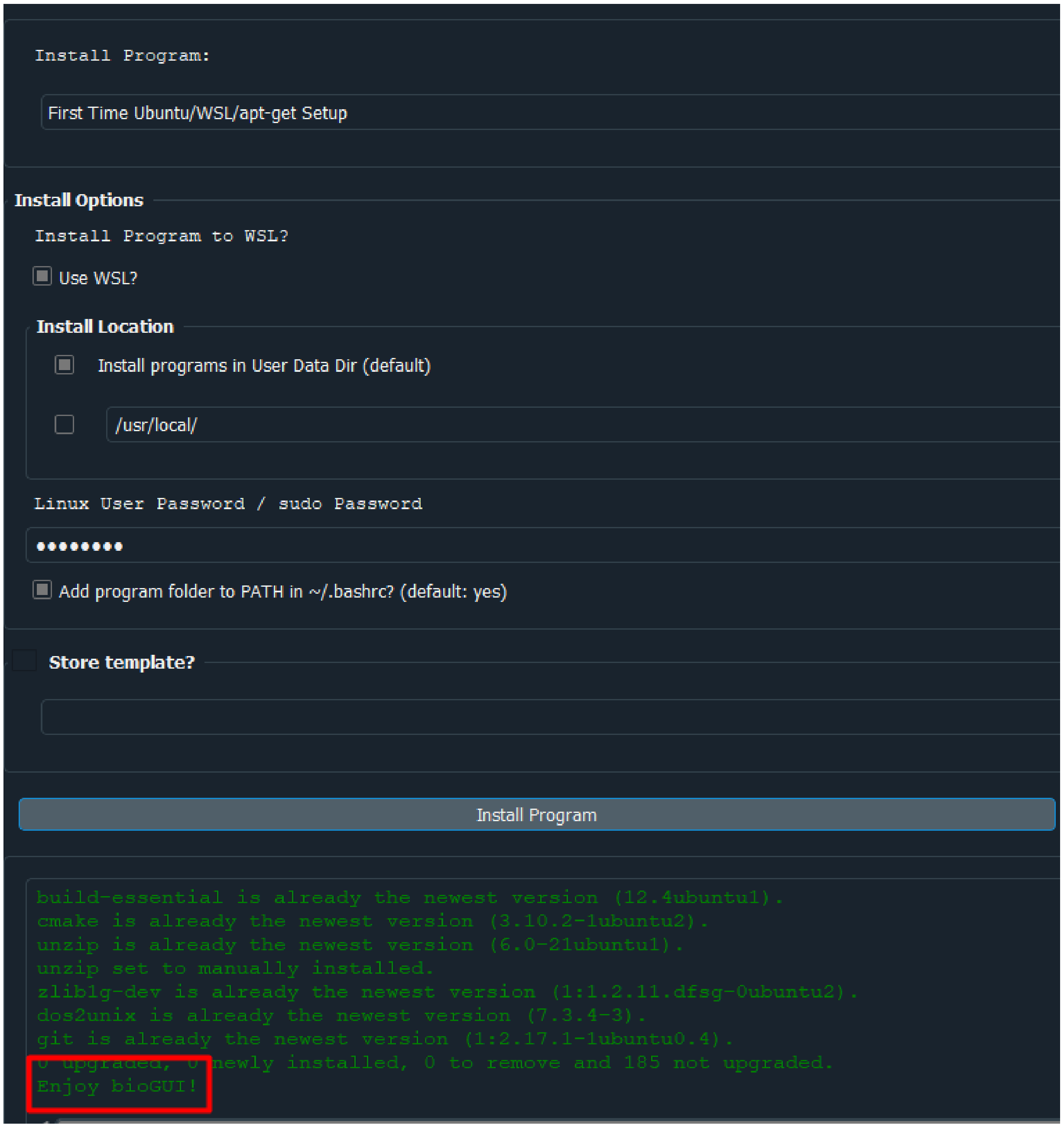
On any aptitude supported platform (e.g. Windows with WSL, Ubuntu), please download the "First Time Ubuntu/WSL/apt-get Setup" from the list of available templates and install it via *Install Template Module* (install program: *First Time Ubuntu/WSL/apt-get setup*).

Below this process is shown by an animation.

2.4.1 Setup First Time Use

For the *First Time Use* setups, please make sure to insert your *sudo/user* password and deselect to save a template.

Please make sure that bioGUI closes the setup with the following message:

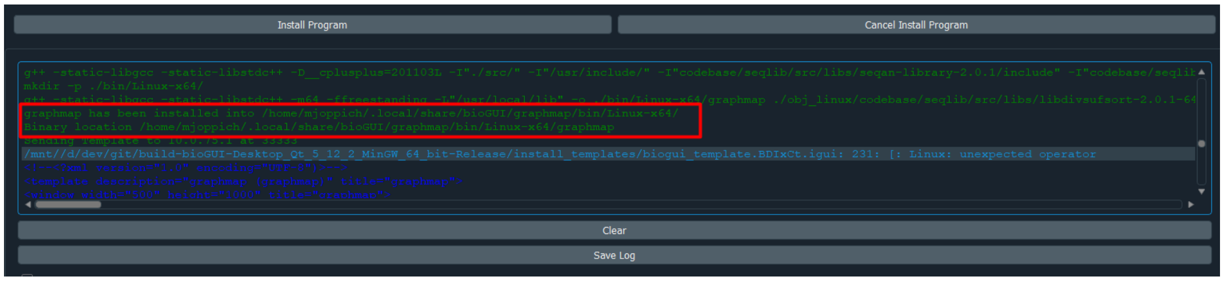

Install Program:
First Time Ubuntu/WSL/apt-get Setup
Install Options
Install Program to WSL?
Use WSL?
Install Location
Install programs in User Data Dir (default)
/usr/local/
Linux User Password / sudo Password
••••••••
Add program folder to PATH in ~/.bashrc? (default: yes)
Store template?
Install Program

```
build-essential is already the newest version (12.4ubuntu1).
cmake is already the newest version (3.10.2-1ubuntu2).
unzip is already the newest version (6.0-21ubuntu1).
unzip set to manually installed.
zlib1g-dev is already the newest version (1:1.2.11.dfsg-0ubuntu2).
dos2unix is already the newest version (7.3.4-3).
git is already the newest version (1:2.17.1-1ubuntu0.4).
0 upgraded, 0 newly installed, 0 to remove and 185 not upgraded.
Enjoy bioGUI!
```

If you do not see this message, (raise an issue on GitHub) with attaching the *log.txt* file in the bioGUI folder, or */tmp/log_biogui.txt* on Mac OS, as well as an screenshot of the attempt.

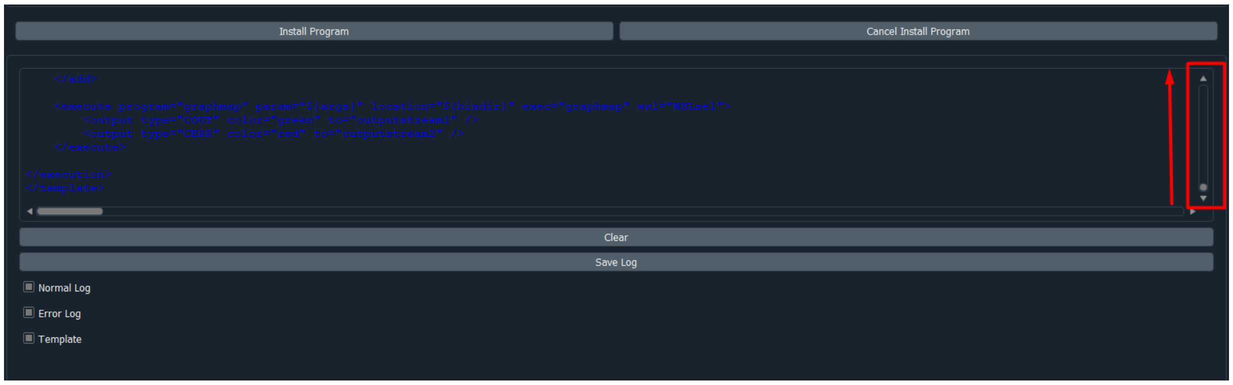
2.4.2 Install an install module

After the install module has been installed, you will see a message stating that *bioGUI* installed the software:



```
g++ -static-libgcc -static-libstdc++ -D__cplusplus=201103L -I"./src/" -I"/usr/include/" -I"codebase/seqlib/src/libs/seqan-library-2.0.1/include" -I"codebase/seqlib
mkdir -p ./bin/Linux-x64/
g++ -static-libgcc -static-libstdc++ -m64 -ffreestanding -L"/usr/local/lib" -o ./bin/Linux-x64/graphmap ./obj_linux/codebase/seqlib/src/libs/libdivsufsort-2.0.1-64
graphmap has been installed into /home/mjoppich/.local/share/bioGUI/graphmap/bin/Linux-x64/
Binary location /home/mjoppich/.local/share/bioGUI/graphmap/bin/Linux-x64/graphmap
Sending Template to 10.0.75.1 at 33333
/mnt//d/dev/git/build-bioGUI-Desktop_Qt_5_12_2_MinGW_64_bit-Release/install_templates/biogui_template.BDIxCt.igui: 231: [: Linux: unexpected operator
<!--<?xml version="1.0" encoding="UTF-8"?>-->
<template description="graphmap (graphmap)" title="graphmap">
<window width="500" height="1000" title="graphmap">
```

If you do not see the message, but only blue text, like below, you need to scroll up:



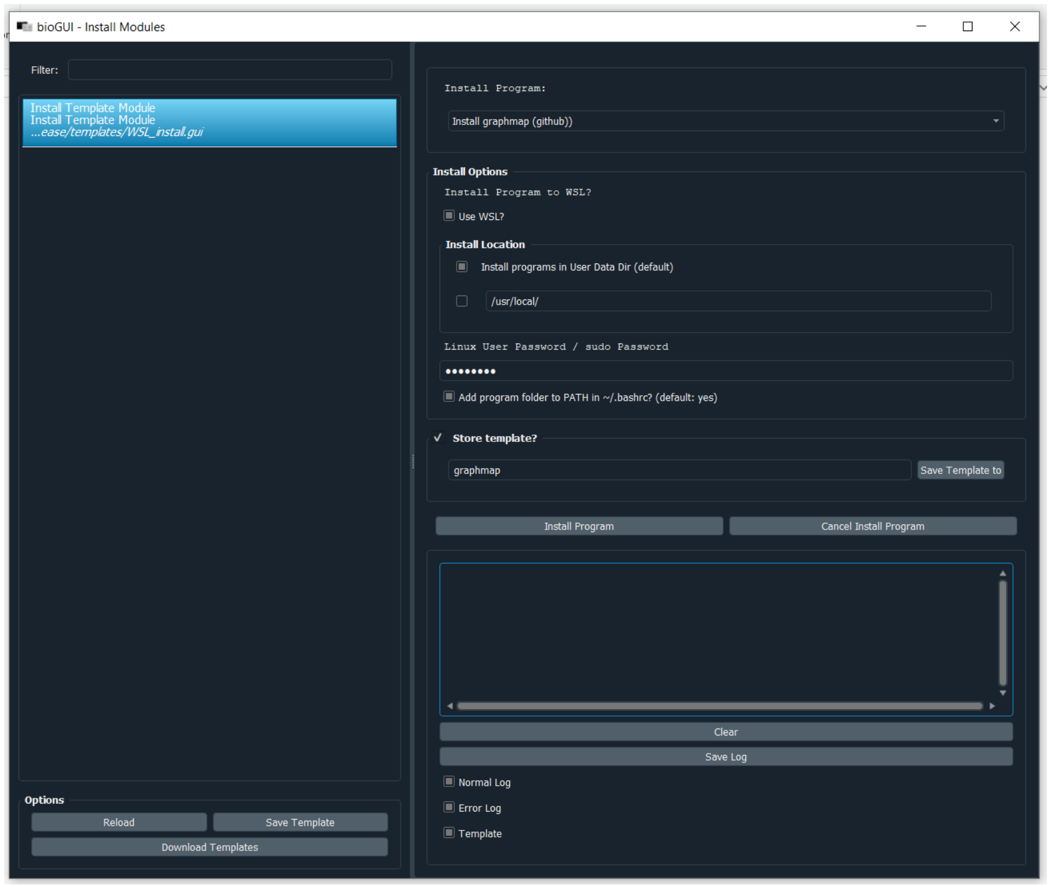
```
    </add>

    <execute program="graphmap" param="${args}" location="${bindir}" exec="graphmap" wsl="WSLsel">
        <output type="COUT" color="green" to="outputstream1" />
        <output type="CERR" color="red" to="outputstream2" />
    </execute>

</execution>
</template>
```

If you do not see this message, (raise an issue on GitHub) with attaching the *log.txt* file in the bioGUI folder, or */tmp/log_biogui.txt* on Mac OS, as well as an screenshot of the attempt.

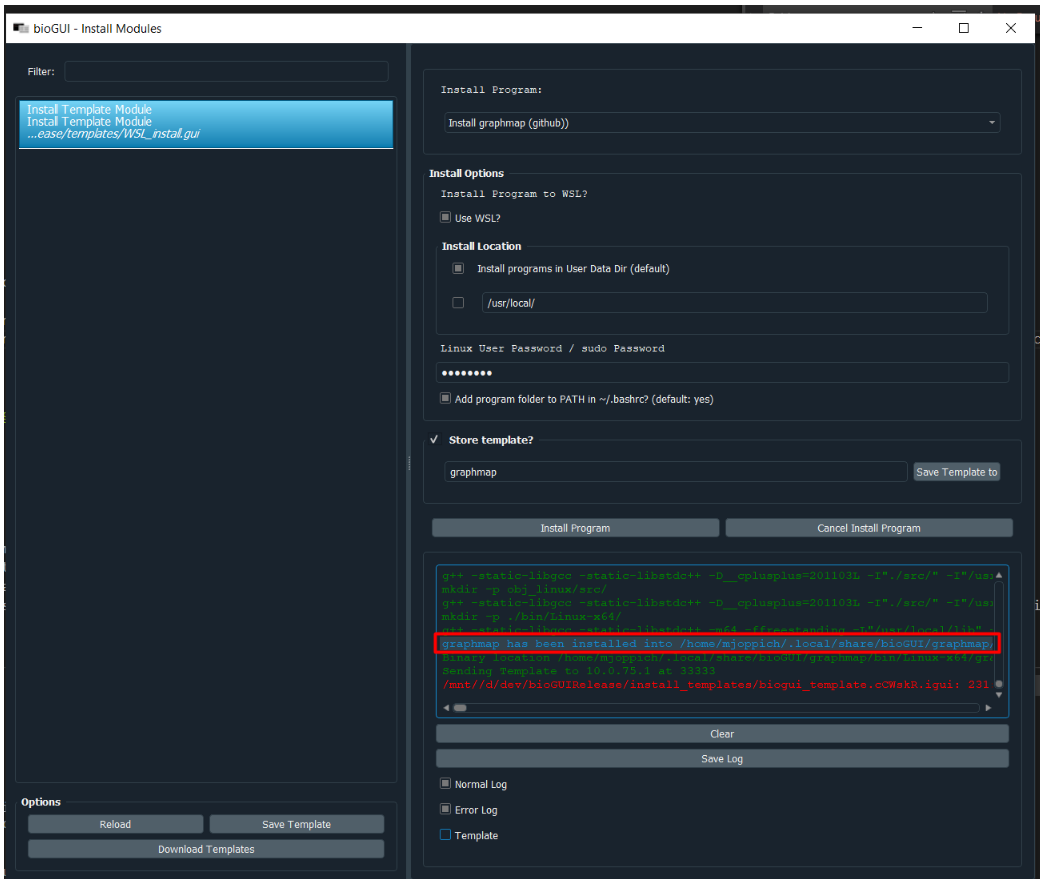
2.4.3 Use an Install Module

After installing a software, e.g. graphmap, you can reload the list of available templates by pressing the reload button.

Download the install module for graphmap as shown in the previous section. Next select the *graphmap* install module in the *Install Templates* template and fill out your *Linux User Password/sudo Password*. If you are on Windows, make sure that the *Use WSL?* checkbox is selected.

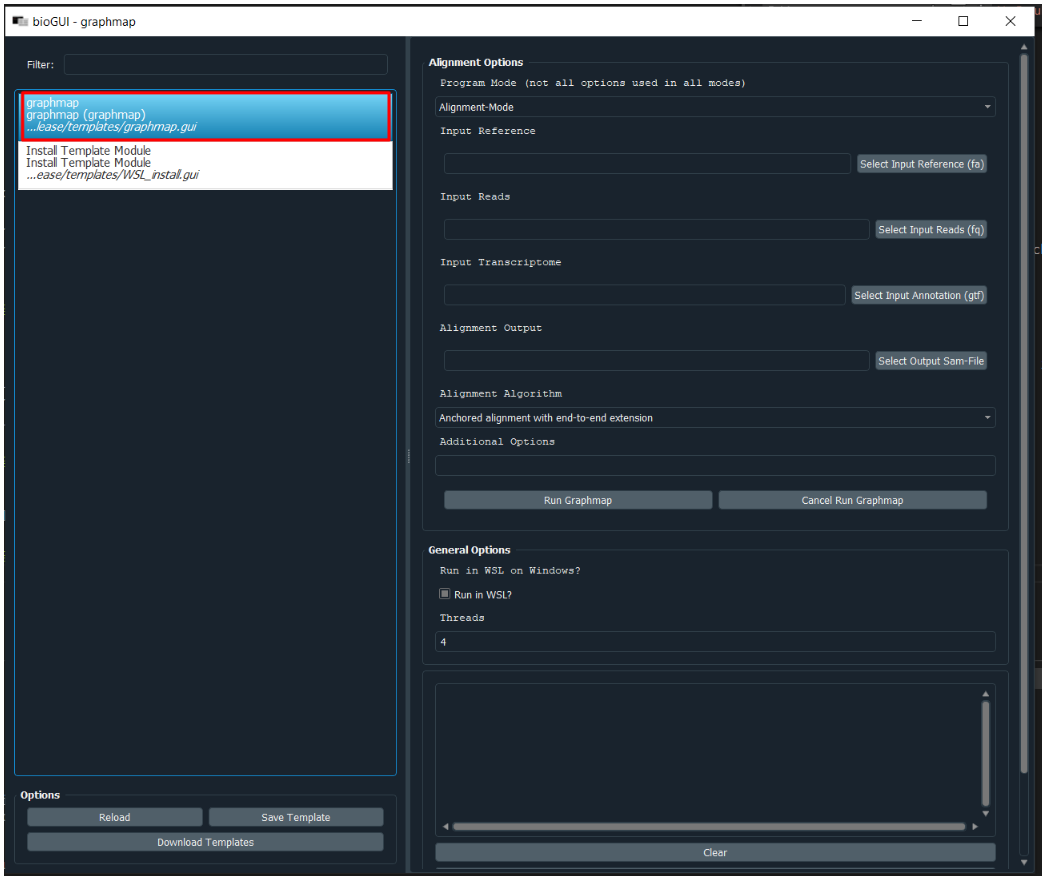


Enter the name of the template you would like to use in the input field within the *Store Template* group box (which you should have selected). In order for bioGUI to find a template it must reside in the *template* folder next to the bioGUI executable. Finally press *Install Program* and wait until the installation finishes. Besides the template being sent to bioGUI (in blue text), you should see the green message that your program has been installed correctly:



```
g++ -static-libgcc -static-libstdc++ -D__cplusplus=201103L -I"./src/" -I"/us
mkdir -p obj_linux/src/
g++ -static-libgcc -static-libstdc++ -D__cplusplus=201103L -I"./src/" -I"/us
mkdir -p ./bin/Linux-x64/
g++ -static-libgcc -static-libstdc++ -m64 -ffreestanding -L"/usr/local/lib"
graphmap has been installed into /home/mjoppich/.local/share/bioGUI/graphmap/
Binary location /home/mjoppich/.local/share/bioGUI/graphmap/bin/Linux-x64/gra
Sending Template to 10.0.75.1 at 33333
/mnt//d/dev/bioGUIRelease/install_templates/biogui_template.cCWskR.igui: 231
```

You can now *Reload* the templates again and will see a *graphmap* template on the left:



graphmap
graphmap (graphmap)
...lease/templates/graphmap.gui

Install Template Module
Install Template Module
...ease/templates/WSL_install.gui

Alignment Options

```
Program Mode (not all options used in all modes)
```

Alignment-Mode

```
Input Reference
Input Reads
Input Transcriptome
Alignment Output
Alignment Algorithm
```

Anchored alignment with end-to-end extension

```
Additional Options
```

General Options

```
Run in WSL on Windows?
```

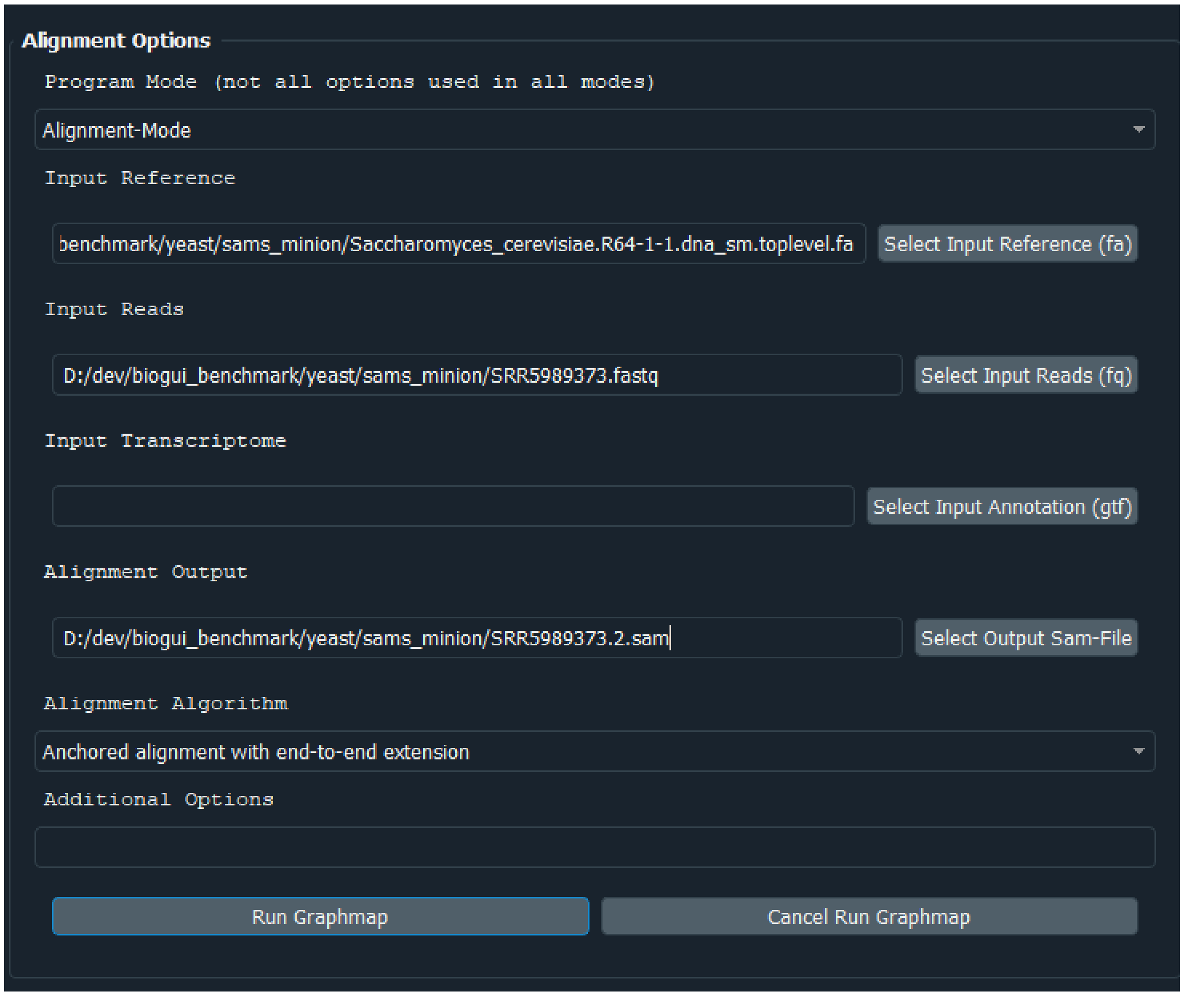
☐ Run in WSL?

```
Threads
```

4

In this template you have to possibility to select the alignment mode (align or overlap/owler), specify the reference sequence, reads, gene annotation file (for splice-aware alignment) as well as the output file. To select files via a file dialog, click the corresponding button on the right. Using the *Additional Options* field, you can also enter command-line parameters directly. This may be necessary because the template is not complete, or new options have been added (or because you know shortcuts).

Exemplarily we filled our some parameters:


Alignment Options

Program Mode (not all options used in all modes): Alignment-Mode

Input Reference: benchmark/yeast/sams_minion/Saccharomyces_cerevisiae.R64-1-1.dna_sm.toplevel.fa

Input Reads: D:/dev/biogui_benchmark/yeast/sams_minion/SRR5989373.fastq

Input Transcriptome:

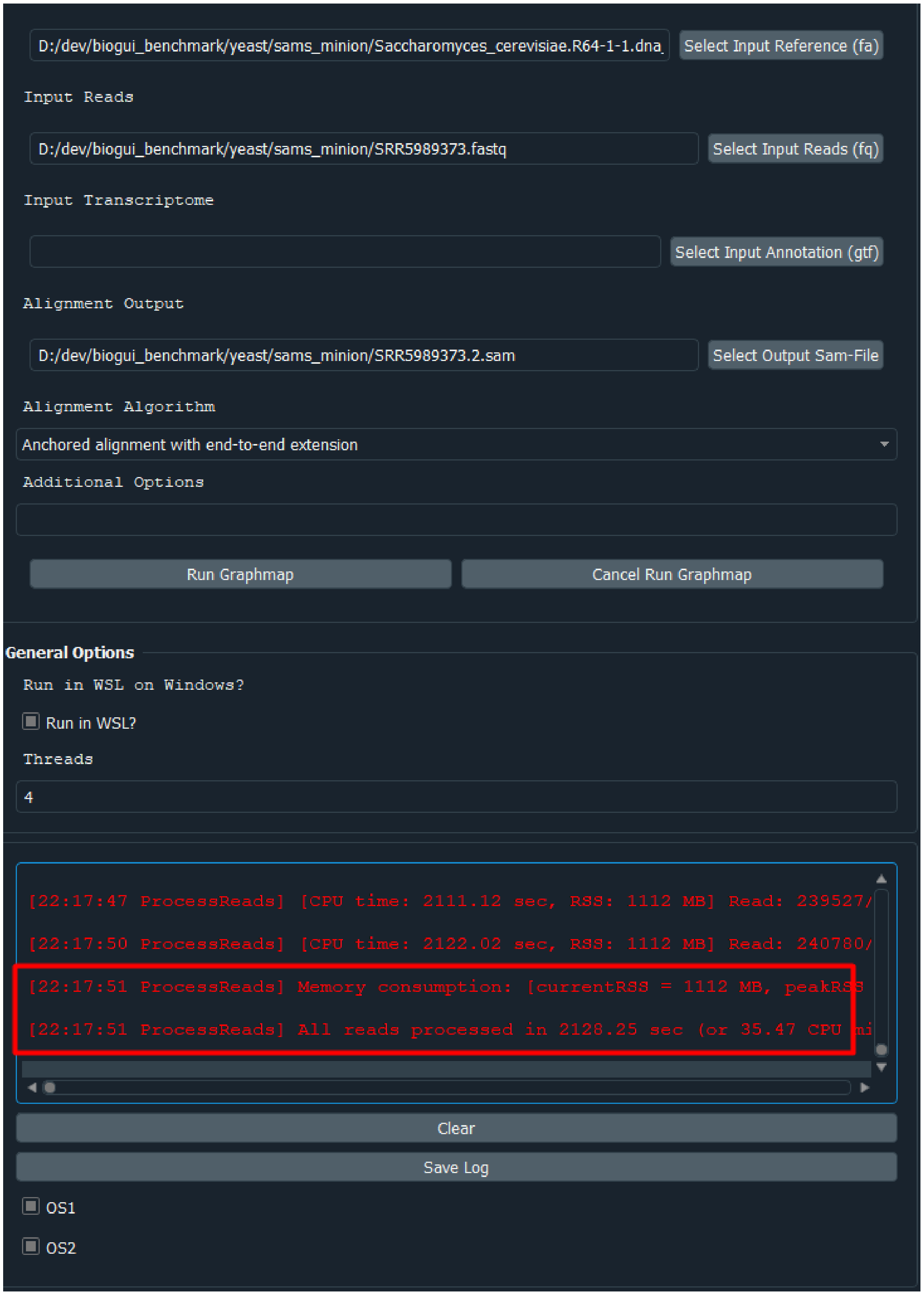
Alignment Output: D:/dev/biogui_benchmark/yeast/sams_minion/SRR5989373.2.sam

Alignment Algorithm: Anchored alignment with end-to-end extension

Additional Options:

Clicking on *Run Graphmap* will now execute graphmap via bioGUI. In this case, the called command-line tool is `graphmap -r <reference> -d <reads> -o <output>`.

Upon completion, the bioGUI options button (lower left) will become enabled again, as well as the *Run Graphmap* button. If you need to cancel the current process, click the *Cancel Run Graphmap* button. You will see all the intermediate output from graphmap and have the option to save the command-line output using the *Save log* button. If you want to save the inputs you made, using the *Save template* button of the bioGUI options, you can save the inputs you made. Save the template in the *template* directory in the same folder as bioGUI to see the template.

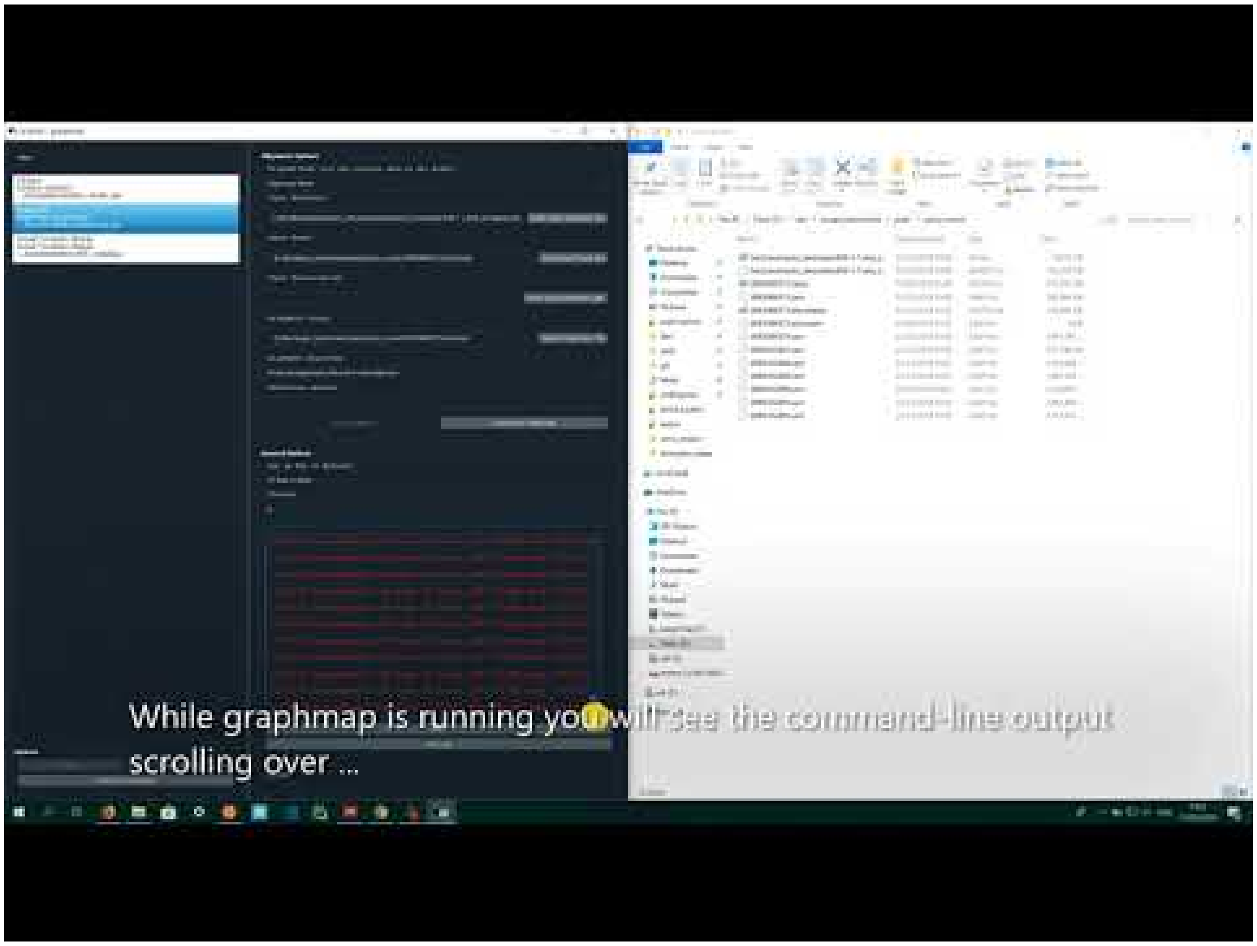

D:/dev/biogui_benchmark/yeast/sams_minion/Saccharomyces_cerevisiae.R64-1-1.dna.
Select Input Reference (fa)
Input Reads
D:/dev/biogui_benchmark/yeast/sams_minion/SRR5989373.fastq
Select Input Reads (fq)
Input Transcriptome
Select Input Annotation (gtf)
Alignment Output
D:/dev/biogui_benchmark/yeast/sams_minion/SRR5989373.2.sam
Select Output Sam-File
Alignment Algorithm
Anchored alignment with end-to-end extension
Additional Options
Run Graphmap
Cancel Run Graphmap
General Options
Run in WSL on Windows?
Run in WSL?
Threads
4
Clear
Save Log
OS1
OS2

```
[22:17:47 ProcessReads] [CPU time: 2111.12 sec, RSS: 1112 MB] Read: 239527/
[22:17:50 ProcessReads] [CPU time: 2122.02 sec, RSS: 1112 MB] Read: 240780/
[22:17:51 ProcessReads] Memory consumption: [currentRSS = 1112 MB, peakRSS
[22:17:51 ProcessReads] All reads processed in 2128.25 sec (or 35.47 CPU mi
```

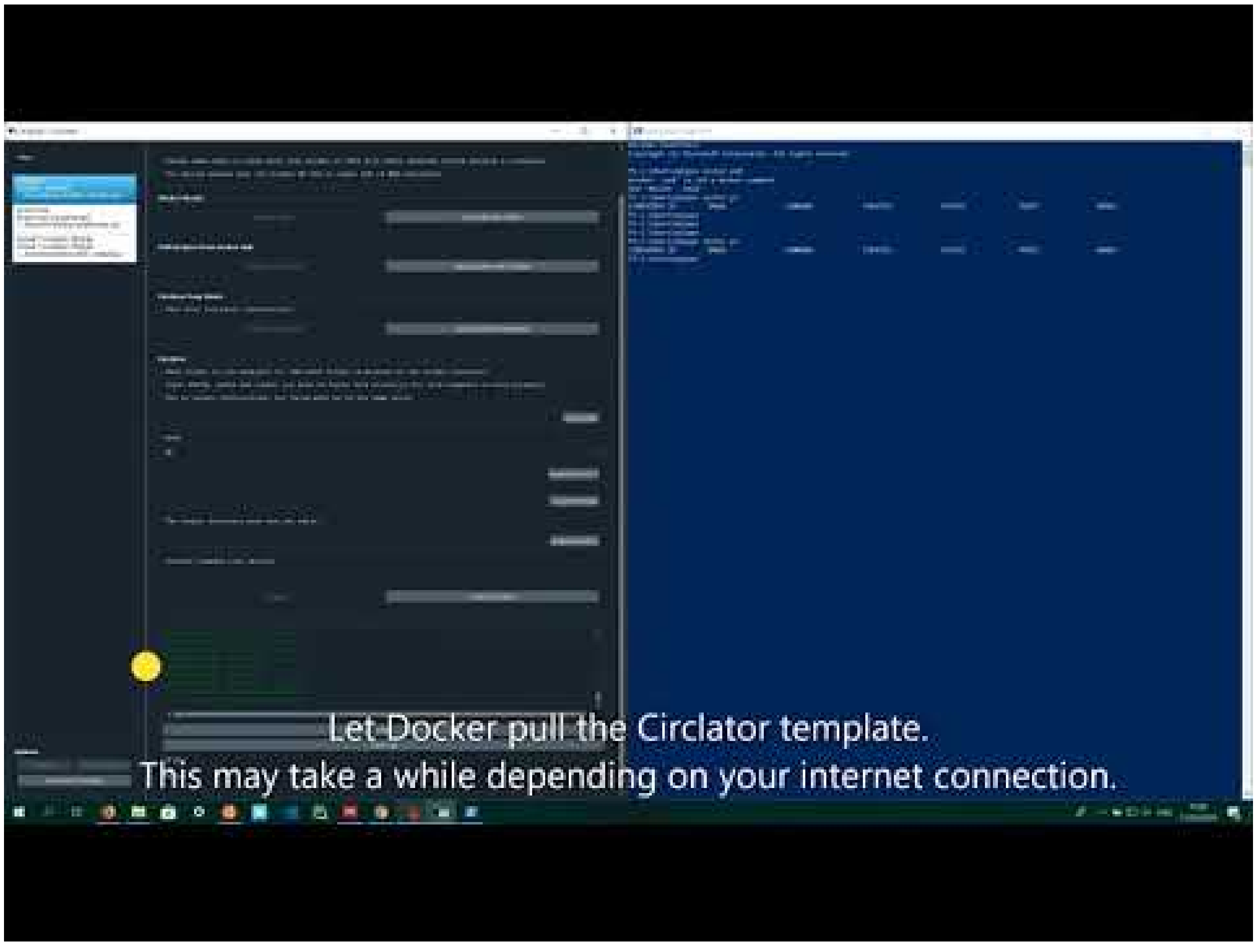
2.4.4 Use an Install Module (video tutorial)

If you prefer to understand these steps as part of a video, please have a look attaching

Installing graphmap (from install module) and using it:

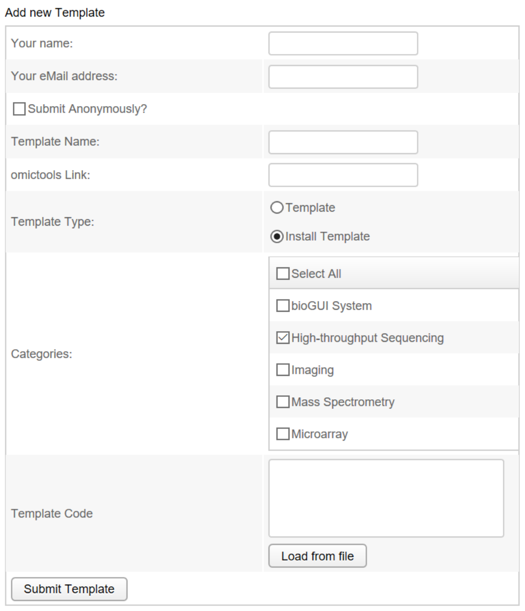

While graphmap is running you will see the command-line output scrolling over ...

Using the circlator docker template:


Let Docker pull the Circlator template.
This may take a while depending on your internet connection.

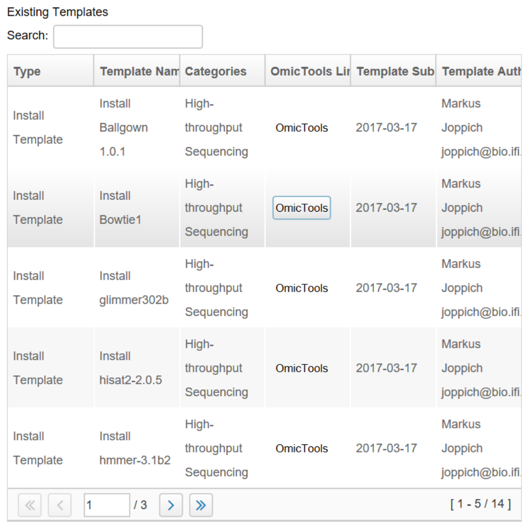
2.5 Adding Own Templates

Part of *bioGUI* is a simple accessibility of templates for various programs. Therefore, application developers and sophisticated users can upload their templates to our website in order to make their template available to other users. The user has to submit his name, eMail address, a template name and the template itself. He can select whether he wants to be an anonymous user (user name is always hidden) and whether this is an installation script (which downloads and creates the ac{GUI} template specifically for this installed application) or a regular template. Additionally, categories for the template can be supplied, e.g. whether this is a template for a sequencing tool, or proteomics. A screenshot of the template submission is shown below.


Add new Template

Your name:	
Your eMail address:	
☐ Submit Anonymously?	
Template Name:	
omictools Link:	
Template Type:	○ Template ● Install Template
Categories:	☐ Select All ☐ bioGUI System ☑ High-throughput Sequencing ☐ Imaging ☐ Mass Spectrometry ☐ Microarray
Template Code	Load from file
Submit Template	

Additionally the (bioGUI website) also contains a searchable list of already available templates. Available templates are only shown and can be downloaded via the *bioGUI* application.

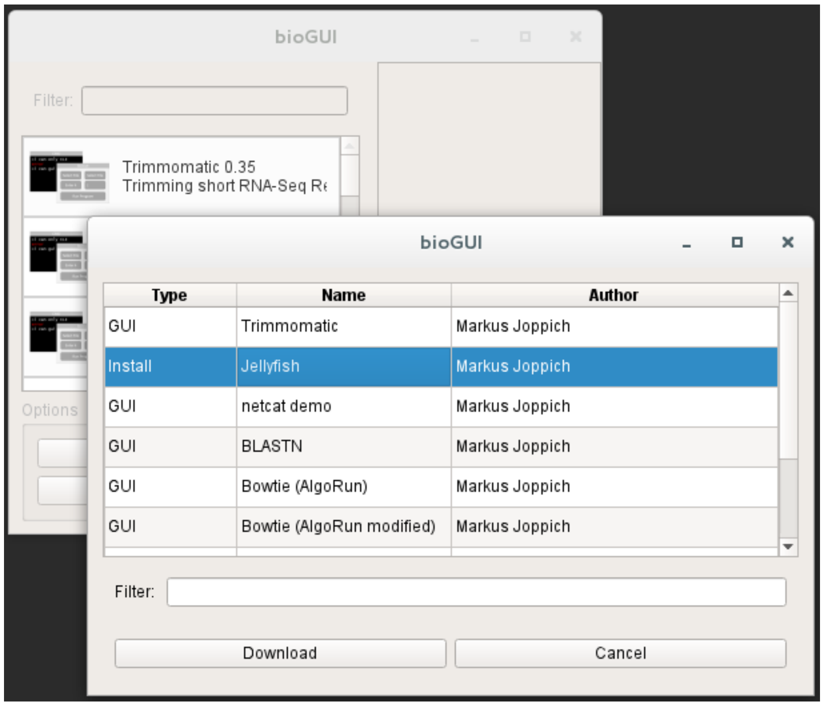

Existing Templates

Search:

Type	Template Nam	Categories	OmicTools Lir	Template Sub	Template Auth
Install Template	Install Ballgown 1.0.1	High-throughput Sequencing	OmicTools	2017-03-17	Markus Joppich joppich@bio.ifi
Install Template	Install Bowtie1	High-throughput Sequencing	OmicTools	2017-03-17	Markus Joppich joppich@bio.ifi
Install Template	Install glimmer302b	High-throughput Sequencing	OmicTools	2017-03-17	Markus Joppich joppich@bio.ifi
Install Template	Install hisat2-2.0.5	High-throughput Sequencing	OmicTools	2017-03-17	Markus Joppich joppich@bio.ifi
Install Template	Install hmmer-3.1b2	High-throughput Sequencing	OmicTools	2017-03-17	Markus Joppich joppich@bio.ifi

1 / 3 [1 - 5 / 14]

Within *bioGUI*, clicking the *Download Templates* button, a new dialog window opens showing a list of available templates. Columns can be sorted by double clicking the header, and using the search only templates which contain the searched words are shown. Upon selecting one or multiple (keep ctrl-key down while clicking) rows and clicking the *Download* button, those templates are downloaded and available for the user. Since it is possible to copy or alter templates, *bioGUI* never overwrites existing templates, but will create a new copy.



Type	Name	Author
GUI	Trimmomatic	Markus Joppich
Install	Jellyfish	Markus Joppich
GUI	netcat demo	Markus Joppich
GUI	BLASTN	Markus Joppich
GUI	Bowtie (AlgoRun)	Markus Joppich
GUI	Bowtie (AlgoRun modified)	Markus Joppich

CHAPTER 3

How to setup WSL (Bash on Ubuntu)

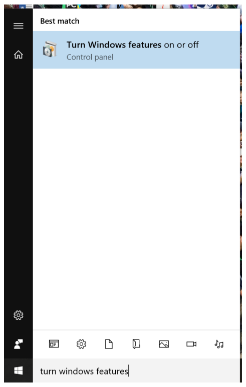
Depending on your Windows version you need to activate Developer Mode first. How this is done is explained at the end of this page.

For all recent versions of Windows 10 you can start with step 1.

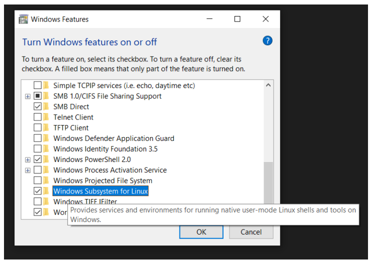
Warning: Some antivirus software (e.g. Kaspersky) disable internet access for unknown/new programs. Make sure bioGUI can access the internet!

3.1 Step 1: Activate WSL feature

First WSL has to be enabled from Windows features. Therefore, simply search for the *Turn Windows features on or off* option in the control panel.



Once found, look for the *Windows Subsystem for Linux (Beta)* row and make sure to check the corresponding box.



Windows Features

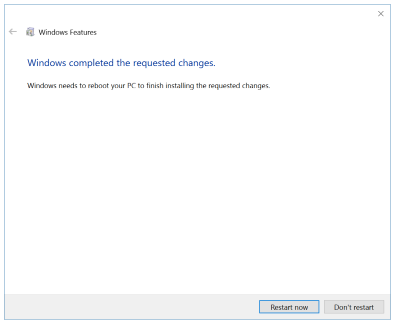
Turn Windows features on or off

To turn a feature on, select its checkbox. To turn a feature off, clear its checkbox. A filled box means that only part of the feature is turned on.

- [ ] Simple TCPIP services (i.e. echo, daytime etc)
- [■] SMB 1.0/CIFS File Sharing Support
- [✓] SMB Direct
- [ ] Telnet Client
- [ ] TFTP Client
- [ ] Windows Defender Application Guard
- [ ] Windows Identity Foundation 3.5
- [✓] Windows PowerShell 2.0
- [ ] Windows Process Activation Service
- [ ] Windows Projected File System
- [✓] Windows Subsystem for Linux
- [ ] Windows TIFF IFilter
- [✓] Wor

Provides services and environments for running native user-mode Linux shells and tools on Windows.

This will download and install the desired WSL feature. Finally apply the change and make sure to reboot your computer



Windows Features

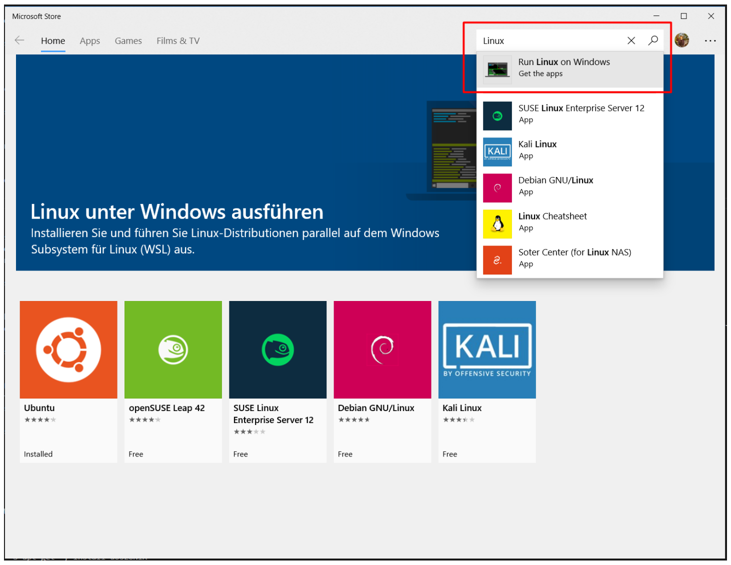
Windows completed the requested changes.

Windows needs to reboot your PC to finish installing the requested changes.

3.2 Step 2: Install Linux

After having enabled the WSL feature, we can visit the Microsoft Windows Store to download Linux.

In order to do so, we open the Windows Store app, and search for *Linux*. We select the *Run Linux on Windows* menu entry.

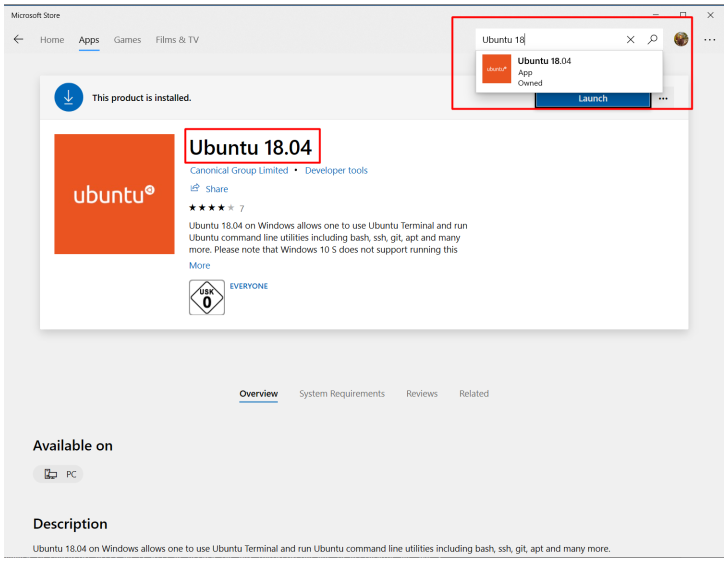


Linux unter Windows ausführen

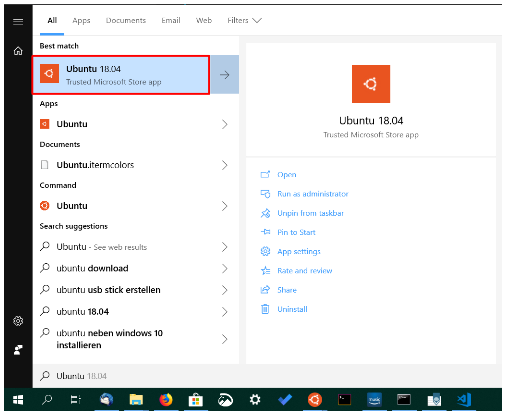
Installieren Sie und führen Sie Linux-Distributionen parallel auf dem Windows Subsystem für Linux (WSL) aus.

There are many different flavors (comparable to strains in biology) of linux and some are already offered on the Windows store. Best compatibility for *bioGUI* has Ubuntu.

Important: You should consider using the latest Ubuntu version available. This is Ubuntu 18.04 at the time of writing. You specifically have to search for *Ubuntu 18.04* in the store !

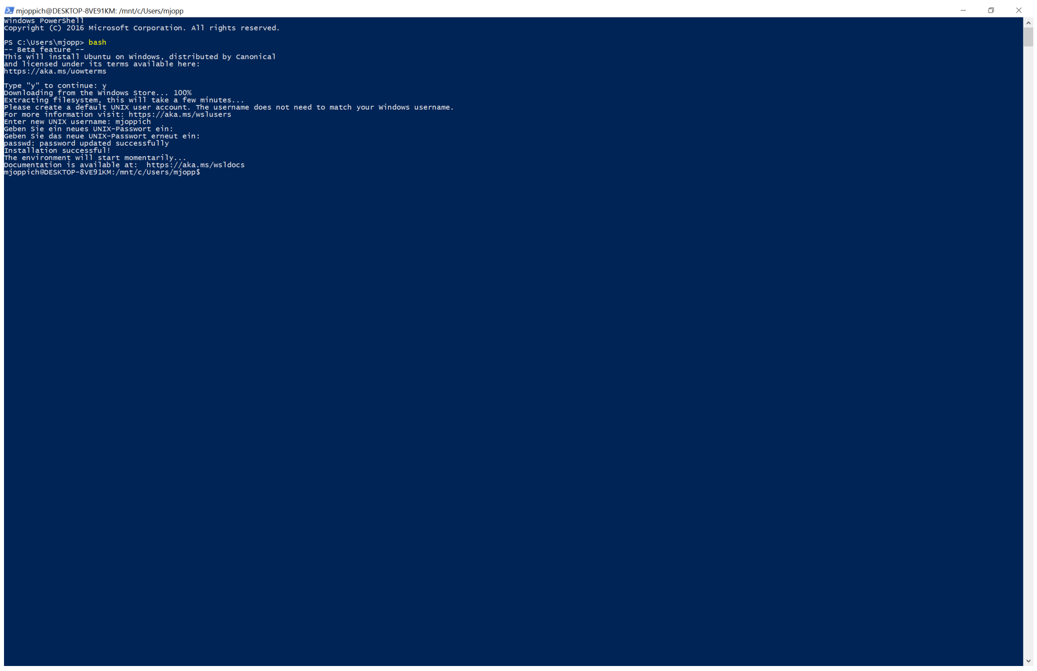

Microsoft Store
Home
Apps
Games
Films & TV
Ubuntu 18
Ubuntu 18.04
App
Owned
Launch
This product is installed.
Ubuntu 18.04
Canonical Group Limited • Developer tools
Share
Ubuntu 18.04 on Windows allows one to use Ubuntu Terminal and run Ubuntu command line utilities including bash, ssh, git, apt and many more. Please note that Windows 10 S does not support running this
More
EVERYONE
Overview
System Requirements
Reviews
Related
Available on
PC
Description
Ubuntu 18.04 on Windows allows one to use Ubuntu Terminal and run Ubuntu command line utilities including bash, ssh, git, apt and many more.

Now let the Windows store install your Linux app and once that is done, open your newly installed Linux:



The black screen will guide you through the install process. It will first unpack itself and then ask you to create a linux user account.

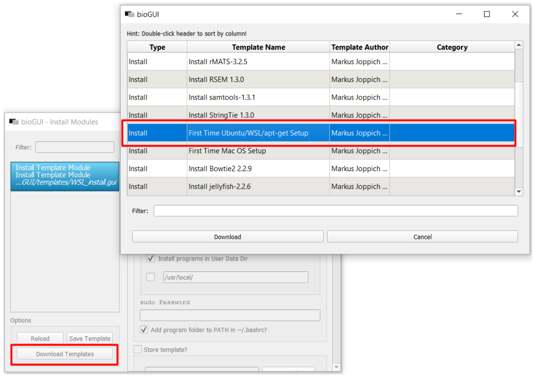
It is recommended to choose a username and password you can easily remember. Remembering the password is essential here, as it will be needed for any installation to be performed on *WSL* and by *bioGUI*.



```
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

PS C:\Users\mjopp> bash
-- Beta feature --
This will install Ubuntu on Windows, distributed by Canonical
and licensed under its terms available here:
https://aka.ms/uowterms

Type "y" to continue: y
Downloading from the Windows Store... 100%
Extracting filesystem, this will take a few minutes...
Please create a default UNIX user account. The username does not need to match your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username: mjoppich
Geben Sie ein neues UNIX-Passwort ein:
Geben Sie das neue UNIX-Passwort erneut ein:
passwd: password updated successfully
Installation successful!
The environment will start momentarily...
Documentation is available at:  https://aka.ms/wsldocs
mjoppich@DESKTOP-8VE91KM:/mnt/c/Users/mjopp$
```

3.3 Step 3: Prepare WSL

Before you can use *bioGUI* on *WSL*/Ubuntu please make sure to run the *First time Ubuntu/WSL/apt-get Setup* from *bioGUI*. For the *sudo/user password* please enter the password for your linux user account from the step above.



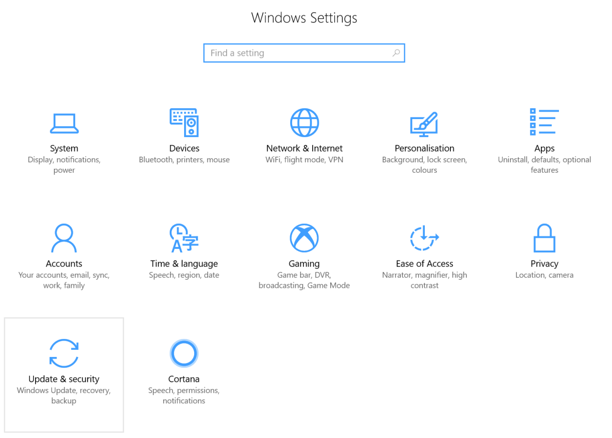
Type	Template Name	Template Author	Category
Install	Install rMATS-3.2.5	Markus Joppich ...	
Install	Install RSEM 1.3.0	Markus Joppich ...	
Install	Install samtools-1.3.1	Markus Joppich ...	
Install	Install StringTie 1.3.0	Markus Joppich ...	
Install	First Time Ubuntu/WSL/apt-get Setup	Markus Joppich ...	
Install	First Time Mac OS Setup	Markus Joppich ...	
Install	Install Bowtie2 2.2.9	Markus Joppich ...	
Install	Install jellyfish-2.2.6	Markus Joppich ...	

In case there are problems please contact the author of the software.

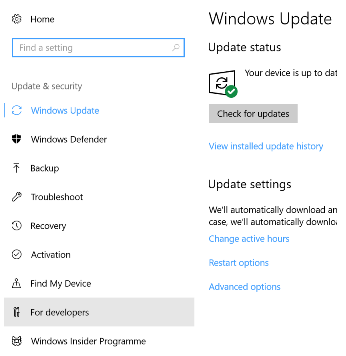
If you are running an old Windows 10 version, you first need to activate developer mode before you are able to enable the Windows Subsystem for Linux feature.

Step 0: Activate Developer Mode

Since WSL/Bash on Ubuntu on Windows is a developer feature, first the developer mode has to be actived. Therefore we go into the Settings app and select *Update & Security*.


Windows Settings
Find a setting
System
Display, notifications, power
Devices
Bluetooth, printers, mouse
Network & Internet
WiFi, flight mode, VPN
Personalisation
Background, lock screen, colours
Apps
Uninstall, defaults, optional features
Accounts
Your accounts, email, sync, work, family
Time & language
Speech, region, date
Gaming
Game bar, DVR, broadcasting, Game Mode
Ease of Access
Narrator, magnifier, high contrast
Privacy
Location, camera
Update & security
Windows Update, recovery, backup
Cortana
Speech, permissions, notifications

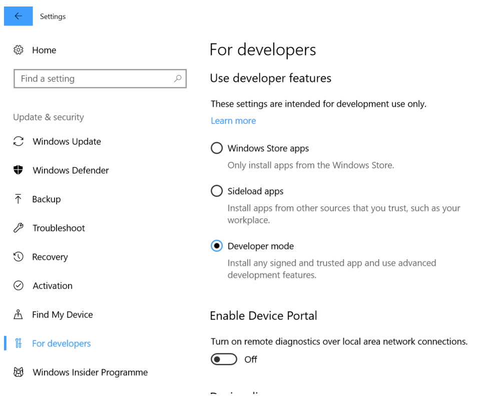
We further navigate into the *For developers* tab on the left.



Your device is up to dat

We'll automatically download an
case, we'll automatically downlo:

In the *For developers* options we switch from *Windows Store apps* to *Developer mode*.



These settings are intended for development use only.

Windows Store apps
Only install apps from the Windows Store.

Sideload apps
Install apps from other sources that you trust, such as your workplace.

[✓ Developer mode]
Install any signed and trusted app and use advanced development features.

Turn on remote diagnostics over local area network connections.
Off

CHAPTER 4

How does bioGUI work?

bioGUI relies on the philosophy that the visual appearance as well as the assembly of the command line arguments can be represented as a network. Especially templates are seen as network, both in their visual description but most importantly also in the execution model.

4.1 Install modules

bioGUI install modules are designed to make the regular user's life as easy as possible. Originally created for WSL, so pure Windows users can also easily install their needed software without the need to care about dependencies, the idea of the *bioGUI* repository evolved. Install modules are more than just templates, because they install the actual software onto the user's system, while customizing the *bioGUI* template for this application. All install modules are regular shell scripts, that are called with fixed parameters.

In general, *install modules* can be divided into two parts:

1. installing the software and its dependencies
2. submitting the *bioGUI* template

For the first part, first the command line arguments are collected and saved. The order of the command line arguments is specified as

1. installation directory (PROGDIR)
2. **sudo** password (if not supplied, installing dependencies is allowed to fail)
3. [0, 1] for adding software binary to $PATH.
4. IP to send template to
5. PORT to send template to

A typical template is then structured as follows:

```
#! Install hisat2-2.0.5
```

The text after the shebang is shown as title in the *Install template Module*.

```
if [ ! "$2" = "" ]; then

    if [ "$(uname)" == "Darwin" ]; then

        echo "Installing brew gcc"
        brew install gcc

    else

        echo "No dependencies"
        echo $2 | sudo -S apt-get update
        echo $2 | sudo -S apt-get install build-essential
    fi

else
    echo "No sudo password, not installing dependencies"
fi
```

If a **sudo** password is supplied, dependencies are installed. This must be compatible with Ubuntu's aptitude, as this is what WSL runs on. Using the *uname* switch, *bioGUI* also supports Mac OS and *brew*, for instance.

Since a lot of harm can be done using the super-user account, install modules are manually curated after submission.

```
## set all variables
PROG=hisat2-2.0.5

PROGDIR=$1
if [ -z "${PROGDIR}" ]; then
    PROGDIR=~/bioGUI/progs/
fi

PROGDIRESC=$(echo $PROGDIR | sed 's/ /\\ /g')

## create progdir
mkdir -p "$PROGDIR"
```

Next some variables need to be set up. It showed of great benefit to create a $PROG variable containing the application and version. The $PROGDIR variable contains the installation path, which is also checked to exist. For several reasons it is also a good idea to have an escaped version of the install path by hand.

```
#download and unzip
if [ ! -f "$PROGDIR/$PROG.zip" ]; then
    wget ftp://ftp.ccb.jhu.edu/pub/infphilo/hisat2/downloads/hisat2-2.0.5-source.zip -
↪O "$PROGDIR/$PROG.zip"
fi

if [ ! -d "$PROGDIR/$PROG" ]; then
    cd "$PROGDIR"
    unzip $PROG.zip -d "$PROGDIR/"
fi
```

Before actually installing the program, make sure to download and unzip/untar the application source code. For reasons of parsimony, this is only done when the expected file or folder does not exist.

```
#install prog
cd "$PROGDIR/$PROG"
```

(continues on next page)

(continued from previous page)

```
make

if [ $# -gt 2 ]; then

    if [ "$3" = "1" ]; then

        if ! grep -q "$PROGDIRESC/$PROG" ~/.bashrc; then
            echo "export PATH=\"$PROGDIRESC/$PROG:\$PATH\" " >> ~/.bashrc;
        fi
    fi

fi
```

After downloading and unzipping, the application can be built in the target directory. Finally, if wanted, the path to the application's executable is added to the $PATH variable.

Certain programs may need some fixes to work properly on Mac OS, Linux or WSL. This is the place where such fixes could go.

Finally we can send the template to *bioGUI*, if an IP address and port have been specified:

```
if [ $# -eq 5 ]; then

IP=$4
PORT=$5

NCCMD=""

if [ "$(uname)" == "Darwin" ]; then
    NCCMD="nc -c $IP $PORT"
else
    NCCMD="nc -q 0 $IP $PORT"
fi


$NCCMD <<EOF

<template description="hisat2 2.0.5 aligner" title="hisat2 2.0.5">
    ...
    <const id="bindir">${PROGDIR}/${PROG}/</const>
    ...
    <execute program="hisat2" param="\${cl}" location="\${bindir}" exec="hisat2" wsl=
↪"WSLsel">
        <output type="COUT" color="green" to="outputstream1" />
        <output type="CERR" color="red" to="outputstream2" />
    </execute>

</template>
EOF

fi
```

Make sure to use `nc` to send the content back to *bioGUI*. Unfortunately the nc-programs differ on Mac OS and Ubuntu, hence the command must be altered according to the underlying OS.

In order to customize the template inbetween the `EOF`, bash variables to be replaced must be written as `${var-name}`. This conflicts with how *bioGUI* expects variables. Therefore, make sure to escape the backslaash where you want to access variables in the *bioGUI* template!

4.2 bioGUI Templates

bioGUI templates consist of two parts: the `<window>`-part which defines the visual appearance and the `<execute>`-part which defines how the command line arguments of an application are assembled from the graphical input elements.

4.3 Visual Model of Templates

4.3.1 Layouts

There exist three different layouts in *bioGUI*:

1. horizontal
2. vertical
3. grid

Layouts may have either visual components as child, or further layouts. However a visual component may only have one layout child and this must be the first child.

For instance, the *cols* attribute for the *grid* layout tells how many columns are needed. If only one attribute is specified, the other attribute is calculated from the number of children and the given attribute.

Layout Components

node name	**allowed attributes**
<GRID>	[cols, rows]
<HGROUP>	[]
<VGROUP>	[]

4.3.2 Visual Elements

In contrast to layout components, visual elements are direct input elements.

Graphical Components

node name	allowed attributes
<ACTION>	[program]
<CHECKBOX>	[selected, selectonwindows, value]
<COMBOBOX>	[selected]
<COMBOITEM>	[value]
<FILEDIALOG>	[filter, folder, location, multiples, multiples_delim, output]
<FILELIST>	[height, title, width]
<FILESELECTBOX>	[delim, filter, location]
<GROUP>	[height, title, width]
<GROUPBOX>	[multi]
<IMAGE>	[height, src, width]
<INPUT>	[multi, type {string, int, float, password}]
<LABEL>	[link]
<RADIOBUTTON>	[value]
<SLIDER>	[max, min, step]
<SLIDERITEM>	[display, value]
<STREAM>	[height, title, width]
<STREAMBOX>	
<WINDOW>	[height, title, width]

4.4 Execution Model of Templates

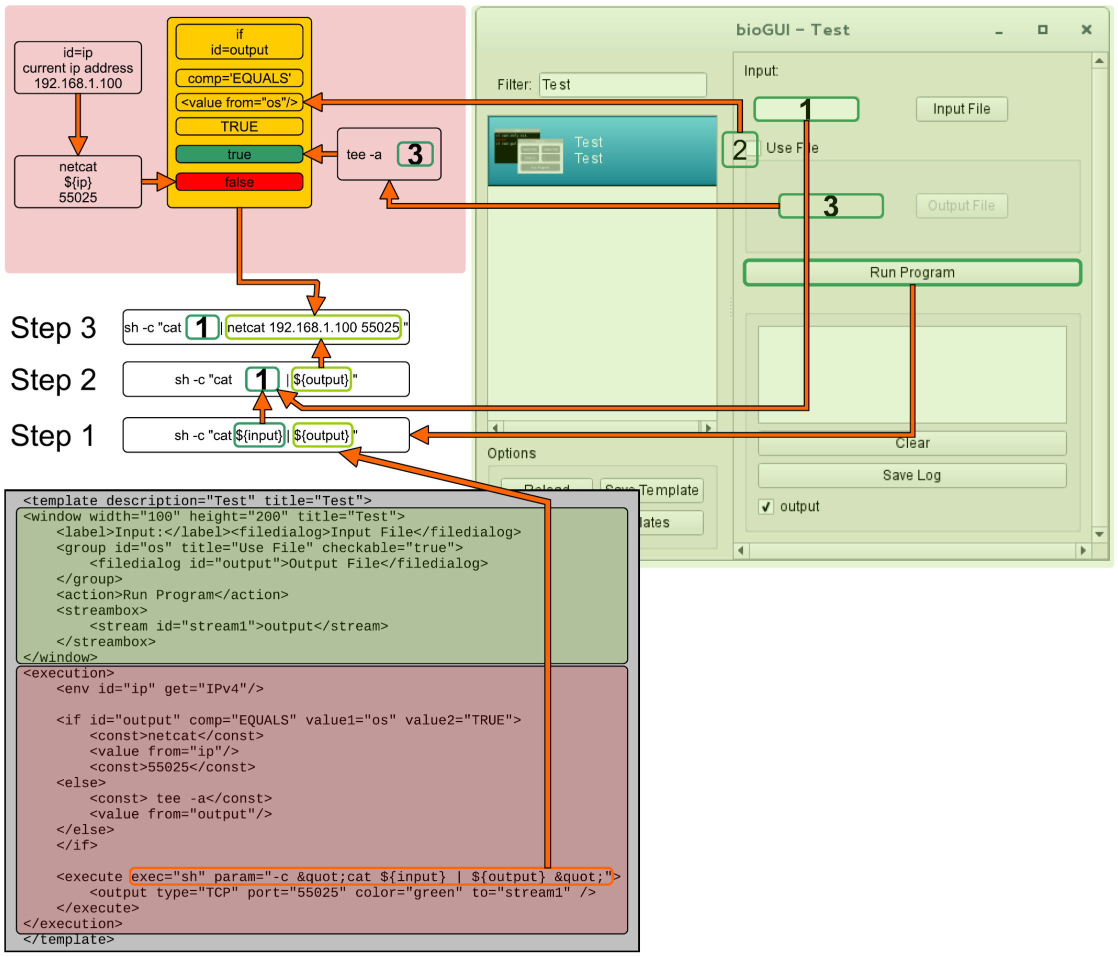
4.4.1 Execution Network

Within a *bioGUI* template, the `<execution>...</execution>` part defines how the command line argument to be executed is assembled. The idea is again based on a network of predefined nodes. The nodes can either be visual components, accessed by their respective **id**, or *Execution Nodes* .

Upon starting an application with *bioGUI*, the execution network is responsible to construct the command line arguments with which the target application is called. Therefore, all executable nodes in the `<execution>` part are searched and *evaluated* one after the other (if there exist several). Since execution must be started via an `action` visual element, which can have a *program* attribute, this allows to specify which executable nodes are executed: if the `program` attribute is set, this must match with the `program` attribute of the executable node.

Finally an executable node is executed. Upon this the command line arguments are assembled. This is shown exemplarily in the below figure:

List of available execution nodes:



```
<template description="Test" title="Test">
<window width="100" height="200" title="Test">
    <label>Input:</label><filedialog>Input File</filedialog>
    <group id="os" title="Use File" checkable="true">
        <filedialog id="output">Output File</filedialog>
    </group>
    <action>Run Program</action>
    <streambox>
        <stream id="stream1">output</stream>
    </streambox>
</window>
<execution>
    <env id="ip" get="IPv4"/>

    <if id="output" comp="EQUALS" value1="os" value2="TRUE">
        <const>netcat</const>
        <value from="ip"/>
        <const>55025</const>
    <else>
        <const> tee -a</const>
        <value from="output"/>
    </else>
    </if>

    <execute exec="sh" param="-c &quot;cat ${input} | ${output} &quot;">
        <output type="TCP" port="55025" color="green" to="stream1" />
    </execute>
</execution>
</template>
```

Fig. 1: Illustration of an *Execution Network* for a simple example application. The command line arguments for the executable `sh` are collected from the nodes with `id` s *input* and *output*. While *input* refers to a visual component node, the *output* id refers to an `if` node, which collects data from another visual component node (**3**) or a node which constructs a *netcat* command, depending on whether the visual node with id *os* (**2**) equals `TRUE` or not.

Execution Nodes

node name	allowed attributes
<add>	[ID, TYPE, sep]
<const>	[ID, TYPE]
<else>	[]
<env>	[GET, ID, TYPE]
<execute>	[EXEC, ID, PROGRAM, TYPE, location, param, program, wsl]
<update>	[deferred, target, attrib, value]
<messagebox>	[deferred]
<file>	[FROM, ID, SEP, TO, TYPE]
<httpexecute>	[CL_TO_POST, DELIM, ID, PORT, PROGRAM, TYPE]
<if>	[COMP, ID, SEP, TYPE, VALUE1, VALUE2]
<math>	[ID, OP, TYPE]
<orderedadd>	[FROM, ID, SELECTED, TYPE]
<output>	[COLOR, DEFERRED, FROM, HOST, ID, LOCATION, PORT, TO, TYPE, TYPE]
<relocate>	[FROM, ID, PREPEND, TO, TYPE, UNIX, WSL]
<replace>	[ID, REPLACE, REPLACE_WITH, TYPE]
<script>	[ARGV, ID, SCRIPT, TYPE]
<value>	[FOR, FROM, ID, TYPE]

CHAPTER 5

Developer Guide

It is highly recommend to read the mechanism section *How does bioGUI work?* first. For more information about visual/execution node behaviour, check section *bioGUI Nodes Description*.

5.1 Extending bioGUI

The creation of nodes for both the visual model as well as the execution model is organised by factories.

5.1.1 Window component factory

In order to add new visual components, one must register the constructor for a new visual elements in the Window-ComponentFactory.

Each constructor for new components must extend the WindowWidgetNode class to return `CreatedElement` objects. This class contains the retriever function for the created element (which fetches values from nodes in *bioGUI*) as well as a WidgetFunctionNode. The `WidgetFunctionNode` contains a pointer to the widget, as well as attribute setters. These attribute setters can update specific attributes/properties of the widget, for instance the image to be displayed (*<UPDATE> node*).

5.1.2 Execution node factory

The execution model has no special node factory, but the `XMLParserExecution` serves as such (this may change soon). Here, nodes must be registered. Execution nodes must extend the `ExecutionNode` class. The most important function to implement is the `std::string evaluate(...)` function. This function has three parameters, namely `std::map< std::string, ExecutionNode*>* pID2Node`, `std::map<std::string, std::string>* pInputID2Value` and `std::map<std::string, WidgetFunctionNode*>* pInputID2FunctionWidget`.

`pID2Node` is a map which contains a pointer to the *ExecutionNode* for a given *id*. `pInputID2Value` is a map which maps any visual element (by id) to its value (determined by the retriever function). Finally `pInputID2FunctionWidget` is a map from any visual element (by id) to its WidgetFunctionNode. This map is essential for updating visual elements (e.g. changing the image being displayed).

CHAPTER 6

bioGUI Nodes Description

6.1 Visual Model Nodes

6.1.1 <GROUP> node

```
<group ordered="true" id="orderedgroup" title="Step options">
<grid rows="1" cols="2">

    <group id="slidingwindow_opt" selected="false" title="Sliding Window" checkable=
↪"true">
        <label>Size</label>
        <input id="slidingwindow_size" type="int"/>
        <label>Quality</label>
        <input id="slidingwindow_quality" type="int"/>
    </group>

    <group id="leadingwindow_opt" selected="false" title="Leading Window" checkable=
↪"true">
        <hgroup>
            <label>Quality</label>
            <input id="leadingwindow_quality" type="int"/>
        </hgroup>
    </group>

</grid>
</group>
```

```
<orderedadd id="steps" from="orderedgroup" selected="true">
    <value from="slidingwindow" for="slidingwindow_opt"/>
    <value from="leading" for="leadingwindow_opt"/>
</orderedadd>
```

Each group node can have its own layout, which must be the first and only child of a group node. An *ordered* group node will give its children (visual model nodes) an order, so the user can select in which order something is taken. For this to function, the values must be retrieved using the `orderedadd` node in the exeution model. The *for* attribute must be the id of an element within the ordered group. And the *from* attribute must be the attribute of an exeuction node which contains the value to be written at this position.

A group may also be *checkable*, which means that it has a checkbox. It will return "true" if checked, "false" otherwise. Setting `checked_value` or `unchecked_value`, respectively, allows custom values. Setting `selected="true"` will make it checked right from the beginning.

If `exclusive="true"` is set as attribute, only one child may be selected at a time.

6.1.2 <COMBOBOX> node

A combobox is *checkable*, which means that it has a checkbox. It will return "true" if checked, "false" otherwise. Setting `checked_value` or `unchecked_value`, respectively, allows custom values. Setting `selected="true"` will make it checked right from the beginning.

6.1.3 <STREAMBOX>/<STREAM> node

The `STREAMBOX` is the default output box, where standard out could be written to. In order to function, a streambox must have at least one `STREAM` child with an ID. This stream is then connected with one or multiple *<EXECUTE>/<ACTION> node* .

```
<streambox id="output1">
    <stream id="outputstream1">std out</stream>
    <stream id="outputstream2">err out</stream>
</streambox>
```

6.2 Layout Nodes

6.2.1 <HGROUP>/<VGROUP> node

```
<vgroup>

    <label>First:</label>
    <label>Second: under first</label>

</vgroup>
```

6.2.2 <GRID> node

The *GRID* node layouts its children visual model nodes in a grid. If only rows or columns are specified, the other value is calculated from the number of children.

```
<grid rows="2" cols="2">

    <label>Top Left</label>
    <label>Top Right</label>

    <label>Bottom Left</label>
    <label>Bottom Right</label>
```

(continues on next page)

(continued from previous page)

```
</grid>
```

6.3 Execution Model Nodes

6.3.1 <RELOCATE> node

The relocate node probably is the most useful node of all, at least for templates aiming at Windows Subsystem for Linux/Bash on Ubuntu on Windows enabled templates. If the `relocate` node is used as below, the *windows_location* is transformed from a Windows path, to a UNIX path if the value of the node with ID `WSLsel` is `true`. For example, `C:\files\sample.dat` is transformed into `/mnt/c/files/sample.dat`.

```
    <checkbox id="WSLsel" value="true" selectonwindows="true">run in WSL?</checkbox>
<relocate id="wsl_location" wsl="${WSLsel}" from="${windows_location}"/>
```

Apart from the WSL use-case, the reloacte node can also be used manually, e.g. to change pathes on a remote server.

```
<relocate from="" to=""/>
```

6.3.2 <IF>/<ELSE> node

Using `if` nodes, condition specific evaluation of nodes can be performed. For instance, depending on the state of a checkbox, either one or the other file can be taken as input. An `if` node accepts three *comp*arison modes: ``is_set``, ``equals`` or ``EQUALS``. The first mode evaluates true, if the node reference in attribute *value1* evaluates to any value which is not empty. The other two modes compare the node referance output of *value1* and *value2* and evaluate the `if` part if *value1* and *value2* are equal (`equals`, case sensitive) or are non-case-sensitive equal (`EQUALS`). If the comparison does not return, the `else` part

```
<if value1="id_elem1" comp="is_set">
    <value from="inputfile_1"/>
<else>
    <value from="inputfile_2"/>
</else>
</if>
```

6.3.3 <VALUE> node

The `value` node collects the *value* from the node with the id given in the *from* attribute. If no such node exists, the value is interpreted as *text*. However, the `const` node may be more suitable here.

```
<value from="inputfile_1"/>
```

6.3.4 <ORDEREDADD> node

See *<GROUP> node* .

6.3.5 <ENV> node

The `env` (environment) node returns several system properties, such as IP addresses, the current OS, etc. . If asked for a specific OS, the node may return `true` or `false`. The `DATADIR` returns the path to where applications are stored, such as applications installed via WSL.

```
<env id="envip" get="IP"/>
<env id="..." get="IP|IPv4|IPv6|LINUX|UNIX|MAC|WIN|DATADIR"/>
```

6.3.6 <SCRIPT> node

For highest flexibility, <*script*> nodes can refer to or contain **LUA** code. For instance

```
<const id="node1">some_file.tex</const>

<script argv="${node1},pdf">
<![CDATA[

function evaluate(arg1, arg2)
    return(string.sub(arg1, 0, -3) .. arg2)
end


]]>
</script>
```

would first split all supplied arguments from the script *argv* attribute and resolve those, which refer to another node (indicated by *${nodeid}*). In this case, the node with id *node1* is a constant value of *somefile.tex*. The second argument is also constant text (_pdf_). Therefore the inline script would be called as `evaluate(some_file.tex, pdf)`. The return value is thus *somefile.pdf*.

6.3.7 <EXECUTE>/<ACTION> node

```
    <image id="statimg" src="" width="100" height="100"/>
...
    <action id="with_program" program="python-prog"/>
    <action id="no_program"/>

</window>
<execution>

<execute program="python-prog" exec="python" param="some.py" wsl="${WSLsel}">
    <output type="COUT" color="green" to="outputstream1"/>
    <output type="CERR" color="red" to="outputstream1"/>
    <update deferred="true" target="statimg" attrib="src" value="..."/>
    <messagebox deferred="false">This is shown before program starts.</messagebox>
    <messagebox deferred="true">This is shown when program ended.</messagebox>
</execute>

<execute exec="cowsay" param="hello" wsl="${WSLsel}">
    <output type="COUT" color="green" to="outputstream1"/>
    <output type="CERR" color="red" to="outputstream2"/>
</execute>
```

(continues on next page)

(continued from previous page)

```
</execution>
```

Execution and action node form a unit: the action button uses the exeuction network to execute a program. For instance, the *action* node with id *with_program* has the program attribute set. Thus, only executable nodes with a program attribute set to this value will be executed. Here, only the *python* program will be executed.

In contrast, the action node with id *no_program* has no program attribute set. Therefore, all available executable nodes will be executed. Thus, both the *python* program and the *cowsay* program will be executed.

Executable nodes may have several children. In general, these children may have a `deferred` attribute which means that these nodes are either activated *before* (`deferred="false"`) the executable is started, or *after* (`deferred="true"`).

The **WSL** attribute signals *bioGUI* whether a program should be executed in WSL/Bash on Ubuntu on Windows, or not. If this is set to *true*, the application is executed in WSL on Windows.

<OUTPUT> node

Output nodes transfer information while a program is running. If they are of type `STD`, they transfer both standard console output (`COUT`) as well as standard console error (`CERR`). The output is transported to the specified stream (*<STREAMBOX>/<STREAM> node*) in the given color.

There also exist output nodes of type `TCP`. These must have **host** (from where is information received) and **port** attributes set. Additionally nodes of type `FILE` directly save output to a file.

<UPDATE> node

Using *update* nodes, attributes of visual elements can be updated. This could, for instance, be the location of an image to be displayed (as in the example above).

<MESSAGEBOX> node

Using *messagebox* nodes, message boxes can be created. The text of a message box may also contain node ids (given in brackets `${id}`).

CHAPTER 7

bioGUI Install Modules Example

7.1 Example: hisat2

7.1.1 Dependency/Install Part

This is already covered in section *Install modules*.

7.1.2 Template Part

The template description and title is shown in the left, template selection window of *bioGUI*. The title attribute of the `window` tag is shown as application window title.

```
<template description="hisat2 2.0.5 aligner" title="hisat2 2.0.5">
<window title="hisat2 2.0.5 aligner">
```

All following elements are placed in a vertical layout (therefore from top to bottom). The following group collects the hisat2 index as well as the output file via a file dialog window. Remember to specify whether a file to be selected is input our output, a directory or whether multiple files can be selected (and how they are delimited).

If you know your application can run in WSL/Bash on Ubuntu on Windows, include a checkbox with the `selectonwindows="true"` attribute.

```
<vgroup>

<group title="Alignment Information">
    <checkbox id="WSLsel" value="true" selectonwindows="true">run in WSL?</checkbox>
    <label>hisat2 Index</label> <filedialog id="hisat_index" output="false" folder=
↪"false">Select hisat2 index</filedialog>
    <label>output file</label> <filedialog id="hisat_output" output="true" folder=
↪"false">Select output file</filedialog>
</group>
```

Further input is collected. Not here, that you can either activate *paried end* or *single end* read data. The `exclusive="true"` signals *bioGUI* to only allow one `group` child to be checked.

```
<group title="Input Data" exclusive="true">
    <hgroup>
    <group id="pairedend" selected="false" title="Paired End" checkable="true">
        <filedialog id="hisat_paired_m1" location="1" multiple="true">Mate 1 files</
↪filedialog>
        <filedialog id="hisat_paired_m2" location="2" multiple="true">Mate 2 files</
↪filedialog>
    </group>
    <group id="singleend" selected="true" title="Unpaired reads" checkable="true">
        <filedialog id="hisat_unpaired_reads" location="">Input File 1</filedialog>
    </group>
    </hgroup>
</group>
```

The following group contains several input options where a checkbox is sufficient. For more program control the phred-/report-checkboxes could also be placed inside an *exclusive* group.

```
<group title="Input Options">
    <checkbox id="hisat_input_f">Reads are FASTA files</checkbox>
    <checkbox id="hisat_input_r">Reads are files with one input sequence per line</
↪checkbox>
    <checkbox id="hisat_input_trim5">Trim bases from 5p</checkbox><input id="hisat_
↪input_trim5_bases" type="int"></input>
    <checkbox id="hisat_input_trim3">Trim bases from 3p</checkbox><input id="hisat_
↪input_trim3_bases" type="int"></input>

    <checkbox id="hisat_input_phred33">Qualities are phred+33</checkbox>
    <checkbox id="hisat_input_phred64">Qualities are phred+64</checkbox>

    <checkbox id="hisat_input_max_reports">Search at most x distinct, primary␣
↪alignments for each read</checkbox><input id="hisat_input_max_reports_num" type="int
↪"></input>
</group>

<group title="Alignment Options">
    <checkbox id="hisat_align_nofw">Do not attempt to align unpaired reads to the␣
↪forward reference strand</checkbox>
    <checkbox id="hisat_align_norc">Do not attempt to align unpaired reads to the␣
↪reverse reference strand</checkbox>

    <checkbox id="hisat_align_no-softclip">Disable soft-clipping</checkbox>
    <checkbox id="hisat_align_no-spliced">Disable spliced alignment</checkbox>
    <checkbox id="hisat_align_no-mixed">Disable mixed alignments (if no conc/disc)</
↪checkbox>
    <checkbox id="hisat_align_no-discordant">Do not look for discordant alignments if␣
↪no concordant are available</checkbox>

    <combobox id="hisat_align_mate_orientations">
    <comboitem value="--fr">forward/reverse</comboitem>
    <comboitem value="--rf">reverse/forward</comboitem>
    <comboitem value="--ff">forward/forward</comboitem>
    </combobox>

</group>
```

(continues on next page)

(continued from previous page)

```
<group title="Output Options">
    <checkbox id="hisat_align_no-unaligned">Disable output of unaligned reads</
↪checkbox>
    <checkbox id="hisat_align_reorder">Guarantees output to be in order of input</
↪checkbox>
    <checkbox id="hisat_transcriptome_assembly_stringtie">Report alignments tailored␣
↪for transcript assemblers including StringTie</checkbox>
    <checkbox id="hisat_transcriptome_assembly_cufflinks">Report alignments tailored␣
↪specifically for Cufflinks</checkbox>
</group>

<group title="Performance Options">
    <label>Number of threads</label><input id="hisat_perf_threads" type="int">1</
↪input>
    <checkbox id="hisat_perf_reorder">Maintain order of reads</checkbox>
</group>
```

HISAT allows many option. Sometimes not all are documented or some are used so seldom, that it is not worth to include them as visual checkboxes. A simple input element can serve as container for user-defined command line arguments. Note that this `action` button has the program attribute set!

```
<group title="Advanced">
    <label>Additional command-line flags</label><input id="hisat_adv_specific_options
↪"></input>
</group>

<action program="hisat2">Align!</action>
```

A very important preprocessing step for HISAT is to build the index. Here (the minimal) needed input for building the index is collected. Also the `action` button will only launch the program to build an index.

```
<group title="build index">
        <label>Input Reference</label> <filedialog id="hisat_build_inref" output=
↪"false" folder="false">Select hisat2 index</filedialog>
        <label>hisat2 Index</label> <filedialog id="hisat_build_idx_out" output="true
↪" folder="false">Select hisat2 index</filedialog>
        <action program="hisat2index">Build Index</action>
</group>
```

Finally we need a streambox to collect any (command line) output.

```
<streambox id="output1">
    <stream id="outputstream1">std out</stream>
    <stream id="outputstream2">err out</stream>
</streambox>

</vgroup>
</window>
```

As the visual part has been closed, we need to start the execution part. First a const-node containing the location of the binary is created. This is filled from the install template (note the unescaped `$`).

Then nodes needed for launching the hisat2index program are defined. Since this program is WSL-enabled, any folder/file must be relocated from the windows path to the WSL path. This is done using relocate nodes. Then the command line arguments are assembled using the add node. With the `sep` attribute, the delimiter can be set.

Finally the program is executed and output is redirected to the outputstream nodes.

```
<execution>
    <const id="bindir">${PROGDIR}/${PROG}/</const>

    <relocate id="hisat_build_inref_rel" from="\${hisat_build_inref}" wsl="\${WSLsel}
↪"/>
    <relocate id="hisat_build_idx_out_rel" from="\${hisat_build_idx_out}" wsl="\$
↪{WSLsel}"/>

    <add id="cl_idx" sep=" ">
        <value from="hisat_build_inref_rel"/>
        <value from="hisat_build_idx_out_rel"/>
    </add>

    <execute program="hisat2index" param="\${cl_idx}" location="\${bindir}" exec=
↪"hisat2-build" wsl="WSLsel">
        <output type="COUT" color="green" to="outputstream1" />
        <output type="CERR" color="red" to="outputstream2" />
    </execute>
```

The actual HISAT exeuction more input files are needed, thus more relocations are needed. Note that we also use a *LUA* script here to crop the file extensions from the HISAT index. For a detailled description of the script node, see *<SCRIPT> node*.

```
<relocate id="hisat_paired_m1_rel" wsl="\${WSLsel}" from="\${hisat_paired_m1}"/>
<relocate id="hisat_paired_m2_rel" wsl="\${WSLsel}" from="\${hisat_paired_m2}"/>
<relocate id="hisat_unpaired_reads_rel" wsl="\${WSLsel}" from="\${hisat_unpaired_
↪reads}"/>
<relocate id="hisat_output_rel" wsl="\${WSLsel}" from="\${hisat_output}"/>

<relocate id="hisat_index_rel_raw" from="\${hisat_index}" wsl="\${WSLsel}"/>

<script id="hisat_index_rel" argv="\${hisat_index_rel_raw}">
<![CDATA[

function evaluate(arg1)

    if (string.match(arg1, ".%d.ht2$")) then
        return(string.sub(arg1, 0, arg1:find(".%d.ht2$")-1))
    end

    return(arg1)
end


]]>
</script>
```

Depending on whether *pairedend* or *singleend* data is being used, the input file arguments are assembled:

```
<if id="input_files" sep=" " comp="EQUALS" value1="pairedend" value2="true">
    <const>-1</const>
    <value from="hisat_paired_m1_rel"/>
    <const>-2</const>
    <value from="hisat_paired_m2_rel"/>
<else>
    <const>-U</const>
    <value from="hisat_unpaired_reads_rel"/>
```

(continues on next page)

(continued from previous page)

```
</else>
</if>
```

Then all command line arguments are combined in the `cl` add node. Here checkboxes are masked using if nodes. However, setting the attribute `unchecked_value=""` in the checkbox would have the same effect. Finally the execute node with the program attribute for HISAT is placed.

It is important to remember that only when the `action` button is pressed, this node is activated. Only then the `cl` node is evaluated and all the referenced input nodes are collected and evaluated!

```
<add id="cl" sep=" ">
    <value from="hisat_adv_specific_options"/>
    <const>-x</const>
    <value from="hisat_index_rel"/>

    <value from="input_files"/>
    <const>-S</const>
    <value from="hisat_output_rel"/>

    <if comp="IS_SET" value1="hisat_input_f" >-f</if>
    <if comp="IS_SET" value1="hisat_input_r" >-r</if>

    <if comp="IS_SET" value1="hisat_input_trim5" ><value from="hisat_input_trim5_bases
↪"/></if>
    <if comp="IS_SET" value1="hisat_input_trim3" ><value from="hisat_input_trim3_bases
↪"/></if>

    <if comp="IS_SET" value1="hisat_input_phred33" >--phred33</if>
    <if comp="IS_SET" value1="hisat_input_phred64" >--phred64</if>

    <if comp="IS_SET" value1="hisat_align_nofw" >--nofw</if>
    <if comp="IS_SET" value1="hisat_align_norc" >--norc</if>

    <if comp="IS_SET" value1="hisat_transcriptome_assembly_stringtie">--downstream-
↪transcriptome-assembly</if>
    <if comp="IS_SET" value1="hisat_transcriptome_assembly_cufflinks" >--dta-cufflinks
↪</if>

    <if comp="IS_SET" value1="hisat_input_max_reports" ><value from="hisat_input_max_
↪reports_num"/></if>

    <if comp="IS_SET" value1="hisat_align_no-softclip" >--no-softclip</if>
    <if comp="IS_SET" value1="hisat_align_no-spliced" >--no-spliced</if>
    <if comp="IS_SET" value1="hisat_align_no-mixed" >--no-mixed</if>
    <if comp="IS_SET" value1="hisat_align_no-discordant" >--no-discordant</if>

    <value from="hisat_align_mate_orientations"/>
</add>

<execute program="hisat2" param="\${cl}" location="\${bindir}" exec="hisat2" wsl=
↪"WSLsel">
    <output type="COUT" color="green" to="outputstream1" />
    <output type="CERR" color="red" to="outputstream2" />
</execute>

</execution>
```

(continues on next page)

(continued from previous page)

```
</template>
```

7.2 Example: Trimmomatic

7.2.1 Dependency/Install Part

For the install module, we start with the shebang which gives the module its name:

```
#! Install Trimmomatic 0.36
```

The text after the shebang is shown as title in the *Install Trimmomatic 0.36*. For the install part we remember that this is essentially a bash script. Thus any bash commands will work here.

We first have to ensure that all dependencies are installed. Trimmomatic has only java as dependency. We can install java (openJDK) on Ubuntu and in WSL, however, will ask the user to install JAVA on Mac OS:

```
if [ ! "$2" = "" ]; then

    if [ "$(uname)" == "Darwin" ]; then

        echo "PLEASE INSTALL JAVA PRIOR USING TRIMMOMATIC!"

    else

        echo "Installing dependencies: openjdk 9"
        echo $2 | sudo -S apt-get update
        echo $2 | sudo -S apt-get -y install openjdk-9-jre
    fi

else
    echo "No sudo password, not installing dependencies"
fi
```

Following the dependencies we should set multiple variables. It has shown useful to have the program name in a variable (once for processing, and once for displaying to the user):

```
## set all variables
PROG=trimmomatic_0_36
PROGNICE=Trimmomatic-0.36
```

Apart from the program name we also fetch the install dir as the first parameter given to the install module:

```
PROGDIR=$1

if [ -z "${PROGDIR}" ]; then
    PROGDIR=~/bioGUI/progs/
fi

PROGDIRESC=$(echo $PROGDIR | sed 's/ /\\ /g')
APPBINARYDIR=${PROGDIRESC}/${PROG}/
```

If the PROGDIR (install dir) has not been given (is empty), we set it to a default value. We create an escaped version of the install dir (PROGDIRESC) and save the program directory where we expect the program to reside

(APPBINARYDIR). In the following we create the progam's install directory, download the application and move everything into place:

```
## create progdir
mkdir -p "$PROGDIR"

#download and unzip
echo "Downloading Trimmomatic"
wget http://www.usadellab.org/cms/uploads/supplementary/Trimmomatic/$PROGNICE.zip -O "
↪$PROGDIR/$PROG.zip"

unzip -o "$PROGDIR/$PROG.zip" -d "$PROGDIR/$PROG"
mv "$PROGDIR/$PROG/"*/* "$PROGDIR/$PROG"
```

The user can select to add the program to the system's PATH variable. If this should be done, the third parameter supplied to the install module is a *1*. We add the escaped path to the PATH variable:

```
if [ $# -gt 2 ]; then

    if [ "$3" = "1" ]; then

        if ! grep -q "$PROGDIR/$PROG" ~/.bashrc; then
            echo "export PATH=\"$PROGDIRESC/$PROG:\$PATH\" " >> ~/.bashrc;
        fi
    fi

fi
```

We are almost done. We now want to verify whether the installation has been successful. We determine this by checking whether the install directory is not empty and whether the executable is available at the thought place. If this is not the case we return the content of all affected paths for better debugging possibilities. Additionally we give the user an ERROR message with possible actions.

```
APPBINARYESC="${PROGDIRESC}/${PROG}/"
APPBINARY="${PROGDIRESC}/${PROG}/trimmomatic-0.36.jar"
if [ -z "$(ls -A ${APPBINARYESC})" ] || [ ! -f ${APPBINARY} ]; then
(>&2 echo ${APPBINARYESC})
(>&2 ls ${APPBINARYESC})

(>&2 echo " \n \n \n")

(>&2 echo ${PROGDIRESC})
(>&2 ls ${PROGDIRESC})

    (>&2 echo " \n \n \n")

(>&2 echo ${APPBINARY})
(>&2 ls ${APPBINARY})

(>&2 echo " \n \n \n")

(>&2 echo "ERROR: The application directory is empty after installation.")
(>&2 echo "ERROR: If you experience problems please re-install the software and␣
↪create an issue on https://github.com/mjoppich/bioGUI.")
(>&2 echo "ERROR: For creating the issue, please upload the log.txt file of your␣
↪installation attempt.")

    (>&2 echo " \n \n \n")
```

(continues on next page)

(continued from previous page)

```
else

    echo "${PROG} has been installed into ${APPBINARYESC}"
    echo "Binary location ${APPBINARY}"

fi
```

Now we need to transfer the template to bioGUI. This is done via a TCP connection (because std-out and std-err are already used by the install module). The user's IP address is given as fourth parameter to the install module, the corresponding port as the fifth parameter. Unfortunately netcat behaves differently on virtually every operating system, we must call netcat differently on Mac OS and Ubuntu, to ensure that the connection is closed upon sending the EOF.

```
IP=$4
PORT=$5

NCCMD=""

if [ "$(uname)" == "Darwin" ]; then
    NCCMD="nc -c $IP $PORT"
else
    NCCMD="nc -q 0 $IP $PORT"
fi
```

Finally the template is sent via netcat and the bash EOF feature:

```
$NCCMD <<EOF

<template ...>
   ...
</template>

EOF
```

The following section explains the template

7.2.2 Template Part

The template description and title is shown in the left, template selection window of *bioGUI*. The title attribute of the `window` tag is shown as application window title. We must remember that this template is sent via netcat and the EOF feature of bash. Thus any bash variable (indicated by a leading dollar sign) are replaced. If we need a bioGUI variable in the template (e.g. in the execution network), the dollar sign must be escaped!

```
<template description="Trimming short RNA-Seq Reads" title="$PROGNICE">
    <window title="$PROGNICE">
```

All following elements are placed in a vertical layout (therefore from top to bottom). The following group collects the input files. If the user selected single-end mode, only one input and output file is needed. For paired-end sequencing data, 2 input files and 4 output files are needed.

If your application should be run in WSL/Bash on Ubuntu on Windows, include a checkbox with the `selectonwindows="true"` attribute. The `exclusive="true"` signals *bioGUI* to only allow one `group` child to be checked.

```
<checkbox id="WSLsel" value="true" selectonwindows="true">Run in WSL?</checkbox>
<group title="Method" exclusive="true">
    <hgroup>
        <group id="pairedend" selected="false" title="Paired End" checkable="true">
            <filedialog id="paired_if1" location="1">Input File 1</filedialog>
            <filedialog id="paired_if2" location="2">Input File 2</filedialog>
            <group title="Options">
                <checkbox>Validate Pairs</checkbox>
            </group>
            <filedialog id="paired_of1p" location="" output="true">Output 1P</
↪filedialog>
            <filedialog id="paired_of1u" location="" output="true">Output 1U</
↪filedialog>
            <filedialog id="paired_of2p" location="" output="true">Output 2P</
↪filedialog>
            <filedialog id="paired_of2u" location="" output="true">Output 2U</
↪filedialog>
        </group>
        <group id="singleend" selected="true" title="Single End" checkable="true">
            <filedialog id="single_if1" location="">Input File 1</filedialog>
            <filedialog id="single_of1" location="" output="true">Output</filedialog>
        </group>
    </hgroup>
</group>
```

Further options are now collected. Trimmomatic is a special case, because the order of the parameters can alter the result. We thus need an ordered group (ordered=true). The order can then later, in the execution network, be retrieved. For a better visual appearance, elements are arranged in a 3x3 grid.

```
<group ordered="true" id="orderedgroup" title="Step options">
    <grid rows="3" cols="3">
        <group id="adapters_sel" title="Adapters" checkable="true">
            <filelist id="illuminaclip_auto_file" allowempty="true" path="$
↪{APPBINARYDIR}/adapters/" ext="*.fa"/>
            <filedialog id="illuminaclip_man_file" location="">Adapter Sequence</
↪filedialog>
            <label>Seed Mismatches</label>
            <input type="int" id="illuminaclip_seed">2</input>
            <label>Palindrome Clip Threshold</label>
            <input type="int" id="illuminaclip_palin">30</input>
            <label>Simple Clip Threshold</label>
            <input type="int" id="illuminaclip_simple">10</input>
        </group>
        <group id="slidingwindow_opt" selected="false" title="Sliding Window"␣
↪checkable="true">
            <label>Size</label>
            <input id="slidingwindow_size" type="int"/>
            <label>Quality</label>
            <input id="slidingwindow_quality" type="int"/>
        </group>
        <group id="leadingwindow_opt" selected="false" title="Leading Window"␣
↪checkable="true">
            <hgroup>
                <label>Quality</label>
                <input id="leadingwindow_quality" type="int"/>
            </hgroup>
        </group>
```

(continues on next page)

(continued from previous page)

```
        <group id="trailingwindow_opt" selected="false" title="Trailing Window"␣
↪checkable="true">
            <hgroup>
                <label>Quality</label>
                <input id="trailingwindow_quality" type="int"/>
            </hgroup>
        </group>
        <group id="crop_opt" selected="false" title="Crop" checkable="true">
            <hgroup>
                <label>Length</label>
                <input id="crop_length" type="int"/>
            </hgroup>
        </group>
        <group id="headcrop_opt" selected="false" title="Headcrop" checkable="true">
            <hgroup>
                <label>Length</label>
                <input id="headcrop_length" type="int"/>
            </hgroup>
        </group>
        <group id="minlen_opt" selected="false" title="Min Len" checkable="true">
            <hgroup>
                <label>Length</label>
                <input id="minlen_length" type="int"/>
            </hgroup>
        </group>
        <group title="PHRED base">
            <combobox id="phred" selected="phred64">
                <comboitem>phred33</comboitem>
                <comboitem>phred64</comboitem>
            </combobox>
        </group>
    </grid>
</group>
<group title="Options">
    <checkbox>Quiet Mode</checkbox>
</group>
<group title="System Settings">
    <label title="example 2">Threads</label>
    <input id="threads_opt" hint="Amount of Threads">2</input>
</group>
```

The remaining part of the GUI template are output options. The streambox captures stdout and stderr output from the launched processes and shows this to the user. The action element is a button which starts the execution of the execution network (or a specific program if specified).

```
    <streambox id="output1">
        <stream id="outputstream1">OS1</stream>
        <stream id="outputstream2">OS2</stream>
    </streambox>

    <action id="runprog" hint="click here to run program">run program</action>
</window>
```

We have now finished the visual part (closed window element) and need to start the execution part. This part Unfortunately is relatively bulky, because of the many input/output files.

```
<execution>
    <if id="illuminaclip_file_tmp" value1="illuminaclip_auto_file" comp="is_set">
        <value from="illuminaclip_man_file"/>
        <else>
            <value from="illuminaclip_man_file"/>
        </else>
    </if>
    <if id="illuminaclip_file" comp="EQUALS" value1="WSLsel" value2="true">
        <relocate from="\${illuminaclip_file_tmp}" wsl="true"/>
        <else>
            <value from="illuminaclip_file_tmp"/>
        </else>
    </if>

    <relocate id="paired_if1_rel" from="\${paired_if1}" sep=" " wsl="\${WSLsel}"/>
    <relocate id="paired_if2_rel" from="\${paired_if2}" sep=" " wsl="\${WSLsel}"/>
    <relocate id="paired_of1p_rel" from="\${paired_of1p}" sep=" " wsl="\${WSLsel}"/>
    <relocate id="paired_of1u_rel" from="\${paired_of1u}" sep=" " wsl="\${WSLsel}"/>
    <relocate id="paired_of2p_rel" from="\${paired_of2p}" sep=" " wsl="\${WSLsel}"/>
    <relocate id="paired_of2u_rel" from="\${paired_of2u}" sep=" " wsl="\${WSLsel}"/>
    <relocate id="single_if1_rel" from="\${single_if1}" sep=" " wsl="\${WSLsel}"/>
    <relocate id="single_of1_rel" from="\${single_of1}" sep=" " wsl="\${WSLsel}"/>
```

The relocate nodes are only used within WSL, when the Windows file path must be translated into the Unix one. For all other Operating Systems the input path is maintained and no changes are made.

```
<add sep=":" id="illuminaclip">
    <const>ILLUMINACLIP</const>
    <value from="illuminaclip_file"/>
    <value from="illuminaclip_seed"/>
    <value from="illuminaclip_palin"/>
    <value from="illuminaclip_simple"/>
</add>
<add sep=":" id="slidingwindow">
    <const>SLIDINGWINDOW</const>
    <value from="slidingwindow_size"/>
    <value from="slidingwindow_quality"/>
</add>
<add sep=":" id="leading">
    <const>LEADING</const>
    <value from="leadingwindow_quality"/>
</add>
<add sep=":" id="trailing">
    <const>TRAILING</const>
    <value from="trailingwindow_quality"/>
</add>
<add sep=":" id="crop">
    <const>CROP</const>
    <value from="crop_length"/>
</add>
<add sep=":" id="headcrop">
    <const>HEADCROP</const>
    <value from="headcrop_length"/>
</add>
<add sep=":" id="minlen">
    <const>MINLEN</const>
    <value from="minlen_length"/>
</add>
```

Using the above add-nodes all options are assembled. For each possible option, one add-node fetches the user supplied parameters.

Using the orderedadd-node we can assemble these parameters in the order specified by the User in the GUI. Using the selected attribute, only selected options are added together. Note that the for-attribute in the value nodes is the ID within the ordered group of the GUI template.

```
<orderedadd id="steps" from="orderedgroup" selected="true">
    <value from="illuminaclip" for="adapters_sel"/>
    <value from="slidingwindow" for="slidingwindow_opt"/>
    <value from="leading" for="leadingwindow_opt"/>
    <value from="trailing" for="trailingwindow_opt"/>
    <value from="crop" for="crop_opt"/>
    <value from="headcrop" for="headcrop_opt"/>
    <value from="minlen" for="minlen_opt"/>
</orderedadd>
```

Finally the input files are added together, as well as the remaining general settings.

```
<add sep=" " id="files_pe">
    <const>PE</const>
    <value from="paired_if1_rel"/>
    <value from="paired_if2_rel"/>
    <value from="paired_of1p_rel"/>
    <value from="paired_of1u_rel"/>
    <value from="paired_of2p_rel"/>
    <value from="paired_of2u_rel"/>
</add>
<add sep=" " id="files_se">
    <const>SE</const>
    <value from="single_if1_rel"/>
    <value from="single_of1_rel"/>
</add>
<if id="files" value1="pairedend" comp="EQUALS" value2="true">
    <value from="files_pe"/>
    <else>
        <value from="files_se"/>
    </else>
</if>
<if id="threads" value1="threads_opt" comp="is_set">
    <value from="threads_opt"/>
    <else>
        <const>1</const>
    </else>
</if>
```

The remaining work to be done is to assemble all input arguments: files, settings and the steps to be executed. This outcome can be accessed via the add node with id cl.

```
<add sep=" " id="cl">
    <value from="files"/>
    <add sep="">
        <const>-threads </const>
        <value from="threads"/>
    </add>
    <add sep="">
        <const>-</const>
        <value from="phred"/>
```

(continues on next page)

(continued from previous page)

```
    </add>
    <value from="steps"/>
</add>
```

Finally the execute node defines what program is executed. The location contains the path to the exec-utable, to which the param-eters are passed, which are here referenced from the node with id cl. The output nodes within the execute environment define where the STDOUT and STDERR output from the process should be displayed (which is the outputstream1/2 here, as defined in the GUI template). Developers can also add deferred nodes here, which are executed as soon as the program has terminated. This might be useful to move files, open/show files, etc.

```
        <const id="bindir">${PROGDIR}/${PROG}/</const>
        <execute location="" exec="java" param="-jar &quot;\${bindir}/trimmomatic-0.
↪36.jar&quot; \${cl}" wsl="WSLsel">
            <output type="COUT" color="green" to="outputstream1" />
            <output type="CERR" color="red" to="outputstream2" />
        </execute>

    </execution>

</template>
```

CHAPTER 8

Licences

8.1 bioGUI & Qt

bioGUI is licenced under GPL v3 (also due to its Qt dependancy). Thus for *bioGUI* and *Qt* the following GNU General Public License version 3 is relevant.

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

```
    b) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by a
    written offer, valid for at least three years and valid for as
    long as you offer spare parts or customer support for that product
    model, to give anyone who possesses the object code either (1) a
    copy of the Corresponding Source for all the software in the
    product that is covered by this License, on a durable physical
    medium customarily used for software interchange, for a price no
    more than your reasonable cost of physically performing this
    conveying of source, or (2) access to copy the
    Corresponding Source from a network server at no charge.

    c) Convey individual copies of the object code with a copy of the
    written offer to provide the Corresponding Source.  This
    alternative is allowed only occasionally and noncommercially, and
    only if you received the object code with such an offer, in accord
    with subsection 6b.

    d) Convey the object code by offering access from a designated
    place (gratis or for a charge), and offer equivalent access to the
    Corresponding Source in the same way through the same place at no
    further charge.  You need not require recipients to copy the
    Corresponding Source along with the object code.  If the place to
    copy the object code is a network server, the Corresponding Source
    may be on a different server (operated by you or a third party)
    that supports equivalent copying facilities, provided you maintain
    clear directions next to the object code saying where to find the
    Corresponding Source.  Regardless of what server hosts the
    Corresponding Source, you remain obligated to ensure that it is
    available for as long as needed to satisfy these requirements.

    e) Convey the object code using peer-to-peer transmission, provided
    you inform other peers where the object code and Corresponding
    Source of the work are being offered to the general public at no
    charge under subsection 6d.

A separable portion of the object code, whose source code is excluded
from the Corresponding Source as a System Library, need not be
included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any
tangible personal property which is normally used for personal, family,
or household purposes, or (2) anything designed or sold for incorporation
into a dwelling.  In determining whether a product is a consumer product,
doubtful cases shall be resolved in favor of coverage.  For a particular
product received by a particular user, "normally used" refers to a
typical or common use of that class of product, regardless of the status
of the particular user or of the way in which the particular user
actually uses, or expects or is expected to use, the product.  A product
is a consumer product regardless of whether the product has substantial
commercial, industrial or non-consumer uses, unless such uses represent
the only significant mode of use of the product.

"Installation Information" for a User Product means any methods,
procedures, authorization keys, or other information required to install
and execute modified versions of a covered work in that User Product from
a modified version of its Corresponding Source.  The information must
```

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

8.2 LUA

This project requires a static LUA library to build. For the release of this software, LuaBinaries static liblua53.a was used and requires the following copyright notice to be included:

8.3 OpenSSL

This project also requires a static OpenSSL library to build. For the release of this software, OpenSSL static libraries were used and require the following copyright notice to be included:

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/) This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com).

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)