Glygen Array Development Notes **Last Update** **Date: 06/17/2022**

## Password Encryption Tool and Encrypted passwords in application.yml and/or application.properties

In order to protect the password that the system is using, all the passwords/secrets in application.yml and/or application.properties files should be encrypted and stored as ENC(……) in the above property files. An example entry:

From application.yml:

spring:

datasource:

url: jdbc:postgresql://localhost:5432/glygen

username: glygen

password: ENC(6fDqWRT97iuR7f1U4akTM918JIiZPKRT)

There is a tool under “util” package, named PasswordGeneratorUtil to generate the encrypted passwords to be put in above file. PasswordGeneraterUtil should be run with two arguments:

PasswordGeneratorUtil password DB

where password is the password that needs to be encypted.

Before running this tool, an environment variable JASYPT\_SECRET needs to be set. This secret is the one that the system will use to decrypt the password while running. Therefore, the value used for JASYPT\_SECRET when generating the encrypted passwords should be the same one as the one that is set as environment variable when glygen array backend system is running.

## Password Encryption for the Glygen array application users

The system comes with an “admin” user (and 3 other special users: cfgdata, ncfgdata and imperialdata) that is imported into the database at initial setup. Their passwords should be encrypted before putting into data.sql file (and data-prod.sql file). The same tool “PasswordGeneratorUtil” can be used to generate such encryption as well. This time it needs to be called with only one argument

PasswordGeneratorUtil password

The resulting password, say xxxxx, should then be used in the data.sql as follows:

insert into users (userid, username, password, enabled, firstname, lastname, email, affiliation, affiliationwebsite, publicflag, logintype) values (1, admin, '{bcrypt}xxxxx’, true, 'Glygen', 'User', 'glycomic@uga.edu', 'UGA', 'uga.edu', true, 'LOCAL');

## Instructions to setup email sending through gmail

<https://plswiderski.medium.com/google-api-authentication-with-oauth-2-on-the-example-of-gmail-a103c897fd98>

This may need to be done each time the gmail password is changed.

Here are the main steps:

Google-API console, create/enable gmail api

Get the client-id and client-secret from there and update them in application.yml. Make sure to encrypt them before putting them into application.yml (see Section Password Encryption Tool and Encrypted passwords in application.yml and/or application.properties)

Using the client-id, create a GET request at <https://accounts.google.com/o/oauth2/v2/auth?client_id=187637922392-nm8r2q89o9gub1ftmuos32coutiumkt1.apps.googleusercontent.com&response_type=code&scope=https://www.googleapis.com/auth/gmail.send&redirect_uri=http://localhost&access_type=offline>

This will redirect to a URL containing the “code” similar to the following:

http://localhost/?code=4/AACbpkMFarNdMwz1qVPV0mWcnfjSt0zMcNcUogSMgr2lcZU2G7qjf7B-f1lmTkhRpfgXFBwxzd9adhvRD1Oymgk#

Extract code from the URL and use it in the next step.

POST to https://www.googleapis.com/oauth2/v4/token?

With body similar to the following:

code=4/0AY0e-g4Gdyok1\_TsG838IbFzeYEIKHEJPGLjqzTlIrvS2BrQk04BkvBafIA8iv9tpAinSQ&client\_id=213261356634-v6Dh325uhmkovn8f3vmd0uf3iir780ij.apps.googleusercontent.com&client\_secret=t4ezxYDEiYAQcm8Bw3Xq5qIh&grant\_type=authorization\_code&redirect\_uri=http://localhost

This will return “access-token” and “refresh-token”. Extract those and put them In application.yml (google:gmail-:access-token).

## Instructions to get API key for glytoucan

In order to access GlyToucan API, the [glygenarray.api@gmail.com](mailto:glygenarray.api@gmail.com) needs to be authorized in GlyToucan and the API key should be placed in application.yml in the appropriate place.

Go to Glytoucan.org, click on “Sign in” and signin with glygenarray.api google account. Then go to the profile page and copy the API key. This API key needs to be encrypted before placing into application.yml (see Section Password Encryption Tool and Encrypted passwords in application.yml and/or application.properties)

## Instructions to generate metadata ontology

There is a tool to generate the ontology file from the excel spreadsheets in the util package, named “MetadataOntologyParser”. This application should be run with an argument to point to the folder containing the latest metadata spreadsheets. The only requirement is that the name of the spreadsheets should match the template names we use in the system. The current template names are as follows:

sample\_template

printer\_template

scanner\_template

slide\_template

data\_processing\_software\_template

image\_analysis\_software\_template

assay\_template

feature\_template

spot\_template

printrun\_template

Therefore, the files in this folder should be renamed to “sample\_template.xlsx” to be used for “sample” template, to “printer\_template.xlsx” for “printer” template and so on.

Example:

java org.glygen.array.util.parser.MetadataOntologyParser /Users/sena/Desktop/metadata-spreadsheets/final

Once the ontology is generated (and the changes are committed to github), the web service to “populate ontology” ([**/api/admin/populateTemplates**](https://glygen.ccrc.uga.edu/ggarray/api/swagger-ui.html#/operations/admin-controller/populateTemplatesUsingPOST)) should be executed (after updating the code on the server from github) by the admin user to update the system’s data with the latest metadata.

## Instructions to generate CFG data on the server (or your local system)

The first step of adding cfg data (from scratch) is to upload the slide layouts from “cfg\_lib.xml” file (using the frontend). You need to choose “add multiple slide layouts” in SlideLayout page and upload cfg\_lib.xml file. After clicking submit, you need to choose the slide layout to upload. Since we currently have metadata and parsers for CFG 5.2 version, upload that one. Once the process is finished, you should see all glycans (530)/linkers (22)/features (609)/block layouts (32) and the slide layout in “contribute” section. If, for some reason, there is an error adding one of the block layouts, then delete the slide layout and upload again. The missing block layout should be added this time (this happens due to 183 Sp8 glycan which cannot be used as a base type glycan – works the second time since there is already another base glycan with the same sequence (8 Sp8 Alditol Version)).

The next step is to load the metadata tables to the postgres database where we would like to have the experiments (your local postgres docker or the server’s). The metadata information can be found in “cfg5.2.sql” in the top-level folder of the backend application. Put this file into your <home>/glygen-array/postgres folder (this is mounted inside the postgres docker container). In order to import the sql file, you need to run isql inside the docker container.

docker exec -it postgres\_postgres\_1 bash

plsql -U glygen -W < /var/lib/postgresql/data/cfg5.2.sql

This should load the tables required for CFG5.2 metadata to the postgres database.

The final step is to run the application located in the “java client” (<https://github.com/senaarpinar/glygenarray-java-client>). The application is named “CFGDatasetApplication.java”. Before running the application, make sure the application.yml contains the correct urls for the backend application to use (local vs. server). This application requires 3 arguments: username password folder. Username and password should be the credentials for “cfgdata” (or any other data users with special role to bypass metadata validation). The folder should be the folder containing sub-folders for the CFG experiments (excel files for processed data, and text files for the raw data where applicable).

Example:

java CFGDatasetApplication cfgdata xxxxx /Users/sena/Desktop/GlycanArrayData/Glygen-CFGData/5\_2

(xxxxx should be the actual password)