SNAPGENE SERVER MANUAL

v1.1.39

Requirements

The snapgene server package was developed and tested on Ubuntu Server 14.04 LTS. Ubuntu 15+ and Fedora 23+ are also tested and supported.

Installation

Install using the .deb or .rpm package from the command line using "sudo dpkg -i snapgene-server_1.0.1.deb" (or whatever version is being installed). Likewise for rpm packages.

The installation directory is /opt/gslbiotech/snangene-server. For this document we will refer to this directory as $EXEDIR.

New versions can be installed directly over old versions as long as the server is not running. Run $EXEDIR/stop before installing to ensure the server is completely shutdown first.

Other prerequisites

See script $EXEDIR/install/install_ubuntu and install_fedora for the libraries and other changes required to run the server successfully.

Snapgene server uses the zmq library for interprocess and intra-net communication within a server environment. The snapgene-server executable is written in C++ and the libzmq.so.1 shared library is installed automatically into the executable directory. The server also makes use of python scripts that require the python zmq package to be installed. The Debian package is named "python-zmq".

Snapgene server is capable of running as a separate restricted system user. The installation assumes the name "snapuser". See bash script "install_ubuntu" for the command line to create the user. The script also creates a group named "snapuser" but this can be changed by IT to aid file permissions when making file specific requests to the server. The name "snapuser" can be changed too as long as the ss-start script is updated as well.

Snapgene server requires some X window system to generate SVG maps and make font measurements. Headless servers typically do not have xserver installed. To fix this xvfb can be installed which is a lightweight virtual replacement for xserver. See install_ubuntu for the typical installation of this. xvfb does not need to startup at boot time because the software uses xvfb-run which starts an isolated xvfb session for a specific executable.

Configuration

Before starting the snapgene server the main configuration file can be modified to control the number of daemons and the TCP ports they open. This file is in /etc/snapgene-server.conf.

Each daemon has a given "server index" from 1 to 100. For each index the tcp port can also be changed. Use enable=0 to disable the server index without removing the entire configuration section from the file.

Operations

Starting and stopping the server

The start and stop scripts are in $EXEDIR/tools with names start and stop respectively. Both require root privileges. When starting a server each daemon instance creates a pid file in /var/run/snapgene-server/x.pid where x is the server index. A log file is also created in /var/log/snapgene-server/x.log where x is the server index.

To start the server the server run: “sudo $EXEDIR/start”

To stop the server run: “sudo $EXEDIR/stop”

Daemons can also be stopped gracefully by sending SIGTERM to the specific process. The sender will need to wait until the currently processing request is completed before this is noticed by the daemon and the shutdown process is commences. On shutdown the specific pid file will be deleted from /var/run/snanpgene-server.

To check if a snapgene server is still running, the user can run "pgrep snangene-server". This will return a list of PIDs of active daemons. To check if each daemon is still processing messages the user can run "$EXEDIR/info". This will send a status request message to the server and wait for a response before returning.

Currently the server is not started at boot time. This is left as a task for IT.

Sending requests to a daemon

Each snapgene server daemon running a single processing loop waiting for requests on their specific TCP port. The port is opened by the ZMQ library and will only handle ZMQ requests. Each ZMQ request is a single text string encoding a single JSON object. The response is also a single text string encoded as a JSON object.

The JSON object is always of the form:

```
{
  "request": "$REQUEST_NAME",
   ...other request parameters...
}
```

where $REQUEST_NAME is the name of the request to be made. Parameters are specific to that request.

The response is always of the form

```
{
  "response": "$REQUEST_NAME",
  "responseCode": $RESPONSE_CODE,
  "responseMessage": "$RESPONSE_MESSAGE",
  "serverIndex": $SERVER_INDEX
  ...other response parameters...
}
```

The response value is always the request name. If the request succeeds then responseCode will be 0. Otherwise a non-zero number is returned which is specific to that response. A responseMessage is also returned which contains a human readable text message for the reason for the response.

The server index is also returned in response messages. This is helpful when dispatching messages to multiple requests and seeing which daemon actually performed the work.

To send a request or multiple requests to a dameon use the script $EXEDIR/tools/request.py command. It can send a single request or multiple commands. Run "./tools/request.py --help" for more details.

Requests can be sent directly from your own php/python servers as well. See the code info-one.py or request.py for examples.

User and Group Permissions

Snapgene server daemons can and are run as a unique user and group. Both and user name and group name are “snapuser”. The user running each daemon is determined in the script /opt/gslbiotech/snapgene-server/tools/start. The final line is:

sudo -u snapuser xvfb-run $BINDIR/snapgene-server.sh --daemon --index=$INDEX

This is the only reference to snapuser and therefore could be changed to a different user if necessary (or launch the daemons using the current user by omitting the “sudo -u snapuser” completely).

Special note about snapuser:
Since deamons run as user “snapuser”, file permissions must be considered when sending requests to daemons that read and write files. For example, generating a map requires the daemon to read a sequence file and to write out several files related to the DOM, CSS, JS, etc. The deamon’s snapuser will need the path to be read and/or write access at the user or group level. Another option is to create a “scratchpad” directory and permit full read and write access with the following commands:

mkdir /tmp/snapgene-scratch
chmod 444 /tmp/snapgene-scratch

and an example of using this to generate a .png map:

/opt/gslbiotech/snapgene-server/tools/request -c ‘{“request”: “generatePNGMap”, “inputFile”: “/opt/gslbiotech/snapgene-server/resources/pIB2-SEC13-mEGFP.dna” “outputPng”: “/tmp/snapgene-scratch/mynewimage.png” }’

Licensing

Snapgene server uses the same license schema as snapgene desktop version with a few differences. The server looks in /etc/snapgene-server/ directory for the required .klf file. Without it, no meaningful requests to the server will succeed (the server will still respond but with error details). The .klf file can expire. The expiration date is returned with status messages (see below).

A python script has been provided to download and install the license file from our servers. The script path is $EXE_DIR/tools/license-utility.py. From this utility you can pull a new license or update an old license using your snapgene provided groupname and registration code.

Snapgene Requests

Below is the list of currently supported commands, their parameters and a description of the command:

Request server status

Inputs Parameters:

request	“status”

Output Parameters:

response	“status”
serverVersion	Version number of the server
requestCount	Number of requests to the server (this request does not count)
lastRequest	Time of the last request (this request does not count)
uptime	Time the server has been running
license	State of the current license (if it is found, valid, expiration, etc.)
licenseExpiration	Date of the license expiration (YY-MM-DD)

Errors:

0	Success

Requests status information for the server. Can be used to "ping" the server and make sure it is still operational. This request doesn't actually change any the server status information (lastRequest, requestCount).

Convert to .dna File

Converts a file from a different DNA format to snapgene native .dna.

Inputs Parameters:

request	“importDNAFile”
inputFile	Full path to file to import
outputFile	Full path to .dna file to write to disk

Output Parameters:

response	“importDNAFile”
code	Return code for the request

Errors:

0	Success
1	Could not open input file
2	File type unsupported
4	Coup not create output file
5	File stamping failer
6	Encoding failed
7	MICA Indexing failed
8	Recognition sequence indexing failed

Convert from .dna File

Input Parameters:

request	"exportDNAFile"
inputFile	Full path to .dna file to read
outputFile	Full path of file to write to disk.
exportFilter	Type of filter to export. Currently supported export filters are listed below.

Output Parameters:

response	"exportDNAFile"
Code	
0	Success
1	Unable to open input file or the file is not a .dna file.
2	Export filter unsupported
3	Internal exporting code failed

Converts a file from snapgene native .dna to another file format. The outputFile must contain the extension .gb or .fasta to convert to that particular format. Otherwise the output format is ambiguous and will the request will fail.

Export Filter Types:

SnapGene DNA Sequence	com.gslbiotech.dna
GenBank - Standard	biosequence.gb
GenBank - SnapGene	biosequence.gbk
FASTA	public.fasta
Text File	public.txt

Detect features

Input Parameters:

request	"detectFeatures"
inputFile	Full path to .dna file to read
outputFile	Full path to .dna file to write to disk
featureDatabase	Full path to feature database file. This should use the .dna file format but often uses the extension .ftrs (either will work)

Ouput Parameters:

response	"detectFeatures"
features	List of features with the same format as the request "reportFeatures"
code	
0	Success
1	Unable to open input file
2	Unable to create output file
3	Feature database file does not exist

Detects feature on a .dna file. Will only detect new features present in the input feature database.

To detect from multiple databases, call this command repeatedly. Input and output parameters can be the same to modify
the file in place.

For each new feature added, and optional note is appended to the feature's /note field (as specified in the genbank file
format. This can also be viewed in Snapgene Desktop in the /note field at the bottom of the edit features dialog box.

Report features

Input Parameters:

request	reportFeatures
inputFile	Full path to .dna file to read

Ouput Parameters:

response	reportFeatures
features	Array of features
code	
0	Success
1	Unable to open input file

Returns the list of all features in the inputFile.

The feature object is of the following format:

name	name of the feature
type	type of the feature (there are many--I don't have them all listed here)
rangeBegin	beginning base pair of the entire feature
rangeEnd	ending base pair of the entire feature
direction	can be one of the following: nondirectional, forward-directional, reverse-directional, bidirectional
segments	array of segments, introns are not included.
attributes	object with many key/value pairs, they are not all listed here. Values are string arrays.

The segment object is of the following format:

name	name of the segment
rangeBegin	beginning base pair of the segment
rangeEnd	ending base pair of the segment
color	color of the segment in #AAAAAAAA format
type	can be one of the following: standard, gap, run-on-translation

Generate SVG Map

Inputs Parameters:

request	“generateSVGMap”
inputFile	full path to .dna file to generate map from
linear	true for linear map, false for circular map
showEnzymes	True to show enzymes on the map
showPrimers	True to show primers on the map
showFeatures	True to show features on the map
showORFs	True to display ORFs on the map
designSize	set the requested size of the map in "width x height". The actual size might be different because the map generation code performs extra logic to ensure the text graphical elements are all visible
outputSvgDom	file to text fragment of the svg objects and tooltip DOM objects
outputSvgJSStatic	file to text fragment of the javascript code required for hover/selection (static part) the static part can be shared for all maps for the same version of snapgene server.
outputSvgJSDynamic	file to text fragment of the javascript code required for hover/selection (dynamic part)
outputSvgCss	file to text fragment of the css code required for proper rendering of the SVG DOM
optimize	Removes extra SVG attributes that are not necessary for correctly rendering the SVG.
dataModel	User defined attribute used to identify a particular map or sequence view. Useful when supporting multiple maps or sequences on the same page

dataId	User defined attribute used to identify a particular map or sequence view. Useful when supporting multiple maps or sequences on the same page.
title	Defines the title to use for the map. If the title is not defined or set to an empty string, then the default title (the filename without the extenion) is used. To clear the title completely from display, set the value to one space, e.g. “ “.
subtitle	Defines the subtitle to use for the map. If the subtitle is not defined or set to an empty string, then the default title (the number of base pairs) is used. To clear the title completely from display, set the value to one space, e.g. “ “.
addMouseEvents	Adds mouse handlers in the hittest region of the DOM. Default is true.

Output Parameters:

response	“generateSVGMap”
code	Return code for the request

Errors:

0	Success
1	Update to open input for or it isn't a DNA file.
2	DNA file can be opened, requires updating, but couldn't be updated
4	Unable to write SVG file
5	Unable to write JS file
6	Unable to write CSS file
100	Internal Error

Generates a web based map in several fragments.

NOTE: This method used to be called generateMapParts.

Generate PNG Map

Generates a similar map compared to generateSVGMap except the output is PNG instead of Dom, JS and css files.

Input Parameters:

request	"generatePNGMap"
inputFile	full path to .dna file to generate map from
linear	true for linear map, false for circular map
showEnzymes	true to display enzymes
showFeatures	true to display features
showPrimers	true to display primers
showORFs	true to display ORFs
designSize	set the requested size of the map in "width x height". The actual size might be different because the map generation code performs extra logic to ensure the text graphical elements are all visible
outputPng	Output PNG file

Output Parameters:

response	"generatePNGMap"
code	Error code for the request

Error Codes:

0	Success
1	Update to open input for or it isn't a DNA file.
2	DNA file can be opened, requires updating, but couldn't be updated
100	Internal Error

Generate SVG Sequence

Input Parameters:

request	"generateSVGSequence"
inputFile	full path to .dna file to generate a sequence from
showEnzymes	true to display enzymes
showFeatures	true to display features
showPrimers	true to display primers
showTranslation	true to display translations (this is currently unsupported)
outputSvgDom	file to text fragment of the svg objects
outputSvgJSStatic	file to text fragment of the javascript code required for hover/selection (static part) the static part can be shared for all maps for the same version of snapgene server.
outputSvgJSDynamic	file to text fragment of the javascript code required for hover/selection (dynamic part)
outputSvgCss	file to text fragment of the css code required for proper rendering of the SVG DOM
optimize	Removes extra SVG attributes that are not necessary for correctly rendering the SVG.
dataId	User defined attribute used to identify a particular map or sequence view. Useful when supporting multiple maps or sequences on the same page.
addMouseEvents	Adds mouse handlers in the hittest region of the DOM. Default is true.

Output Parameters:

response	"generateSVGSequence"
code	Error code for the request

Error Codes:

0	Success
1	Update to open input for or it isn't a DNA file.
2	DNA file can be opened, requires updating, but couldn't be updated
[[3	Omitted intentionally?]]
4	Unable to write SVG file
5	Unable to write JS file
6	Unable to write CSS file
100	Internal Error

Generates a web based sequence in several fragments.

Import Primers from List

Input Parameters:

request	"importPrimersFromList"
inputFile	full path to input .dna file
inputPrimerFile	full path to json file with primers to import
outputDnaFile	full path to output .dna file with imported primers

Output Parameters:

response	"importPrimersFromList"
code	Error code for the request

Error Codes:

0	Success
1	Unable to open input dna file or it isn't a DNA file.
2	DNA file can be opened, requires updating, but couldn't be updated
3	Unable to open primer file
4	Unable to write output dna file
100	Internal error

Reads a list of primers in a json document and inserts them into a dna file. A new dna file is written.
The primer json document is of the following format:
array of objects of members. Members are:

[Primer Item 1, Primer Item 2, ...]
each primer Item is:

```
{
  "Name":"<primer name>",
  "Sequence":"<primer sequence>",
  "Notes":"<primer notes>"
}
```

Report Primers

Input Parameters:

request	"reportPrimers"
inputDnaFile	full path to input .dna file

Output Parameters:

response	"reportPrimers"
code	error code
primers	json array of primers (see schema below)

Error Codes:

0	Success
1	Unable to open input dna file or it isn't a DNA file.
2	DNA file can be opened, requires updating, but couldn't be updated
100	Internal error

Opens a .dna file and outputs a report in JSON for all primers saved to the .dna file.

JSON Schema for primers:

- color-user friendly name of color for primer
- length-length of sequence (this is always the same as property character length
- molecularWeight-formated string with weight of the primer
- name-name of the primer
- notes-notes for the primer
- percentGC-formated string with the percentage GC for the sequence
- sequence-actual sequence string of the primer
- sites-JSON array of binding sites
    - bindingSiteStart: start of the binding site in BP
    - bindingSiteEnd: ending of the binding site in BP
    - meltingTemperature: degrees centigrade for melting
    - strand: specified which side of the sequence the primer is attached to (“Top” or “Bottom”)

Example output with four primers. One of the primers has two binding sites:


```
{
"code": 0,
```

```
"primers": [
{
"color": "Black",
"length": 39,
"molecularWeight": "12,107.9 Da",
"name": "A206K.REV",
"notes": "<html><body>For introducing the monomerizing A206K
mutation</body></html>",
"percentGC": "56% GC",
"sequence": "GTTGGGGTCTTTGCTCAGCTTGGACTGGGTGCTCAGGTA",
"sites": [
{
"bindingSiteEnd": 2593,
"bindingSiteStart": 2555,
"meltingTemperature": 71,
"strand": "Bottom"
}
]
},
{
"color": "Black",
"length": 39,
"molecularWeight": "11,868.7 Da",
"name": "A206K.FOR",
"notes": "<html><body>For introducing the monomerizing A206K
mutation</body></html>",
"percentGC": "56% GC",
"sequence": "TACCTGAGCACCCAGTCCAAGCTGAGCAAAGACCCCAAC",
"sites": [
{
"bindingSiteEnd": 2593,
"bindingSiteStart": 2555,
"meltingTemperature": 71,
"strand": "Top"
},
{
"bindingSiteEnd": 2593,
"bindingSiteStart": 2555,
"meltingTemperature": 71,
"strand": "Top"
}
]
},
{
"color": "Black",
"length": 36,
"molecularWeight": "10,988.1 Da",
```

```
"name": "SEC13.REV",
"notes": "<html><body>A BamHI site was added, plus two extra bases for
in-frame fusion.</body></html>",
"percentGC": "58% GC",
"sequence": "GCTACTGGATCCCTTTGATCGACTTCGCCAGCGGAC",
"sites": [
{
"bindingSiteEnd": 1640,
"bindingSiteStart": 1610,
"meltingTemperature": 68,
"strand": "Bottom"
},
{
"bindingSiteEnd": 1640,
"bindingSiteStart": 1610,
"meltingTemperature": 68,
"strand": "Bottom"
}
]
},
{
"color": "Green",
"length": 41,
"molecularWeight": "12,624.3 Da",
"name": "SEC13.FOR",
"notes": "<html><body>A KpnI site was added.</body></html>",
"percentGC": "46% GC",
"sequence": "ACGGTAGGTACCATGGCAAGTATAGTTTTCCGTACAAGTCC",
"sites": [
{
"bindingSiteEnd": 719,
"bindingSiteStart": 685,
"meltingTemperature": 64,
"strand": "Top"
}
]
}
],
"response": "reportPrimers"
}
```

Report Enzymes

Input Parameters:

request	"reportEnzymes"
inputFile	full path to input .dna file

Output Parameters:

response	"reportEnzymes"
code	error code
setName	name of the enzyme set
count	number of enzymes in library
enzymes	json array of enzymes (see schema below)

Error Codes:

0	Success
1	Unable to open input dna file or it isn't a DNA file.

For each JSON object in the enzymes JSON array:
id: unique identifier for the enzyme. This should match with the map or sequence.
name: name of the enzyme
Hits: JSON array of hit objects (locations the enzyme cuts the sequence.

Each hit object contains the following:
topCutPosition: top location of the enzyme cut (this is the typical value displayed)
bottomCutPosition: if the cut is not blunt, then the bottom base position will be offset from the top.
methylationSensitive: true or false
Blocked: true or false

Report ORFs

Input Parameters:

request	"reportORFs"
inputFile	full path to input .dna file

Output Parameters:

response	"reportORFs"
code	error code
ORFs	jsonArray of ORF objects.

Error Codes:

0	Success
1	Unable to open input dna file or it isn't a DNA file.

Helpful links

```
zmq    http://zeromq.org/
xvfb   http://www.x.org/archive/X11R7.6/doc/man/man1/Xvfb.1.xhtml
```