Biorels documentation

DESAPHY Jeremy

Table of Contents

[Introduction 4](#_Toc175841930)

[Backend infrastructure – Getting started 5](#_Toc175841931)

[Installation and configuration 5](#_Toc175841932)

[Requirements 5](#_Toc175841933)

[Prerequisite 5](#_Toc175841934)

[Download the code from the repository 5](#_Toc175841935)

[Configuration of the environment 6](#_Toc175841936)

[Environment variables – setenv.sh 6](#_Toc175841937)

[Aliases in setenv.sh 7](#_Toc175841938)

[Sourcing the setenv file 7](#_Toc175841939)

[Preparing the container 8](#_Toc175841940)

[Database Installation 8](#_Toc175841941)

[Biorels configuration – Data sources, Genomes, Proteomes 9](#_Toc175841942)

[Genome selection 9](#_Toc175841943)

[Proteome selection 10](#_Toc175841944)

[Data source selection 11](#_Toc175841945)

[Global option selection 11](#_Toc175841946)

[Generating the user configuration file 14](#_Toc175841947)

[Running Biorels 14](#_Toc175841948)

[Running with SGE Cluster: 14](#_Toc175841949)

[Running on a single CPU – Experimental 14](#_Toc175841950)

[Monitoring BioRels 15](#_Toc175841951)

[Biorels\_monitor output: 15](#_Toc175841952)

[MONITOR\_STATUS 17](#_Toc175841953)

[Deep dive in the infrastructure 18](#_Toc175841954)

[Directory structure 18](#_Toc175841955)

[BACKEND directory architecture 19](#_Toc175841956)

[SCRIPTS & PRIVATE\_SCRIPTS Directory architecture 20](#_Toc175841957)

[PROCESS directory architecture 21](#_Toc175841958)

[PRD\_DATA directory architecture 21](#_Toc175841959)

[Configuration files 22](#_Toc175841960)

[CONFIG\_GLOBAL 22](#_Toc175841961)

[Global variables: 22](#_Toc175841962)

[Links (LINK section) 23](#_Toc175841963)

[Tools (TOOL section) 23](#_Toc175841964)

[CONFIG\_JOB 24](#_Toc175841965)

[Case with taxonomy and gene 25](#_Toc175841966)

[CONFIG\_USER – User configuration file 26](#_Toc175841967)

[Global user configuration 26](#_Toc175841968)

[Processing configuration 26](#_Toc175841969)

[Private configuration 27](#_Toc175841970)

[Uniprot configuration 27](#_Toc175841971)

[3rd party data sources with licenses 28](#_Toc175841972)

[Pubmed configuration 28](#_Toc175841973)

[Promoter configuration 28](#_Toc175841974)

[Proteome configuration 28](#_Toc175841975)

[Genome configuration 29](#_Toc175841976)

[Gene configuration 29](#_Toc175841977)

[Private section and schema 30](#_Toc175841978)

[Developing in Biorels 32](#_Toc175841979)

[Reviewing the fundamentals 32](#_Toc175841980)

[Understanding dependencies 32](#_Toc175841981)

[Adding a program/library to the container 32](#_Toc175841982)

[Adding a Linux package 33](#_Toc175841983)

[Adding a Python library 33](#_Toc175841984)

[Adding a PHP library 33](#_Toc175841985)

[Compiling 33](#_Toc175841986)

[Registering a new data source 35](#_Toc175841987)

[Data source directory 35](#_Toc175841988)

[Job nomenclature 35](#_Toc175841989)

[Adding to CONFIG\_GLOBAL 36](#_Toc175841990)

[Adding to CONFIG\_JOB 37](#_Toc175841991)

[Data source block: 37](#_Toc175841992)

[Reviewing job configuration: 37](#_Toc175841993)

[Concurrent jobs 38](#_Toc175841994)

[For ck\_DATASOURCE\_rel job type 39](#_Toc175841995)

[For dl\_DATASOURCE job type: 39](#_Toc175841996)

[For db\_DATASOURCE job type: 39](#_Toc175841997)

[For prd\_DATASOURCE job type: 40](#_Toc175841998)

[For wh\_DATASOURCE job type: 40](#_Toc175841999)

[For batch jobs 41](#_Toc175842000)

[Adding to CONFIG\_USER 42](#_Toc175842001)

[Adding Shell script 42](#_Toc175842002)

[Generate container shell scripts 43](#_Toc175842003)

[Execute your script 43](#_Toc175842004)

[Additional information 43](#_Toc175842005)

[Molecular entity 44](#_Toc175842006)

[Small molecule standardization 44](#_Toc175842007)

[Counterion definition 45](#_Toc175842008)

[Main Molecule definition 45](#_Toc175842009)

[Molecule entry 46](#_Toc175842010)

[Small molecule as molecular component 46](#_Toc175842011)

[Small molecule as molecular entity 47](#_Toc175842012)

[How to register your molecules 48](#_Toc175842013)

[Database structure 49](#_Toc175842014)

[Database table naming convention 49](#_Toc175842015)

[Database table naming convention. 49](#_Toc175842016)

[Table prefix 49](#_Toc175842017)

[Table Suffix: 51](#_Toc175842018)

[Biorj File format 52](#_Toc175842019)

[Biorj theory 52](#_Toc175842020)

[Biorj rules 53](#_Toc175842021)

[Biorj requirements and unique keys 55](#_Toc175842022)

[Adding rules to Biorj 55](#_Toc175842023)

[Adding a Biorj block 55](#_Toc175842024)

[Update Biorj keys 56](#_Toc175842025)

[Update rules 56](#_Toc175842026)

[Website 58](#_Toc175842027)

[Introduction 58](#_Toc175842028)

[Authentication and security 59](#_Toc175842029)

[Installation and configuration 59](#_Toc175842030)

[Requirements 59](#_Toc175842031)

[Additional configurations & use cases 59](#_Toc175842032)

[Creating a new portal 59](#_Toc175842033)

[Modify WEBSITE\_CONFIG 59](#_Toc175842034)

[Create portal directories 60](#_Toc175842035)

[Define a color for the portal 60](#_Toc175842036)

[Define path to portal 60](#_Toc175842037)

[Troubleshooting 61](#_Toc175842038)

[jobs 61](#_Toc175842039)

[Shell script does not exist. 61](#_Toc175842040)

[Unable to find [JOB\_NAME] in GLB\_TREE 61](#_Toc175842041)

[Supplementary information 62](#_Toc175842042)

[Space requirements for different cases 62](#_Toc175842043)

# Introduction

Welcome to Biorels, an infrastructure that will allow you to take advantage of what the scientific community has best to offer when it comes to data related to biological sciences. The overall objective of Biorels is not only to simplify the access to existing data extracted from scientific publications, patents, clinical trials and many scientific databases but also to create a common ground to share such data, to build on top of it and to expand its capabilities. Biorels is made of two components, the backend infrastructure, and the website. The backend infrastructure has been designed to be flexible enough for scientists to select which data they need and to only store them in the database. We also build it so you can freely develop your code privately. In addition, we developed a website in a modular way to easily visualize that data. The website possesses different level of security and access for enterprise needs.

# Backend infrastructure – Getting started

## Installation and configuration

### Requirements

#### Prerequisite

* Postgres database:
  + Depending on the scope – up to 25Tb of disk space
  + Can be located on AWS or other cloud environments.
* Computing power:
  + SGE Cluster
  + Single CPU (Experimental)
  + Nextflow coming soon
* Singularity container application - [Installation — Singularity container 3.0 documentation (sylabs.io)](https://docs.sylabs.io/guides/3.0/user-guide/installation.html)
* Disk space:
  + depending on the scope – up to 30Tb of disk space
  + More disk space might be necessary if wanting to keep the previous dataset versions

#### Download the code from the repository

We suggest that you clone the BioRels repository from github to get the latest code update and bug fixes.

git clone https://github.com/EliLillyCo/BioRels.git

If you wish to develop your own scripts, you should create your own branch. In the case you wish to publish your branch, a good recommendation would be to name your branch after your lab/organization name.

git checkout -b <branch\_name>

This will download all the code necessary to run the backend infrastructure (BACKEND) and the website (WEBSITE). Once you have downloaded the source code, let’s configure everything.

### Configuration of the environment

#### Environment variables – setenv.sh

The setenv.sh file contains all the shell variables required to properly execute the different scripts. It is in BACKEND/SCRIPT/SHELL/setenv.sh.

The table below provides a list of the different variables that should be modified to fit your infrastructure. In addition, any proxy variables should be defined in this file.

|  |  |
| --- | --- |
| Variable name | Description |
| **REQUIRED:** | |
| TG\_DIR | Root directory path of biorels repository |
| PGPASSWORD | Postgres password |
| PGUSER | Postgres user |
| DB\_PORT | Postgres database port |
| DB\_HOST | Postgres database hostname |
| DB\_NAME | Postgres database name |
| DB\_SCHEMA | Postgres database schema |
| **OPTIONAL:** | |
| SINGULARITY\_BIND | Singularity allows you to mount directories on your host system to directories within your container using bind mounts. This allows you to read and write data on the host system with ease.  More information: [Bind Paths and Mounts — Singularity container 3.0 documentation (sylabs.io)](https://docs.sylabs.io/guides/3.0/user-guide/bind_paths_and_mounts.html)  $SGE\_ROOT must be added to enable cluster job submission. |
| SCHEMA\_PRIVATE | Optional – Name of the private schema  If you want a private schema in parallel to the public schema. |
| **FOR GRID ENGINE:** | |
| SGE\_QMASTER\_PORT | specifies the tcp port on which sge\_qmaster is expected to listen for communication requests |
| SGE\_EXECD\_PORT | The port for sge\_execd is currently set as service. Default: 6435 |
| SGE\_CLUSTER\_NAME | Name of this cluster (used by SMF as a service instance name) |
| DRMAA\_LIBRARY\_PATH | Libdrmaa.so path for the Distributed Resource Management Application API (DRMAA) |
| LD\_LIBRARY\_PATH | Provides location of dynamically linked libraries |
| PATH | Directories to be searched to find a command.  lx-amd64 directory must be added to that path |
| **PROXY (OPTIONAL):** | |
| http\_proxy | Path to your proxy – if any |
| https\_proxy |
| HTTP\_PROXY |
| HTTPS\_PROXY |
| HTTP\_proxy |
| HTTPS\_proxy |

Table 1 – Environment variables

Please review those accordingly. One of THE most critical variables is TG\_DIR as it represents the absolute root path directory of the repository. If you wish to add more variables, please ensure that are also set in the environment file for the singularity contained, located in BACKEND/CONTAINER/env-file.txt. This file serves as a mapping file between setenv and the singularity container to know which environment variables to load in the container.

##### Aliases in setenv.sh

To simplify the code execution, several aliases have been created in setenv.sh

* Biorels\_exe to execute a script with some parameters. Biorels\_exe php test.php
* Biorels\_php is an alias to biorels\_exe to execute php scripts: biorels\_php test.php
* Biorels\_run allows you to get in the container
* Biorels\_monitor run the monitoring script
* Biorels\_python is an alias to biorels\_exe to execute python scripts biorels\_python test.py
* Biorels\_sql is an alias to biorels\_exe to run psql

##### Sourcing the setenv file

Once set, you will need to source the setenv.sh script. To do so, please run the following command from the root directory of the repository:

source BACKEND/SCRIPT/SHELL/setenv.sh

No output should be returned, but you can verify this was executed successfully by printing some of the environment variables set:

echo $TG\_DIR

### Preparing the container

To execute any code within Biorels, you will first need to compile the singularity container that will provide all the necessary third-party packages and tools. If you haven’t already installed singularity, please follow this link: [Installation — Singularity container 3.0 documentation (sylabs.io)](https://docs.sylabs.io/guides/3.0/user-guide/installation.html).

All the configurations, third party packages and tools are defined in biorels.sing.txt, located in $TG\_DIR/BACKEND/CONTAINER.To compile the container, make sure to have the environment variables set by reviewing the section on Environment variables – setenv.sh if you haven’t already. Once done, execute the following commands:

cd $TG\_DIR/BACKEND/CONTAINER

singularity build biorels\_container.sif biorels.sing.txt

The process can take up to 90 minutes (about 1 and a half hour). If all goes well – which it should, you should have a file called biorels\_container.sif in $TG\_DIR/BACKEND/CONTAINER. You can test if your container is successfully compiled by running the following command:

biorels\_php -v

You should have something like this as a result:

PHP 8.3.9 (cli) (built: Jul 5 2024 12:04:09) (NTS gcc x86\_64)

Copyright (c) The PHP Group

Zend Engine v4.3.9, Copyright (c) Zend Technologies

### Database Installation

Now that all the environment variables have been set and sourced, we can now install the database. Please execute the following commands:

cd $TG\_DIR/BACKEND/INSTALL/

biorels\_php run\_install.php

First, it will ask you to confirm that TG\_DIR and DB\_SCHEMA that you have set in setenv.sh are the correct values. Once you confirm, it will read biorels\_public.sql and modify the schema name to your $DB\_SCHEMA value, and then create a file called schema\_ready.sql. This sql file will then be executed to create the schema $DB\_SCHEMA, the tables and sequences necessary for the infrastructure. If you have set SCHEMA\_PRIVATE in setenv.sh file, it will do the same, this time reading biorels\_red.sql to generate private\_schema\_ready.sql that will be executed to create the $SCHEMA\_PRIVATE schema and its tables and sequences.

Following the table generation, the script will then proceed in generating shell files that takes into consideration your container configurations. All scripts in $TG\_DIR/BACKEND/SCRIPT/SHELL/ will be reviewed to ensure the called script exists and a new shell in $TG\_DIR/BACKEND/CONTAINER\_SHELL will be generated so it can be executed within a container.

### Biorels configuration – Data sources, Genomes, Proteomes

We strive at making things simple. For this reason, we created a script that will guide you through the different configuration steps. To run it, please execute the following commands:

cd $TG\_DIR/BACKEND/INSTALL/

biorels\_php prep\_config\_job.php

This script is in 5 distinct steps. You can stop the process at any step to come back to it later as long as you finish the process within the same day. The reason for this is that it will generate a directory named by the current date and will generate in this directory one file per step. Therefore, if you want to:

* Restart the process: delete the directory with the current date
* Restart a finished step: delete the corresponding file.
  + Genome selection: GENOMIC and GENOMIC\_RULESET
  + Proteome selection: PROTEOMES
  + Data sources selection: DATASOURCES
  + Global options: GLOBAL\_OPTIONS
  + Final step: NEW\_CONFIG\_USER

With this process and the generated directory, you can come back to see the options you have selected at any given time.

#### Genome selection

The first step is the selection of genomes:

STEP 1 - Select organisms

Do you want to process genomic assemblies? Y/N.

You will be prompted to choose whether you want to process genome assemblies or not. If not (Answer: N), you can proceed to the next section. If you wish to process genome assembly – answer Y. This will trigger the download of RefSeq and Ensembl list of assemblies, grouped by organisms. You will then be prompted to provide the NCBI Taxonomy ID of the organisms of interest (one at the time).

Please provide the NCBI Taxonomy Identifier for the organism of interest or N to stop.

Note: If don't know it, please search for it here: <https://www.ncbi.nlm.nih.gov/taxonomy>

Example: If you wish to consider Human, type down 9606. It will then provide you with a list of assemblies for this organism. Below is an example of that list – we only present the first few columns to make it more readable. In this instance (December 4th 2023) – there are 3 assemblies available for Homo Sapiens. 2 from RefSeq, GRCh38.p14 which is the Human Reference Genome and T2T-CHM13v2.0 from Telomere 2 Telomere project. Ensembl only offers GRCh38.p14. Each assembly is defined by an ID that you can use to select the assembly. If you want to get GRCh38.p14 from both resources, then type 0,2.

ID SOURCE ASSEMBLY\_ACCESSION ASSEMBLY\_NAME

0 REFSEQ GCA\_000001405.29 GRCh38.p14 GCF\_000001405.40 Homo sapiens latest

1 REFSEQ GCA\_009914755.4 T2T-CHM13v2.0 GCF\_009914755.1 Homo sapiens latest

2 ENSEMBL GCA\_000001405.29 GRCh38.p14 2014-01-Ensembl/2023-03 Human N/A

Please choose among the following assemblies below

If you wish multiple assemblies, list the IDs separated by comma. Example: 1,3

Once done, you will come back to the previous prompt asking to provide the NCBI Taxonomy ID so you can select assemblies from other organisms. When you went through all the organisms, input N to stop.

#### Proteome selection

The second step will focus on selecting proteomes from UniProt. If you don’t wish to consider specific proteomes, type N and move on to the next step.

Do you want to process proteomes? Y/N. Then press return: Y

Please wait while we download the list of proteomes

24004 proteomes listed

Please provide the NCBI Taxonomy Identifier for the organism of interest or N to stop.

Note: If don't know it, please search for it here: https://www.ncbi.nlm.nih.gov/taxonomy

Your value:

The list of proteomes offered by UniProt will then be downloaded and will be searchable by NCBI Taxonomy Id. If you don’t know the corresponding NCBI Tax ID, you can go to <https://www.ncbi.nlm.nih.gov/taxonomy> and search for it.

Your value: 9606

PROTEOME\_ID SUPERREGNUM SPECIES\_NAME

0 UP000005640 eukaryota Homo sapiens (Human)

Please choose among the following proteomes below

If you wish multiple proteomes, list the IDs separated by comma. Example: 1,3

Uniprot usually provides only 1 proteome per organism. However, to remain generic enough, we provide the list of all proteomes and prompt you to select the one(s) you want. If you wish here to select the proteome UP000005640, type 0. You will then come back to the first Proteome prompt asking about the next NCBI Taxonomy ID if you want to consider multiple proteomes. Once done, you can type N to move on to the next step

#### Data source selection

Biorels offers more than 30 data sources, with various levels of integration. The first few lines of the prompt will summarize which data source you will need based on the genome and proteome selection steps. Therefore, in this third step, you will be prompted to select which data sources you want to process. If the data sources selected from the previous steps are enough, you can move forward by typing N/A. If you want them all, type ALL. Otherwise, it will list all available resources. You will then have to type the ones you want, separated by space. Do not worry about dependencies, the script will for it afterward.

Based on genome information and options, you will need the following data sources:

REFSEQ GENOME ENSEMBL UNIPROT

Please provide the list of databases you wish to consider among the list below, each separated by space:

Or N/A if already covered by previous steps

Or ALL if you want all resources

TAXONOMY

GENE

SEQ\_ONTO

GENOME

VARIANT

TRANSCRIPTOME

PUBLI

In the example below, we ask to add the TRANSLATE process, which will try to translate mRNA to protein.

Your choice(s): TRANSLATE

You have selected TRANSLATE, REFSEQ, GENOME, ENSEMBL, UNIPROT

You will need those datasources too: GENE, SEQ\_ONTO, PUBLI, ECO, GO, TAXONOMY

Crudely estimated database size: 268905Mb

Do you want to proceed Y/N. Then press return

From your selection, the process will also identify the necessary dependencies. In the case of UniProt, it will be Taxonomy, ECO ontology, Gene Ontlogy, and Publication. RefSeq/Ensembl will require Taxonomy, Gene and Sequence Ontology. It will then ask you to confirm if this selection is correct or not.

#### Global option selection

If you reached this step, you will have been prompted to select genome assemblies, proteomes and data sources. The script will then proceed in providing you a summary of your current choices:

##################

##################

##################

Summary of current selection:

-> Genomes:

9606 9606 REFSEQ GCA\_000001405.29 GRCh38.p14 GCF\_000001405.40 Homo sapiens latest Patch reference genome 2023/10/02 vertebrate\_mammalian

9606 9606 ENSEMBL GCA\_000001405.29 GRCh38.p14 2014-01-Ensembl/2023-03 Human N/A N/A N/A N/A N/A

-> Proteomes:

9606 UP000005640 9606 HUMAN eukaryota 20596 84119 108303 Homo sapiens (Human) 9606

-> Data source needed based on Genome/Proteome selection:

REFSEQ

GENOME

ENSEMBL

UNIPROT

-> Data source you requested or depends upon:

TRANSLATE REQUESTED

REFSEQ REQUESTED

GENOME REQUESTED

ENSEMBL REQUESTED

UNIPROT REQUESTED

GENE ADDED

SEQ\_ONTO ADDED

PUBLI ADDED

ECO ADDED

GO ADDED

TAXONOMY ADDED

In this example, we have selected 2 Human GRCh38.p14 assemblies, one from RefSeq and one from Ensembl. We also asked to consider the Human Proteome from UniProt and requested TRANSLATE data source. Overall this will configure the system to process 11 data sources: 5 requested and 6 added due to dependencies. You will then be asked a series of question to further customized the infrastructure.

**A private schema name has been set in setenv.sh. Do you want to use it? (Y/N)** – If you have specified SCHEMA\_PRIVATE in setenv.sh, this will ask to confirm that you indeed want to populate that schema. If you answer Y, it will set PRIVATE\_ENABLED to T in CONFIG\_USER.

**Do you want to keep files from previous releases? (Y/N)** – In some instances, such as regulatory purposes, you might be required to keep previous releases. If such case applies to you, type Y.

**Please provide an email address to send issues to:** When a script fails, Biorels will send an email to this email address

**Please provide an email address from which the email will be sent from:** When a script fails, Biorels will send an email using this email address.

**Please provide a prefix for the job names (Default BR\_) - Max 5 letters:** This prefix is used in all job submission to the cluster to identify them from other non biorels jobs.

**ChEMBL: Genes are not required by default for ChEMBL. Do you want to add Gene annotations for ChEMBL records?** If you requested ChEMBL but not GENE, you will be prompted this question. If you choose Y, this will allow to connect Gene annotations to ChEMBL assays.

**X-Ray: Genes are not required by default for X-Ray. Do you want to add Gene annotations for X-Ray records?** If you requested XRAY but not GENE, you will be prompted this question. If you choose Y, this will allow to connect Gene annotations to X-Ray structures.

**DrugBank: Please provide your Drugbank API login (Format: USER:PASSWORD).** DrugBank API isn’t a free resource. Therefore, you will need to provide your own User & password to enable it. Otherwise it will be disabled.

**OMIM: Please provide your OMIM API Key.** OMIM API isn’t a free resource for all. Therefore you will need to provide your own API Key or OMIM will be disabled.

**Uniprot: Do you want to download/process Swiss-Prot? (Y/N)**. Biorels provides the option – on top and independently of the proteomes – to process the whole Swiss-Prot. If you want to do so, Type Y.

**Uniprot: Do you want to download TrEMBL Y/N** . In addition to Swiss-Prot and proteome, Biorels also can download TrEMBL. However, due to the size and high level of changes occurring between releases, it will not process and push it to the database. If you wish to download TrEMBl, Biorels will generate a blastp database from it and will use it to get additional records associated with either the Organisms you’ve selected, ChEMBL records, X-Ray records.

**Pubmed: Do you want to store abstracts? Y/N**. Abstracts can take around 35Gb in the database. If you want to store abstracts, type Y

**Pubmed: Do you want to store citations? Y/N**. Citations will take 2.5Gb.

Twice a week, Biorels will query each gene, drugs, disease, tissue to get the latest papers. Depending on your data sources, you might those questions:

**Pubmed: Do you want to query pubmed for genes? Y/N**

**Pubmed: Do you want to query pubmed for drugs? Y/N**

**Pubmed: Do you want to query pubmed for disease? Y/N**

**Pubmed: Do you want to query pubmed for tissues? Y/N**

API Key can be generated in Pubmed to enable a higher number of queries per second:

**Pubmed: Please provide your Pubmed API Key (N/A if none)?**

**for more information: https://support.nlm.nih.gov/knowledgebase/article/KA-05317/en-us**

**Pubmed API Key:N/A.**

**Gene annotation**

**Based on genomes/proteomes, genes from those taxId will be considered: 9606**

**Do you want to consider other organisms for gene annotations?**

**List taxonomic Identifiers, seperated by space.**

This last question aims at optimizing GENE processing which can take up to 16h if you process the whole gene collections. It will therefore limit the processing to the organisms of interest. The default will be based on your genome, proteome, and data source selection – ChEMBL, SwissProt and X-Ray. If you wish to consider more organisms, you can provide the list of NCBI Taxonomy identifiers.

#### Generating the user configuration file

Thank you for your patience! If you have done all the steps, the process will generate the CONFIG\_USER file in your INSTALL/[CURRENT\_DATE] directory. You can review the different files if you wish. To enable that new CONFIG\_USER file, please execute the prompted command which will look very much like this:

cp ./[DATE]/NEW\_CONFIG\_USER $TG\_DIR/BACKEND/SCRIPT/CONFIG/CONFIG\_USER

This command will copy your newly generated configuration file to Biorels configuration directory. If you wish to get deeper into it, you can go to this section.

### Running Biorels

At this stage, you should have:

* Set up the environment and source setenv.sh
* Created the container
* Created the database and the tables
* Configured the global variables and the data sources you wish to process.

Great! You are ready to run biorels!!!!

#### Running with SGE Cluster:

To do so, nothing’s easier, just run the following command:

biorels\_monitor

**Note:** Biorels\_monitor is an alias to biorels\_php $TG\_DIR/BACKEND/SCRIPT/monitor\_job.php

This will trigger the master job that will look at each job, check if a job meets all criteria and submit it. Thank you for using Biorels – We hope you enjoy it!

#### Running on a single CPU – Experimental

You don’t have a SGE Cluster available or you just want to test it without additional headache, that’s alright. In this case, you will have to go to $TG\_DIR/BACKEND/INSTALL. In that directory, there is a script called gen\_single\_script.php. It will generate a list of shell commands executing the different scripts. ***However you will need php and postgresql installed locally on your machine.***

cd $TG\_DIR/BACKEND/INSTALL

php gen\_single\_script.php > ../commands.sh

And to execute that script:

Cd $TG\_DIR/BACKEND/

sh commands.sh

gen\_single\_script.php will look at your CONFIG\_USER file and the dependencies between the different scripts to generate a list of scripts in the order they are supposed to run. When executing the commands.sh script, it will execute the individual script in their corresponding order, one at the time.

**Limitation:**

* Needs to be manually triggered
* If a script fails for any reason, the next script will still be triggered, resulting in undefined behavior.

**Note:** This process is experimental.

### Monitoring BioRels

A lot will be happening at the beginning since everything needs to be loaded in the database. You will have a few ways to monitor what’s happening.

#### Biorels\_monitor output:

Biorels\_monitor is the main script in charge of running the different scripts. It provides an output that will look like this image below:

A screenshot of a computer program

Description automatically generated

|  |  |
| --- | --- |
| Column | Description |
| A | Header |
| B | Timestamp |
| C | List of failed jobs. Biorels Job ID followed by job name |
| D | Currently running jobs, with their SGE Job ID, BioRels Job ID and job name |
| E | Summary of job running & ended. |
| **Latest job run:** | |
| F | BioRels Job ID |
| G | Job Name |
| H | Working directory in [PRIVATE\_]PROCESS/[JOB\_DIR]/ |
| I | Date that job was processed and generated new data.  A script that checks for a new release and didn’t find any will update run\_date and last\_check\_date but not processed\_date. |
| J | Date that job was ready to run |
| K | Date the job was run |
| L | Run time |
| M | T: Job run successfully. F: failed |
| N | Cause of the job failing |

The output will show the list of currently running jobs, failed jobs and latest job run. However, it does not provide why a job has been triggered or not.

#### MONITOR\_STATUS

Monitor\_status files are located in $TG\_DIR/BACKEND/LOG. When running biorels\_monitor, the script will generate up to 20 versions of that file, one for each iteration of biorels\_monitoring checks. Therefore, MONITOR\_STATUS\_20 will represent a snapshot of the decision making 20 iterations ago.

A screenshot of a computer

Description automatically generated

|  |  |
| --- | --- |
| Column | Description |
| A | Dependency Level |
| B | BioRels JOB ID |
| C | Job name |
| D | Workding directory. -1 if never run |
| E | For level 1, which are triggered on a time basis, shows next submission |
| **Dependency level 2 and above:** | |
| F | Dependency level |
| G | Ruleset. Db\_gene **requires:**  Dl\_gene and wh\_taxonomy to have a newer date than db\_gene date  OR  Pp\_uniprot OR dl\_chembl to have a newer date than db\_gene date  To be triggered. |
| H | Wh\_go as dependency level 3 required db\_pubmed, which is already up to date and ck\_go\_rel. Therefore wh\_go is waiting on ck\_go\_rel to run successfully. |
| I | Wh\_reactome required 4 data sources. 3 are up to date but it is waiting on ck\_reactome\_rel |

Manual changes

##### Disable a script

To disable a script, you will need to:

* open $TG\_DIR/BACKEND/CONFIG/CONFIG\_USER
* locate the script of interest
* replace T by F. (this section is tab delimited)

If you are using an SGE\_CLUSTER, you will have to stop & restart the job monitoring.

If you are running on a single CPU, you will have to regenerate the commands.sh file

Enable a script

To disable a script, you will need to:

* open $TG\_DIR/BACKEND/CONFIG/CONFIG\_USER
* locate the script of interest
* replace T by F. (this section is tab delimited)

If you are using an SGE\_CLUSTER, you will have to stop & restart the job monitoring.

If you are running on a single CPU, you will have to regenerate the commands.sh file

##### Disable a data source

To disable a data source, you have the possible to either disable all scripts of that data source in CONFIG\_USER (see section above) OR run prep\_config\_job.php script again, located in BACKEND/INSTALL/. For the latter, simply do not provide that data source in the list of data sources to process.

##### Enable a data source

To enable a data source, it is strongly recommended to run prep\_config\_job.php script again, located in BACKEND/INSTALL. The main reason is to ensure that all dependencies are also enabled.

Reset a job

If a job has failed for some network issues for instance and you wish to run it again, you have two options:

* Run it manually: biorels\_php SCRIPT\_PATH/SCRIPT\_NAME. This will run your process and the job monitoring will automatically trigger the next scripts if any.
* Modify the database:
* biorels\_sql to run psql and connect to your database
* Run the following query: UPDATE [schema\_name].biorels\_timestamp set processed\_date=null, last\_check\_date=null where job\_name='[YOUR\_JOB\_NAME]'

## Deep dive in the infrastructure

While Biorels is processing the data sources you want, please take some time to understand a little bit more about how the infrastructure works, from its infrastructure to the job definitions.

### Directory structure

For now, we are going to focus on the architecture of the backend infrastructure. There are 4 directories at the root of Biorels. Since WEBSITE is the only one that is not part of the backend infrastructure, please refer to [WEBSITE SECTION]

|  |  |
| --- | --- |
| Directory | Description |
| BACKEND | All the backend data logistic |
| DOCS | Provides the licenses for 3rd party tools and data sources as well as this documentation. |
| PRD\_DATA | Aliases to the latest version of each data source |
| PROCESS | Where all data processing happens. 1 directory per data source |
| PRIVATE\_PROCESS | Where all the private data process happens. 1 directory per data source |
| WEBJOBS | Where executed web jobs can run |
| WEBSITE | Website files |

#### BACKEND directory architecture

The BACKEND directory contains all necessary scripts and configuration files to run the backend infrastructure. Please take a look at the Table 2 below listing the roles of the different directories.

|  |  |
| --- | --- |
| Directory | Description |
| BACKEND | All the backend data logistic |
| |=> CONTAINER | Configuration and environment file to create backend container scripts |
| |=> DEVELOP | **Experimental!** Script to generate new data source scripts |
| |=> INSTALL | SQL files to create schemas and tables |
| |=> LOG | LOG directory providing a status for each script |
| |=> SGE\_LOG | Standard out and error logs for each submitted job |
| |=> MONITOR | Monitoring files: job running/timestamps |
| |=> PRIVATE\_SCRIPT | Directory of all your personal scripts, not to be shared |
| |=> SCRIPT | Contains all the individual job scripts and necessary libraries |
| |=> LIB | Contains php library files |
| |=> LIB\_PYTHON | Contains python library files |
| |=> CONFIG | Configuration files |
| |=> SHELL | Shell wrapper for each script file |
| |=> API | API function calls |
| |=> BIORJ | Import/export |
| |=> monitor\_job.php | Master script |
| |=> CONTAINER\_SHELL | Shell wrapper for each script, to be run using the container |
| |=>SRC | Source code for some of the tools |
| |=> STATIC\_DATA | Predefined data or sets of rule to help in data processing |

Table 2 - Directory structure for Backend infrastructure

Let’s break it down a bit.

* Monitor\_job.php is the conductor of the overall machinery.
* CONTAINER contains the configuration files to build the container, which has all the tools and libraries required to execute the scripts
* INSTALL has all the installation and configuration scripts
* DEVELOP will generate all the files and configuration you’ll need to add a new data source. It’s however experimental
* STATIC\_DATA possess all the static files, i.e. some ruleset or additional mapping information.
* MONITOR contains the list of running scripts so that if you stop monitor\_jobs, it knows what was running
* CONFIG holds the core of the machinery, your configuration file and the jobs configuration
* SCRIPT/PRIVATE\_SCRIPT contains the processing scripts
* CONTAINER\_SHELL will have scripts that are container wrappers of the shell scripts located in SCRIPTS/SHELL
* LOG will have:
  + MONITOR files that is a snapshot of the decision making to run a job as defined by monitor\_jobs
  + SGE\_LOG: log files of the actual scripts.

##### SCRIPTS & PRIVATE\_SCRIPTS Directory architecture

SCRIPTS is the key scientific directory since all the scientific data processing are defined in this directory. Some core directories that are part of Biorels engine, as listed in Table 3. In addition, each data source will have a directory under SCRIPTS, that will contain one to many processing scripts.

|  |  |
| --- | --- |
| Directory | Description |
| API | Scripts for API Database query. See API Section |
| BIORJ | Scripts for Biorj import/export |
| CONFIG | Configuration files. See *Configuration files section* |
| LIB | PHP Library files with functions to enable the processing scripts |
| LIB\_PYTHON | Python Library files with functions to enable the processing scripts |
| SHELL | Shell wrapper for each processing script. Includes setenv.sh |
| WEBJOBS | Specific web jobs that can be run in the backend. See Webjob section |

Table 4 – List of SCRIPTS directories that are part of Biorels engine

#### PROCESS directory architecture

PROCESS directory will be the data core. Every file where being downloaded, being generated or copied over will be saved under PROCESS. Similarly, to SCRIPTS, each data source will have its own directory. Under each data source directory will be all the different versions of that resource, each being stored in a unique subdirectory named either after the version or the date of the download.

For NCBI GENE for instance, the directories will look as such:

$TG\_DIR/PROCESS/**GENE**/2023-11-29/

$TG\_DIR/PROCESS/**GENE**/2023-11-30/

The name of the data source will be defined in red, right under PROCESS, while the date will be created under gene to note the version – in orange.

The same will apply for private scripts where files will be stored in PRIVATE\_PROCESS

#### PRD\_DATA directory architecture

Different customers have different needs, and some of them wants to access to the latest version of a data source easily. PRD\_DATA answers such a need. For each data source, an alias of the same name as the data source will be created – with a link to the latest version of the data. An example of a PRD\_DATA content by executing *ls -l* command:

BIOASSAY\_ONTO -> $TG\_DIR/PROCESS/BIOASSAY\_ONTO/2022-11-16

CLINVAR -> $TG\_DIR/PROCESS/CLINVAR/2023-11-18

EFO -> $TG\_DIR/PROCESS/EFO/2023-11-18

GO -> $TG\_DIR/PROCESS/GO/2023-11-22

REACTOME -> $TG\_DIR/PROCESS/REACTOME/2023-09-14

SWISSLIPIDS -> $TG\_DIR/PROCESS/SWISSLIPIDS/2023-10-12

TRANSLATE -> $TG\_DIR/PROCESS/TRANSLATE/2023-10-16

UBERON -> $TG\_DIR/PROCESS/UBERON/2023-10-28

UNIPROT -> $TG\_DIR/PROCESS/UNIPROT/2023-09-14

We can see in the example above 9 symbolic links representing the latest version for 9 data sources.

### Configuration files

The core of Biorels configuration resides in $TG\_DIR/BACKEND/SCRIPT/CONFIG and is made of 3 files:

* CONFIG\_GLOBAL: The file contains global variables, such as different directory and tool path names.
* CONFIG\_JOB: This file lists the different processes and how/when they can run.
* CONFIG\_USER: This defines the different rules the user has provided during Biorels configuration – Data sources, Genomes, Proteomes step.

#### CONFIG\_GLOBAL

BACKEND/SCRIPT/CONFIG/CONFIG\_GLOBAL contains all the different links, path, parameters, and options that Biorels requires to properly function. Many of them don’t need to be modified unless there is some critical conflict with your environment. However, a few of them should be reviewed before running the system. 3 types of variables are defined in that file: GLOB for global variables, LINK for external links, and TOOL, which can be used to define both tools path and the parameters of those tools.

##### Global variables:

Below are the different global variables located in BACKEND/SCRIPT/CONFIG/CONFIG\_GLOBAL. None of it should be modified by the user during configuration. The format of the Global variable line is as follow:

|  |  |  |  |
| --- | --- | --- | --- |
| Column 1 | Column 2 | Column 3 | Column 4  (optional) |
| GLOB | Link name | Link path | Description |

**Note:** The number of tabs between column does not matter as long as there are at least 3 values – to make it more readable.

|  |  |
| --- | --- |
| Variable name | Description |
| **CORE BIORELS CONFIGURATION** | |
| PROCESS\_DIR | Directory in which downloaded data will be stored – shouldn’t be modified |
| PRIVATE\_PROCESS\_DIR | Directory in which private data will be stored |
| PRIVATE\_SCRIPT\_DIR | Directory where all private scripts are stored – shouldn’t be modified |
| LOG\_DIR | Log directory– shouldn’t be modified |
| MONITOR\_DIR | Monitoring files directory– shouldn’t be modified |
| SCRIPT\_DIR | Script directory – shouldn’t be modified |
| STATIC\_DIR | Static data – shouldn’t be modified |
| PRD\_DIR | Location of symbolic links pointing to the latest version of each data |
| SRC\_DIR | Source code directory -shouldn’t be modified |
| TIMESTAMP | Name of the timestamp file – shouldn’t be modified |
| JOBARRAY | File name to submit job arrays – do not modify |

Table 5 - Global variables

##### Links (LINK section)

To simplify eventual changes in external resources path, all http(s) paths are defined in the CONFIG\_GLOBAL file as a LINK. A LINK record is composed of 3 values, separated by tabs. For the installation process, you shouldn’t change anything.

**Note:** The number of tabs does not matter as long as there are 3 values – to make it more readable.

|  |  |  |
| --- | --- | --- |
| Column 1 | Column 2 | Column 3 |
| LINK | Link name | Link path |

##### Tools (TOOL section)

All tools are built and shipped in the singularity container. The location of each tool within the container is defined in the CONFIG\_GLOBAL in the TOOL section. A TOOL record is composed of 3 values, separated by tabs and provides the path for one application. For the installation process, you shouldn’t change anything. In addition, parameters to different tools can be specified in this section. In this case, the name should have the suffix \_PARAM

**Note:** The number of tabs does not matter as long as there are 3 values – to make it more readable.

|  |  |  |
| --- | --- | --- |
| Column 1 | Column 2 | Column 3 |
| TOOL | tool name | tool path within the container |
| TOOL | TOOLNAME\_PARAM | List of parameters |

#### CONFIG\_JOB

CONFIG\_JOB is the heart of biorels. It defines the numerous jobs, when they are triggered, if they are triggered and what are the dependencies. Please review the table below describing the different columns in CONFIG\_JOB.

**Note:** The number of tabs between column does not matter as long as there are exactly 13 values – to make it more readable. A value of 0 is equivalent in PHP to nothing, so it will be ignored – reason with *Nothing* is defined by a value of -1.

|  |  |  |
| --- | --- | --- |
| Column | Name | Description |
| 1 | SC | Header tag to know it’s a script |
| 2 | JOB\_ID | Unique numeric identifier for the job  Private job are prefixed with P |
| 3 | JOB\_NAME | Name of the job. Must be the same as the job filename |
| 4 | REQUIRED | List of job identifiers, separated by |, that are required to be updated prior to trigger this job if they are enabled  -1 if no required job |
| 5 | REQUIRED\_TRIGGER | List of job identifiers, separated by |, that would trigger the job if the REQUIRED jobs and the REQUIRED\_UPDATED jobs have been run successfully |
| 6 | REQUIRED\_UPDATED | List of job identifiers, separated by |, that are required to be successfully run at least once  -1 if no parent job |
| 7 | DIRECTORY | Working directory name in PROCESS/. Must be the same name as the data source |
| 8 | REQUIREMENT | Type of requirement, based on parent jobs:  C: All parent jobs must be updated (Complete)  A: Any parent jobs must be updated to trigger the updated  D: All parent jobs that are NOT disabled must be updated |
| 9 | JOB\_TYPE | D: Processing job  P: Moving to PRD job |
| 10 | UPDATE\_FREQUENCY | For jobs with no parent – defines a frequency:   * Time format (24h): HH:MM (00:10 is 10 past midnight) * D[N]: Every N day. D3: every 3 days * W[N]: Every N Week. W2: every 2 weeks   For jobs with parent:   * P (when Parent jobs are successfully completed) |
| 11 | JOB\_RUNTYPE | S: Script  R: Runtime |
| 12 | CONCURRENT | List of jobs Identifiers separated by |. If any of such jobs are running, this job will wait. |
| 13 | MEMORY | -1 if no extra memory necessary  Otherwise value in Mb |
| 14 | DESCRIPTION | Simple textual description of the job purpose. |

Table 7 - Description of CONFIG\_JOB columns

##### Case with taxonomy and gene

To better explain this, we will look at 2 data sources, taxonomy and gene. Below is a snapshot of the CONFIG\_JOB in which we have defined 7 scripts, 1 related to taxonomy and 6 for genes.

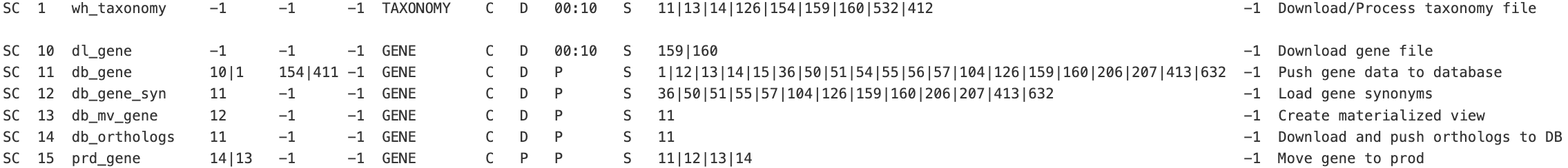


Table 8 - Example of CONFIG\_JOB

In the example above, we are covering two data sources, TAXONOMY and GENE – which we usually define in the directory name for convention. TAXONOMY (column 6) is managed by 1 script, called wh\_taxonomy (column 3). From a scientific perspective, the taxonomy is on the top of the scientific description, since all genomic information is ultimately associated to a given taxon. Therefore, there is no parent dependency (column 4 & 5). We want to maintain the data updated frequently, so we make sure it’s enabled (column 10) set the update to be triggered 10 minutes past midnight (column 9). It is a regular script (column 11) with no extra memory required (column 13). We will come back on the CONCURRENT column after explaining the gene.

We also want to cover gene information, and as such, we make sure that all GENE scripts are enabled (column 10). The first script dl\_gene, will download all gene-related files. Since it’s just downloading files, there is no dependency on other scripts (column 4 & 5) and we can trigger it every day at 10 minutes past midnight (column 9) to keep our database up to date. So far, we have been looking at scripts with no dependencies (parents). However, db\_gene – which process gene information and stores it in the database – will require to have the gene-related files (from dl\_gene) as well as the taxonomy fully loaded in the database (from wh\_taxonomy) – before it can run. Thus, db\_gene will have two parents: wh\_taxonomy and dl\_gene, with job id 1 and 10 respectively (Column 4). The frequency column (column 9) is changed to P (for Parents), to make sure it will be triggered only when both wh\_taxonomy and dl\_gene have been successfully run. In the even where there is a failure in downloading the gene files or processing taxons, db\_gene will not be triggered.

**Concurrent jobs:** We define here as concurrent jobs any job that will modify the content of tables in the database that either the job of interest will modify or will rely upon. In the case of TAXONOMY and GENE data, we have a clear relationship between the two, since a gene is defined in a given taxon. Thus, if we decide to modify the taxon table while modifying the gene table, there is a possibility of conflict where either a taxon is missing, deleted or replaced and therefore a gene couldn’t be mapped to a taxon. For this reason, db\_gene (job id 11) is concurrent for wh\_taxonomy (job id 1, column 12) and wh\_taxonomy is concurrent of db\_gene. This imply that if db\_gene is running, wh\_taxonomy cannot run. Similarly, if wh\_taxonomy is running, db\_gene cannot run. However, the latter situation is redundant with the db\_gene parent requirement. This additional rule exists in which db\_gene parent requirements are met AND it has been more than 24h that wh\_taxonomy hasn’t been run. This situation would happen in the case of a reboot.

Once db\_gene has completed successfully, db\_gene\_syn and db\_orthologs will be triggered since their only parent script is db\_gene. They will run in parallel. Once db\_gene\_syn is completed, db\_mv\_gene will be run. At last, once db\_mv\_gene and db\_orthologs are successfully completed, prd\_gene will be executed.

#### CONFIG\_USER – User configuration file

CONFIG\_USER is the configuration file generated during the Biorels configuration – Data sources, Genomes, Proteomes step. It provides all the information about high level options, genomes, proteomes and jobs to process.

**Note:** You will need to stop and re-run monitor\_jobs to enable those changes. See Running Biorels

##### Global user configuration

EMAIL and EMAIL\_FROM are respectively email addresses for Biorels to communicate to and from, respectively.

##### Processing configuration

To minimize the risk of collision with other applications, Biorels offers the possibility to specify a prefix to use for all the jobs Biorels will submit. Changing **JOB\_PREFIX** will update all future job submission. You can specify the prefix for jobs submitted from the website with **WEBJOB\_PREFIX**.

**Warning:** Please ensure that no jobs or web jobs are currently running, otherwise Biorels will lose track of them while they are still running, will resubmit them which will create collisions.

**Warning:** Do not use the same job prefix and webjob prefix.

A few additional configurations are possible:

* WEBJOB\_LIMIT: Maximal number of web jobs that can run simultaneously. Default: 20
* CHECK\_ITER: Time in seconds in which the system will run a check when no jobs are running. Default: 3600
* CHECK\_RUN: Time in seconds in which the system will run a check when some jobs are running
* KEEP\_PREVIOUS\_DATA: T if you want to keep and archive the previous versions of each data source. F to delete them

##### Private configuration

Once one parameter exists in CONFIG\_USER to enable private processes. It is used as an additional safeguard. Please set PRIVATE\_ENABLED to T if you want to use private processes, F otherwise. It is strongly recommended to review the PRIVATE\_SECTION.

##### Uniprot configuration

UniProt offers access to different files depending on your needs. Swiss-Prot encompass all the UniProt records that have been reviewed, while TrEMBL provides all the records that haven’t been reviewed and have been programmatically annotated. Uniprot also offers proteomes specific to given organism. To ensure we offers the same level of granularity, BioRels allow users to configure Uniprot in a similar fashion. If you wish to cover only specific species, please refer to Species configuration section. Additionally, you can select whether you want to consider SwissProt, TrEMBL or both.

If you wish to enable SwissProt, set WITH\_UNIPROT\_SP to T. F otherwise

If you wish to enable TrEMBL, set WITH\_UNIPROT\_TREMBL to T. F otherwise.

**Process behavior:** If you have requested to process some type of Uniprot data, dl\_swissprot and dl\_trembl will be enabled. However, if WITH\_UNIPROT\_SP or WITH\_UNIPROT\_TREMBL are set to F, dl\_swissprot and dl\_trembl will stop immediately and send a SUCCESS status, allowing the rest to proceed.

**Important note:** TrEMBL is too big for Biorels to proceed. Setting up WITH\_UNIPROT\_TREMBL will download TrEMBL data from UniProt and generate a blastp database. If you have requested some organism’s proteomes, it will also add TrEMBL records from those organisms that are not in the proteomes. In addition, if you need specific TrEMBL entries from ChEMBL or X-Ray, it will extract them from the file rather than downloading them.

**How to choose?**

* If you work on a specific organism, the proteome will be enough and you will not need SwissProt or TrEMBL.
* If you work on multiple organisms – comparing them, the proteomes will be enough
* If you work on a specific target and wants to look at more distant information, SwissProt should be considered
* TrEMBL should only be considered for large entities, companies or university’s platforms to wants to provide broader access and capabilities.

##### 3rd party data sources with licenses

Some data sources will require licenses or API Key. 2 data sources are currently considered in Biorels: DrugBank and OMIM. OMIM will require an API Key, which you can set with **OMIM\_API\_KEY**. DrugBank will require a login/password, that can be set with **DRUGBANK\_LOGIN**. N/A value can be set if you don’t have login/key information.

##### Pubmed configuration

**API Key:**

Biorels performs a lot of queries on Pubmed to search for genes, drugs, clinical trials, diseases, tissues. NCBI Pubmed API is limited by default at 3 queries per second. However, They offer the possibility to increase that number to 10 queries by second with an API Key. To request an API key – which is free, please follow this link: <https://ncbiinsights.ncbi.nlm.nih.gov/2017/11/02/new-api-keys-for-the-e-utilities/>. Once NCBI generates the NCBI API Key, you can set **PUBMED\_API\_ID** value to this API Key. Otherwise, please set it to N/A

**Abstract:**

Pubmed abstracts provides a lot of useful information when looking at a paper. However, it also takes a lot of disk space – 33Gb – without being additionally processed in Biorels. Therefore, Biorels has the option to not store abstracts in the database: **PUBLI\_W\_ABSTRACT**. Set it to Y if you wish to store them, N otherwise.

**Citations:**

In addition to abstracts, Pubmed provides citations recorded in publications. Biorels has the option to not store such citations in the database: **PUBLI\_W\_CITATIONS**. Set it to Y if you wish to store them, N otherwise.

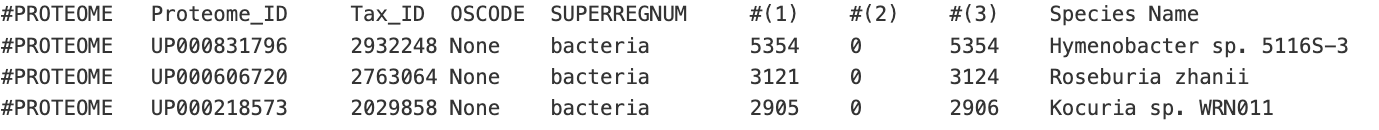
##### Promoter configuration

You can decide how many nucleotides on the 5’ end you want to consider to be the promoter region. **PROMOTER\_RANGE** defines that range and by default it is set to 5000. Not currently in use

##### Proteome configuration

According to UniProt definition, “*A proteome is the set of proteins thought to be expressed by an organism. The majority of the UniProt proteomes are based on the translation of a completely sequenced genome, and will normally include sequences that derive from extra-chromosomal elements such as plasmids or organellar genomes in organisms where these occur. Some proteomes may also include protein sequences based on high quality cDNAs that cannot be mapped to the current genome assembly due to sequencing errors or gaps. These are only included in the proteome following manual review of the supporting evidence, including careful analysis of homologous sequences from closely related organisms.”* (**Source**: <https://www.uniprot.org/help/proteome>).

You can select one or multiple proteomes to be processed in Biorels. Uniprot reference file for proteomes is defined in CONFIG\_GLOBAL > LINK > FTP\_UNIPROTEOME. If you wish to add a proteome, download the UniProt proteome file and select your proteome of interest. Then, copy paste the line between #[PROTEOME] and $[/PROTEOME] sections. Add PROTEOME\t at the beginning of the line.

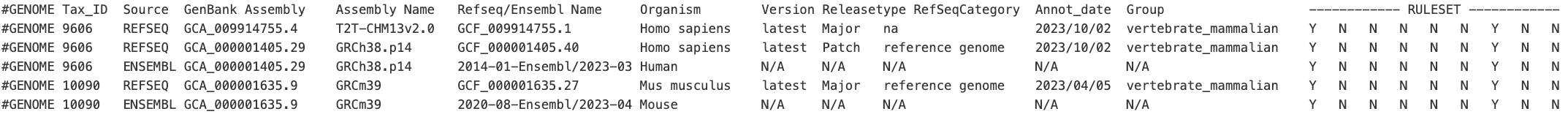


Here is the description of the different columns in the Proteome section:

* Column 1: PROTEOME tag
* Column 2: Uniprot Proteome ID
* Column 3: TaxID
* Column 4: OSCODE
* Column 5: SUPERREGNUM
* Column 6: Number of entries in main fasta (canonical)
* Column 7: Number of entries in additional fasta (isoforms)
* Column 8: Number of entries in gene2acc mapping file
* Column 9: Species name

##### Genome configuration

Biorels allows to process the genomes and transcriptome of different organisms, provided by RefSeq or Ensembl. The Genome section in CONFIG\_USER allows you to define such genomes. It is not recommended to modify that section by yourself. Instead use the Genome selection process of the installer.



##### Gene configuration

For this configuration item, we consider all information related to a gene: gene symbol, identifier, name, synonyms, chromosomal and cytogenetic location – as provided by NCBI Gene.

Considering all genes from all organisms greatly increase computing time, most of that time is being used for transient genes, such as LOC genes. By default, Biorels will optimize the list of processed genes based on the genomes and proteomes requested. However, you have additional configuration rules you can modify.

**Uniprot:** If you enable Uniprot as a data source, you have the possibility to process the genes from all organisms defined in SwissProt or TrEMBL, depending on your choice. This is in addition to the defined proteomes. To enable such rule, set **UNIPROT\_GENE** to Y, N otherwise.

**ChEMBL:** If you enable ChEMBL as a data source, you have the possibility to process the genes from all organisms defined in ChEMBL. To enable such rule, set **CHEMBL\_GENE** to Y, N otherwise.

**Additional taxons:** If you wish to add specific organisms that are not defined by a proteome or a genome, you can add it in the **TAXON\_LIMIT** parameter. Each taxon is defined by its NCBI taxonomy identifier and must be separated by a pipe (|). If you have chosen a proteome or a genome, **TAXON\_LIMIT** will already be populated with the corresponding NCBI taxonomic identifiers for the corresponding organisms. To consider all possible genes of all possible organisms, please set **TAXON\_LIMIT** to N/A – WARNING – This is computationally expensive!!

**Finding the NCBI Taxonomy identifier:** To consider all genes from a given list of organisms, you will need to provide the list of Taxonomic Identifiers – as provided by NCBI Taxonomy – separated by |. To retrieve a Taxonomic Identifier, you can follow this link as an example: <https://www.ncbi.nlm.nih.gov/Taxonomy/Browser/wwwtax.cgi?mode=Info&id=9606> – which is related to Homo Sapiens. Type in the organism name in the search box (Red arrow in Figure 1- NCBI Taxonomy website), click Go. The Taxonomy ID will be the value of interest here (Green arrow in Figure 1- NCBI Taxonomy website).

Example of retrieving Taxonomy Identifier from NCBI Taxonomy,


Figure 1- NCBI Taxonomy website

#### Private section and schema

Biorels offers the possibility to have a private schema, script and process directories. The idea behind it was two-fold. First, we want to have the possibility to compare public data with public+internal data using the same processes and visualization tools. This allows on the website to show either one depending on their security access. We also wanted to have the possibility to create internal workflows that are specific to our business needs. Unless you want such capabilities, which imply some degree of duplication in the data, using the “public” version will be good enough since your data will not be published, and you can fork the repository for your own needs.

How to decide whether you need the private schema and section then?

* if you have a lot of internal data and you want to able to compare private vs private+public.
* If you want different security levels. The public schema for the public data only while the private schema is used for public + internal data with a more restricted access.

To enable the private schema, you should have already set a schema name in setenv.sh. If not, please review the Environment variables – setenv.sh section. In addition, you should have run the runInstall.php to create the tables in the private schema - see Database Installation. Be aware that this will delete the content of the public and the private schema. Once done, you can change PRIVATE\_ENABLED to T in CONFIG\_USER

## Developing in Biorels

Thank you for being interested in contributing and developing Biorels!!! In this section, we will review some of the pre-requisites, then we follow with high level concepts of jobs definition and then deep-dive into writing code.

### Reviewing the fundamentals

If you haven’t already, we strongly recommend that you review the following sections of this documentation:

* Configuration of the environment
* Directory structure
* CONFIG\_GLOBAL
* CONFIG\_JOB
* CONFIG\_USER – User configuration file

### Understanding dependencies

We will define a data source as the data generated and made available on a regular basis by an external organization – private or public. Example of such data source would be UniProt or DrugBank. A data source almost always depends on another data source called a parent data source. However, that dependencies can differs between critical and non-critical.

**Critical dependency:** If you ignore that parent, you will lose some important scientific concept.

*Example:* A UniProt record provides information about a protein in a given organism. The organism here is critical. If ignored, you will lose some important scientific knowledge – which is in which organism this protein is defined in.

**Non-critical dependency:** If you ignore that parent, you will lose some related information that augment the data source.

*Example:* A Uniprot record provides a list of external identifiers related to this protein record. If you ignore those identifiers, you don’t lose any scientific information that is necessary to define a protein record.

Before registering your new data source, you will need to ensure that all critical parents are already in Biorels. If not, then you will need to register them and create the corresponding scripts.

### Adding a program/library to the container

Please open biorels.sing.txt in $TG\_DIR/BACKEND/CONTAINER directory. Biorels.sing.txt contains all the instructions to build the singularity container, defined in different sections. To add the tools or packages that you need, please follow the different sections below.

#### Adding a Linux package

If you wish to add a Linux package or program that is available via yum, please go to the end of **PART 1- Linux packages**. A commented line is available for you to add your own packages. Just remove the # and add your packages.

#### Adding a Python library

Python is compiled during the container’s build to offer one of the latest python versions. Python includes a package manager, called pip to install any additional package. Please refer to **PART 2 – Python Packages** and add your packages.

#### Adding a PHP library

If the package of interest is a native PHP library, please scroll down to **PART 3 – PHP Packages** and add the PHP library. Alternatively, if you need a non-native PHP library, such as ones developed by the PHP community, you can use composer to add such library. Composer is an equivalent of pip for Python. To search for the different packages in PHP, you can go to Packagist: <https://packagist.org>. Once you have the package name, please add it at the end of **PART 3.**

#### Compiling

The container’s build includes gcc and g++ compilers. These compilers are used to compile and build bowtie, bowtie2, blast, EMBOSS, samtools and LillyMol. If you wish to add your own tools that requires compilation, you will first have to add a line in the Download section to download the source code. Please refer to **PART 4 – Compilation** to add the download line using wget. Once done, add the compiling steps. A few guidelines are to be followed:

* Create a directory in $TG\_DIR/BACKEND/APPS/[APP\_NAME]
* Source code should be unzip in a directory under $TG\_DIR/BACKEND/APPS/[APP\_NAME]
* Remove the archive after unzip to save space in the container
* Configure and compile within the APPS/[APP\_NAME directory]
* Clean the objects file after compiling

For all compiled application, it is strongly recommended to create a TOOL line in CONFIG\_GLOBAL providing the path of the tool in the container. This allows to avoid hard-coded path in the scripts. Please refer to Tools (TOOL section) for format.

### Registering a new data source

To process a new **data source,** you will need to create one or multiple **jobs** (or scripts), that will have one or multiple **dependencies** to other jobs, especially within that data source. All the work will be done in **$TG\_DIR/BACKEND/SCRIPT**. We have been trying to be mindful that it might be a lot of files and configuration changes to develop in BioRels. Therefore, to make it easier, we have developed an experimental script that will generate those for you based on some templates. So you can go straight to the section: Experimental script generator, although it would be preferable to go through the sections below first – at least at high level.

#### Data source directory

The first step is to create a new directory called by the name of the data source. Example: UNIPROT, CHEMBL, CHEBI … . For our next steps, we will call it **DATASOURCE**. A few guidelines are proposed:

* All uppercase
* With the same name as the data source. For Uniprot => UNIPROT
* Spaces replaced by \_

#### Job nomenclature

The second step is creating the different jobs in that DATASOURCE directory. We recommend the following architecture below. Please note that we don’t provide here the extension since we can currently develop in both PHP and python. Therefore, please choose the language of your choice and add the corresponding file extension

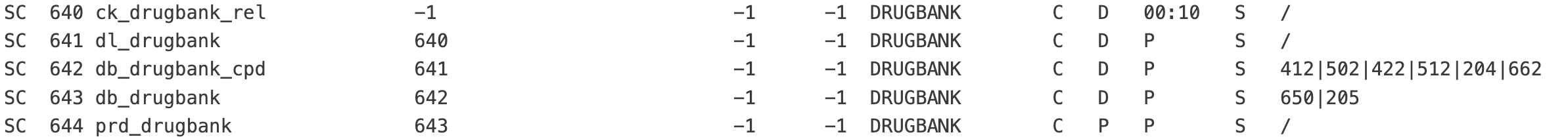
* A ck\_DATASOURCE\_rel script: to be run daily and **c**hec**k** for the **DATASOURCE** to **rel**ease a new version.
* A dl\_DATASOURCE script: to be run after ck\_DATASOURCE\_rel to download the files
* A db\_DATASOURCE script: to be run after dl\_DATASOURCE to process the files
* A prd\_DATASOURCE script: to be run after db\_DATASOURCE to cleanup and move to production

When the data source is very small, such as just one file being processed and pushed into one or a few tables, we’d recommend doing an all in one – that we dubbed **wh**ole:

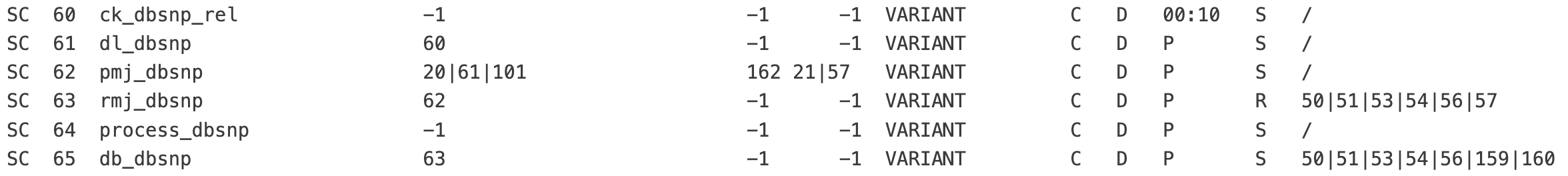
* A wh\_DATASOURCE script: the perform the **wh**ole job of checking for release, download, processed and push to production

A few additional scripts should be added depending on the situation:

* If you are processing small molecules, an additional db\_DATASOURCE\_cpd would be required prior to db\_DATASOURCE.



* If the data source you are processing is very large and require parallel jobs, the following scripts should be added:
  + pmj\_DATASOURCE: to prepare the job batch (pmj stands for **p**repare **m**aster **j**ob)
  + process\_DATASOURCE: to be called in individual job and process the data
  + rmj\_DATASOURCE: to submit the batch and monitor its execution (rmj stands for **r**un **m**aster **j**ob)
  + db\_DATASOURCE would then process the results of those batch jobs and push them to database



Please look at the picture below to understand the different processing paths:

A diagram of a process

Description automatically generated with medium confidence

Figure 2 – Description of the different Workflows for data preparation

Each data source almost always starts with a check (ck\_) script. If the data source is very small, a whole (wh\_) script would perform every steps up to production. If the data source is more complicated, a download script would allow to download the necessary files without having any dependency attached to it that would prevent the download. Then, depending on the type of data and complexity, multiple options are possible. First, if the data source is simple/small enough, a db\_ script would prepare, process and push the data to the database. If this data source possesses molecular structures, it is recommended to have a separate script db\_\*\_cpd to process those compounds independently from and prior to the rest of the data processing. If the data is too big or too complex, it can be broken down into parallel jobs. In this case, an optional preparation script (pp\_) can assess the number or list the records to process or do some cleanup prior to the processing. Then, pmj\_ script would generate the shell scripts necessary for the parallel jobs. The rmj\_ scripts would execute the processing

#### Adding to CONFIG\_GLOBAL

CONFIG\_GLOBAL stores the global variables, the path to different tools as well as their parameters, but also web path to FTP servers. Thus, if you need to add a new web path, you can add it as a link:

LINK FTP\_ENSEMBL <http://ftp.ensembl.org/pub/>

**Guidelines**:

* A link is composed of 3 columns separated by one or multiple tabs
* The number of tabs between columns doesn’t matter – as long as there are 3 values.
* The naming convention is FTP\_**DATASOURCE**
* https paths are recommended

#### Adding to CONFIG\_JOB

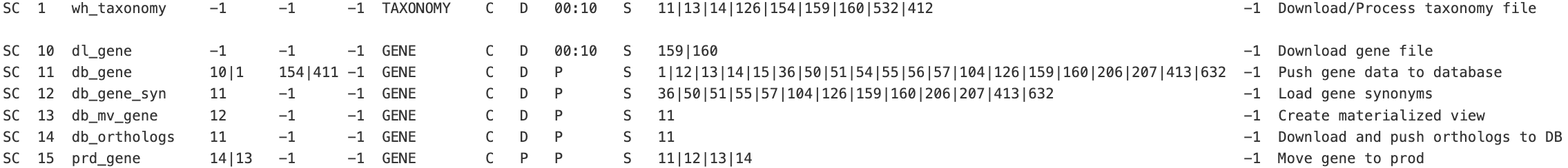
CONFIG\_JOB is the core of the automation. It defines when a job is triggered depending on many different factors. In this section we will review how to properly configure a job.

##### Data source block:

All scripts for a given data source must be with the same line blocks – called a data source block. Each data source block is separated from other data source blocks by an empty line for more clarity. The position of a data source block is important. If you choose to incorporate a new data source in Biorels, you shouldn’t necessarily put the data source block at the end of the CONFIG\_JOB.

Here is a simple guideline:

* Put your data source block after the last critical parent data source block.



In the example above, we have 2 data source blocks: 1 for taxonomy, made of 1 script, and one for gene, composed of 6 scripts. GENE will come after TAXONOMY because any given gene is related to a specific organism.

##### Reviewing job configuration:

Once you have defined where in the file you want to incorporate the data source block, you can create the job lines. Each line will define a job, with its ruleset on when to run or not run it. Below is the walkthrough of each column:

**Column 1:** Each line starting with SC provides the ruleset for a **SC**ript.

**Column 2:** The second column represent the Job identifier (JOB\_ID). If you want to process a new data source, provide the first job a JOB\_ID with a round number, usually 10 above the previous job in the file.

**Column 3:** Job name. Must be the same the script name (minus the extension)

**Column 4:** List of job identifiers that are required to be successful prior to trigger this job – if enabled.

**Column 5:** List of job identifiers that would trigger the jobs if the required jobs and the required\_updated jobs have been run successfully.

**Column 6:** List of job identifiers that are required to be successfully run at least once.

*Important note for Columns 4 to 6:* the behavior can be different between critical and non critical dependencies. If a user chooses not to enable a critical dependency, then its requirement as a dependency to your job will be ignored. Therefore you must enable in your script logic a failsafe if a critical data isn’t there to stop and fail the job.

**Column 7:** Directory name. Must be the same as the DATASOURCE directory name.

**Column 8:** Triggering requirement, based on dependent jobs:

* + C: All parent jobs must be updated (Complete)
  + A: Any parent jobs must be updated to trigger the updated
  + D: All parent jobs that are NOT disabled must be updated

**Column 9**: Job type. D: Processing job / P: Moving to PRD job

**Column 10:** Update frequency. For jobs without parents:

* Time format (24h): HH:MM (00:10 is 10 past midnight)
* D[N]: Every N day. D3: every 3 days
* W[N]: Every N Week. W2: every 2 weeks

For jobs with parent:

* P (when Parent jobs are successfully completed)

**Column 11**: Runtype job: S (Script) R (Runtime, i.e. batch)

**Column 12:** Concurrent jobs.

##### Concurrent jobs

Jobs that alter the database in some ways, whether via an insertion, a delete or an update, are susceptible to conflict with other running jobs. Two situations can arise:

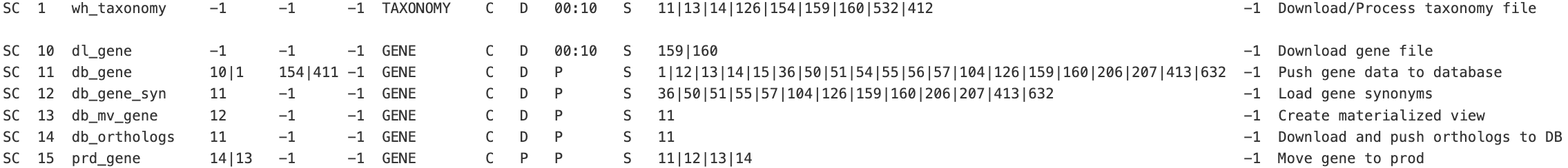
* A job is currently modifying a table X that another job depends upon.
* A job is currently modifying a table X that another job is modifying.

In those two situations, there is a risk that one job is modifying a specific record required for the other, leading to potential failures. To avoid it, Column 12 lists all the Concurrent jobs, i.e. jobs that if they are running, the job of interest will wait. The guideline for concurrent job is as follow:

* Parent jobs, i.e. those defined in columns 4 to 6 does not need to be provided (Situation 1)
* All child jobs, i.e. jobs that have for critical dependency this data source must be considered as concurrent. (situation 1 but reversed)
* All jobs that would modify the same database tables. (Situation 2). Example:

**Important corollary:**

Any job that you will have in CONFIG\_JOB must be defined as a concurrent job of any parent job.



If you review the image above, db\_gene (job id 12), which needs taxon information, is a concurrent job of wh\_taxonomy. Indeed, we do not want to add new genes to a taxon that is being deleted by wh\_taxonomy for example.

##### For ck\_DATASOURCE\_rel job type

* **Column 4, 5 and 6:** must be -1, i.e no dependencies
* **Column 8:** set to A (Any). Since there’s no dependency, that’s equivalent to none
* **Column 9:** set to **D (**Processing job)
* **Column 10:** set to 00:10
* **Column 11:** set to S (for script)
* **Column 12:** -1. No concurrent jobs

The purpose of a ck\_\*\_rel job is to check, on a daily basis for a new release of the data source. It therefore shouldn’t be requiring any dependencies. It is a script that does a processing job that should be triggered every day, here at 10 past midnight. Since it’s just checking, there is no risk for a collision with another job.

##### For dl\_DATASOURCE job type:

* **Column 4:** Must have the ck\_DATASOURCE\_rel as dependency
* **Column 5 and 6:** must be -1, i.e no dependencies
* **Column 8:** set to C (Complete). All dependencies should have been successfully run.
* **Column 9:** set to **D (**Processing job)
* **Column 10:** set to **P** (Parent)
* **Column 11:** set to **S** (for script)
* **Column 12:** -1. No concurrent jobs

The purpose of a dl\_ job is to download the new version of the data source. As such, it should only be triggered if ck\_DATASOURCE\_rel has been successfully run and found a new version (as defined by column 8). Therefore, a dl\_ job usually has one dependency, the ck\_DATASOURCE\_rel job (column 4). Column 10 specifies to not run this job daily but to wait that **P**arent dependencies are successful. Since it’s just downloaded, there is no risk for a collision with another job.

##### For db\_DATASOURCE job type:

A db\_ job aim at processing the newly downloaded files and push the data in the database. This imply that all dependent data must be already in the database prior to processing this data source. For each dependent data source, if it is:

* Updated frequently, then the identifier of the prd\_ script or the db\_script must be provided in **Column 4**
* Updated rarely, or at a lesser frequency than your data source:
  + Identifier of ck\_DATASOURCE\_rel script must be provided in **Column 4**
  + Identifier of prd\_DATASOURCE must be provided in **Column 6**
* **Column 4:** Must have the dl\_DATASOURCE and any other critical or non dependency
* **Column 5:** should be -1, i.e no dependencies
* **Column 6:** For a rarely updated data source, prd\_DATASOURCE of dependent data sources
* **Column 8:** set to C (Complete). All dependencies should have been successfully run.
* **Column 9:** set to **D (**Processing job)
* **Column 10:** set to **P** (Parent)
* **Column 11:** set to **S** (for script)
* **Column 12:** Review dependent jobs

A db\_DATASOURCE must only be triggered when dl\_DATASOURCE and all critical dependencies have been successful (Column 8 to **C**omplete and Column 10 to **P**arent).

**Important: Don’t forget to define this job as concurrent to its critical and non-critical dependencies**

##### For prd\_DATASOURCE job type:

Production scripts are mainly existing to cleanup the processing files, delete the former version and create an alias of the current version to PRD\_DIR. It requires any db\_ jobs to be successful prior to being run.

* **Column 4:** Must have the db\_DATASOURCE. No need for dl\_DATASOURCE since it’s covered by db\_DATASOURCE
* **Column 5:** should be -1, i.e no dependencies
* **Column 6:** should be -1, no dependencies
* **Column 8:** set to C (Complete). All dependencies should have been successfully run.
* **Column 9:** set to **P (**PRD job)
* **Column 10:** set to **P** (Parent)
* **Column 11:** set to **S** (for script)
* **Column 12:** Only if a specific job uses an actual file in the DATASOURCE directory.

##### For wh\_DATASOURCE job type:

Wh\_ jobs are used when the data source is small and everything can be handled in one script, i.e. the check, download, process, push to database and move to prod. For each dependent data source, if it is:

* Updated frequently, then the identifier of the prd\_ script or the db\_script must be provided in **Column 4**
* Updated rarely, or at a lesser frequency than your data source:
  + Identifier of ck\_DATASOURCE\_rel script must be provided in **Column 4**
  + Identifier of prd\_DATASOURCE must be provided in **Column 6**
* **Column 4:** Must have any (non)critical dependency(ies)
* **Column 5:** should be -1, i.e no dependencies
* **Column 6:** For a rarely updated data source, prd\_DATASOURCE of dependent data sources
* **Column 8:** set to C (Complete). All dependencies should have been successfully run.
* **Column 9:** set to **P (**PRD job)
* **Column 10:** set to **P** (Parent)
* **Column 11:** set to **S** (for script)
* **Column 12:** Only if a specific job uses an actual file in the DATASOURCE directory.

**Important: Don’t forget to define this job as concurrent to its critical and non-critical dependencies**

##### For batch jobs

Batch jobs are a special case to be used for long computing exercise where it is necessary to parallelize. A batch process is divided into 4 scripts:

* pmj\_DATASOURCE: to prepare the job batch
* process\_DATASOURCE: to be called in individual job and process the data
* rmj\_DATASOURCE: to submit the batch and monitor its execution
* db\_DATASOURCE would then process the results of those batch jobs and push them to database

###### pmj\_ jobs:

pmj\_ jobs are used to create the shell scripts that will be run in parallel.

If they export data from the database to prepare those jobs, the jobs inserting those data in the database must be listed as dependency.

* **Column 4:** Must have any (non)critical dependency(ies)
* **Column 5:** should be -1, i.e no dependencies
* **Column 6:** For a rarely updated data source, prd\_DATASOURCE of dependent data sources
* **Column 8:** set to C (Complete). All dependencies should have been successfully run.
* **Column 9:** set to **D (**Processing job)
* **Column 10:** set to **P** (Parent)
* **Column 11:** set to **S** (for script)
* **Column 12:** Only if a specific job uses an actual file in the DATASOURCE directory.

###### Rmj\_ jobs:

rmj\_ jobs are used to run all the shell scripts that will be run in parallel.

To avoid hundreds, sometimes thousands of jobs run in parallel, all other rmj\_ jobs should be listed as concurrent jobs. This will avoid having multiple rmj\_ jobs running in parallel of themselves.

* **Column 4:** Must have the pmj\_ job. Other critical dependencies must be handled by pmj\_ job
* **Column 5:** should ALWAYS BE -1
* **Column 6:** For a rarely updated data source, prd\_DATASOURCE of dependent data sources
* **Column 8:** set to C (Complete). All dependencies should have been successfully run.
* **Column 9:** set to **D (**Processing job)
* **Column 10:** set to **P** (Parent)
* **Column 11:** set to **R** (for running)
* **Column 12:** List all other rmj\_ jobs

###### Process\_ jobs:

The process\_ jobs are the scripts called by rmj\_ job to execute code in parallel.

#### Adding to CONFIG\_USER

Once the job is configured in CONFIG\_JOB, we need to add it to the CONFIG\_USER file. As a reminder, the CONFIG\_USER file is aimed to be a user defined configuration file allowing user to specify which script is enabled or not – among other things. In the JOBS section of the CONFIG\_USER file, after #[JOB] line and before #[/JOB] line, you will need to add your script(s). The format for each job is made of 3 columns, separated by tabs. The number of tabs between 2 columns doesn’t matter, as long as 3 non-empty textual values are provided. The format is as follow:

JOB SCRIPT\_NAME STATUS

Where SCRIPT\_NAME is the name of the script/job as defined in column 3 of CONFIG\_GLOBAL. The STATUS must be either T (job enabled) or F (job disabled).

If you are developing scripts for a new data source, please create a section starting with # and followed by the name of the data source:

**# PMC**

JOB ABCD T

JOB ADCE T

**# CLINVAR**

JOB FFAT F

#### Adding Shell script

From there, we need to create a few shell wrappers. The first one is located in $TG\_DIR/BACKEND/SCRIPT/SHELL/ and should be name by the script name and the shell extension. Below is an example with wh\_taxonomy, where the shell script will be named wh\_taxonomy.sh.

|  |  |  |
| --- | --- | --- |
| Job name | Script name | Shell script |
| Wh\_taxonomy | Wh\_taxonomy.php | Wh\_taxonomy.sh |

The shell script should be made of usually two to three lines, such as in the example below with wh\_taxonomy.sh:

1. #!/bin/sh

2. source $TG\_DIR/BACKEND/SCRIPT/SHELL/setenv.sh

3. php $TG\_DIR/BACKEND/SCRIPT/TAXONOMY/wh\_taxonomy.php

Line 1 is the shell interpreter command.

Line 2 MUST be always present and is the main reason for this shell script. It enables to source environment variables.

Line 3 is calling the job’s script.

**Note:** Please note that in this shell script, $TG\_DIR should already be defined.

**Important Note: this shell wrapper allows you to call a script using any language, not just PHP.**

#### Generate container shell scripts

The last script to generate is the container shell script. All container shell scripts (CS) are located in $TG\_DIR/BACKEND/CONTAINER\_SHELL. The goal of each CS wrapper is to allow a script to be executed from a container. In a very similar way, it is made of 3 lines, the shell interpreter command, sourcing the environment variables and the script execution. Please not however that we call biorels\_exe to run the script within the container. This allows job submission more easily.

#!/bin/sh

source $TG\_DIR/BACKEND/SCRIPT/SHELL/setenv.sh

biorels\_exe php $TG\_DIR/BACKEND/SCRIPT/TAXONOMY/wh\_taxonomy.php

#### Experimental script generator

To help in navigating the process of creating those scripts for a new data source, we have created a process that is in $TG\_DIR/BACKEND/DEVELOP. To execute the process, simply call the following command:

cd $TG\_DIR/BACKEND/DEVELOP

Biorels\_php ./new\_script\_startup.php

The script will walk you through a series of question to help assess the requirements for your data source. The first series of question will aim at assessing the type of scripts you will need (Figure 2).

Is your process small enough to be covered by 1 script? (Y/N)

Answering Y will limit number of scripts to 2: a ck\_\*\_rel script and a wh\_ script

Answering N will automatically add dl\_\* and prd\_\* scripts as well as trigger 3 additional questions:

Does your process includes processing compounds (Y/N):

Answering Y will add a db\_\*\_cpd script.

Does your process requires to run parallel jobs? (Y/N):

Answering Y will add pmj\_\*, process\_\*, rmj\_\* scripts

Do you need a preparation/cleanup script prior to the processing script? (Y/N):

Answering Y will add a pp\_\* script

Next, generic questions about the data source requirements and programming language will be asked:

Do you prefer to code in PHP (N for Python)? (Y/N):

Answering Y will generate PHP scripts

Answering N will generate Python scripts

Is this a private data source? (Y/N):

Answering Y will move the code to PRIVATE\_SCRIPT

Answering N will move the code to SCRIPT

Does it require a login/password? (Y/N):

Answering Y will add to CONFIG\_USER an additional line in the GLOBAL section for the user to provide a login/password

What would be a good root FTP/HTTPS Path for the location of the files:

Please provide the root FTP or HTTPS path for the location of the data files, starting with https:// or ftp://

What is the name of the data source (no space):

Please provide the name of the data source. A few guidelines:

* Only alphabetic characters allowed – mixed of Upper/lowercase is acceptable
* Anything else **WILL** result in undefined behavior.

Which data sources are critical dependencies of your data source?

This last question is absolutely critical. You will be asked to list all of the critical dependencies for your data source. In order terms, list all of the data sources that your data source is a child of. Please refer to the pictures in the publication to understand which data source it might be. You do not need to go all the way to a L1, but just the immediate layer of dependency is enough.

All done! This will generate a new directory, named by the name of your data source, in uppercase. In it, you will see a directory of the same name, which will contain the PHP or Python scripts. In addition, you will see a CONFIG\_CHANGES file that will explain you in detail what you will need to change or add. All set! Just try it out now. Please bear in mind that this is experimental, so if it doesn’t work, just reach out to us!

#### Execute your script

Everything is set! You can now execute your script.

## Additional information

### Molecular entity

#### As drug and molecule structures become increasingly complex, establishing a level of abstraction in our database is essential. Scientists have developed remarkable molecules, such as siRNA, antibody-drug conjugates, and AAVs encapsulated with payloads. However, representing and unifying these diverse structures in a clear and comprehensive manner poses significant challenges.

#### In Biorels, a molecular entity serves as an abstract concept that represents what is tested in vitro or in vivo. Each molecular entity comprises one or more components, each with a defined molarity ratio. Components provide a second layer of abstraction, allowing us to define different sets of molecules, again with specified molarity ratios. A component can consist of one or more molecules, which may be connected covalently or through other means. We will illustrate this with some examples for clarity.

#### A small molecule is the simplest molecular entity, functioning as both a molecular component with a molarity ratio of 1 and a molecular entity with a single component also having a molarity ratio of 1.

#### For example, a siRNA conjugated with a small molecule is defined by a single molecular component made up of two molecules: the siRNA and the small molecule. Assuming a 1:1 ratio, their molarity ratios would each be 0.5, resulting in a molecular entity defined by this single component with a molarity ratio of 1.

#### In contrast, an antibody-drug conjugate (ADC) encapsulated in a lipid nanoparticle (LNP) bound to a delivery peptide represents a more complex case. Here, the ADC acts as a molecular component composed of two or three molecules: the antibody, the drug, and possibly a linker. The LNP serves as another molecular component, incorporating the individual molecules of the LNP formulation as well as the peptide. Consequently, the molecular entity is defined by two components: the ADC component and the LNP component, each with its respective molarity ratio.

#### To streamline this process and minimize redundancy, we have developed a protocol for uniquely identifying the various modalities, components, and entities.

#### Small molecule standardization

First, we will need to define some vocabulary:

* The initial molecule with its counterions will be called the ***initial entry***.
* If multiple molecules are present within a SMILES string, the molecule with the longest string will be considered as the ***main molecule***. The other molecules will be considered as ***counterions***
* The SMILES of the initial entry (main molecule + counterions) will be called the ***Full SMILES***
* The SMILES of the counterions will be called ***Counterion SMILES***
* The SMILES of the main molecule will be called ***Molecular SMILES***
* If standardized, a (s) will be added.

To insert small molecules in the database, 2 input files will be required. A SMILES file for the main molecules and a SMILES file for the counterions. The SMILES file must follow the format below:

FULL\_SMILES[space]ID|InChI|InChI-Key|Counterions|Main\_Molecule

* FULL\_SMILES must be the SMILES string of the complete molecule, including counterions.
* ID is the identifier of the data source
* InChI: InChi string generated from the SMILES string. Although you can provide it, it will be regenerated for consistency purposes. Otherwise, set to NULL
* InChiKey: InChi string generated from the SMILES string. Although you can provide it, it will be regenerated for consistency purposes. Otherwise, set to NULL
* Counterions. List of counterions, separated by dot (.). Otherwise set to NULL
* Main\_Molecule: SMILES string of the longest molecular string.

The counterion file will be defined as follow:

Counterion\_smiles[space]counterion\_smiles

The next steps will involve a series of standardization. First, we standardize the FULL\_SMILES using LillyMol. (s) imply standardization.

FULL\_SMILES(s)[space]ID|InChI|InChI-Key|Counterions|Main\_Molecule

LillyMol will then generate two files: a file where all molecules have been successfully standardized, and a file for molecules with a standardization issue. The next step is to switch FULL\_SMILES and Molecule and add a column (T/F) for standardization success (FULL\_MOL\_STD).

Main\_Molecule [space]ID|InChI|InChI-Key|Counterions|FULL\_SMILES(s)|FULL\_MOL\_STD(T/F)

Next, RDKit is going to generate the InChi and Inchi-Key for the standardized FULL\_SMILES.

Main\_Molecule [space]ID|InChI(s)|InChI-Key(s)|Counterions|FULL\_SMILES(s)|FULL\_MOL\_STD(T/F)

At last, the molecule will be standardized using LillyMol.

Main\_Molecule(s) [space]ID|InChI(s)|InChI-Key(s)|Counterions|FULL\_SMILES(s)|FULL\_MOL\_STD(T/F)

At this step, all but the counterions have been standardized.

##### Counterion definition

The counterion file will then be running under a different standardization process in LillyMol and the counterions will be registered in the database. A mapping between the non-standardized counterion and the standardized record in the database will be maintained. A counterion record is uniquely defined by its standardized SMILES string. In the case where a counterion record contains multiple counterions, those counterions will be ordered alphabetically after standardization. The alphabetically ordered standardized smiles will then be saved in SM\_COUNTERION table.

##### Main Molecule definition

After the standardization process, the main molecule standardized SMILES will be stored in SM\_MOLECULE. The standardized SMILES is the unique definition for the main molecule.

##### Molecule entry

A Molecule entry will now store the initial entry, i.e the combination of a main molecule and its eventual counterion(s). A Molecule entry is uniquely defined by the following:

* The standardized SMILES for the main molecule
* The standardized SMILES of the counterion(s) or NULL if none
* The InChi generated on the standardized full SMILES
* The InChi-Key generated on the standardized full SMILES

A md5 hash is generated as a combination of all 4 values, separated by underscore:

Hash= md5(INCHI.'\_'.INCHI\_KEY.'\_'.Main\_molecule(s).'\_'.COUNTERION(s));

**Important:** The InChi and InChi-Key must be generated on the SMILES of initial entry but after standardization.

The information about the molecule entry can then be stored in SM\_ENTRY as follow:

* Sm\_molecule\_id: Foreign key to SM\_molecule defining the main molecule
* SM\_counterion\_id: Foreign key to SM\_counterion defining the counterions – NULL if none
* InChi: Generated InChi string
* InChi-Key: Generated Inchi-Key string
* Md5\_hash: Generated Hash
* Full\_smiles: Standardized full smiles

**Note:** The full smiles is not necessary to uniquely defined a molecule entry. However, it is useful for visualization or data extraction purposes.

##### Small molecule as molecular component

Once the molecule is registered in SM\_ENTRY, we can proceed in registering it as a molecular entity. First, we will need to register it as a molecular component. A molecular component is an abstract representation of a molecule, or set of molecules, covalently linked or not. In the case of a small molecule, the corresponding molecular component will be made of just one molecule, the small molecule itself. When there are multiple molecules involved, we must provide the molar fraction of each individual molecule in the mixture. The sum of all molar fractions must be 1.

A molecular component is defined by several descriptors. A molecular\_component\_structure, is the full SMILES or HELM representation of the different molecules in that component. The components column will be the list of alphabetically ordered hashes, separated by |.

Then we define two different hashes, one for the structure, and one being a composite of structure and molar fraction. For the molecular\_component\_structure\_hash, the guidelines are a follow:

* For a single sub-component, it will be the md5 hash of sub-component (small molecule, nucleic-acid, peptide)
* For multiple sub-components, it will be the md5 hash of the components column, i.e. the list of lphabetically ordered hashes, separated by |.

Md5(firstHash|secondHash|...)

The molecular\_component\_hash differs from molecular\_component\_structure\_hash because it adds the molar\_fraction to the hashes. Thus the molecular\_component\_hash is defined as:

Md5(firstHash:firstMolarFraction|[secondHash:secondMolarFraction…])

Once all descriptors have been defined for a molecular component, you can proceed with inserting the record in molecular\_component table. Then, each individual molecule, whether it is a small molecule, peptide, antibody …, must be linked to that component via the corresponding mapping tables. This is where the molar fraction will be stored.

**Note:** The reason for the need of 2 hashes is for simplicity purposes. Indeed, having the same hash between the molecular\_component record and a sm\_entry record for instance allows to join the two tables by the hashes to obtain information on the single small molecule rather than having to perform multiple processes to get it.

**Note:** To get all identical mixtures of molecules but with different molarities, query the molecular\_component\_structure\_hash.

##### Small molecule as molecular entity

A molecular entity is a set of molecular components. Thus, the logic will be very similar – if not easier - to registering a molecular component. A Molecular entity will be defined by 2 hashes: a structure hash and a composite of structure and molar fraction – as well as the list of molecular component hashes and the full structure – in SMILES or HELM. The molecular components column will be the list of alphabetically ordered molecular\_component\_hash, separated by |.

For the molecular\_structure\_hash, the guidelines are a follow:

* For a single component, it will be the molecular\_component\_structure\_hash of the corresponding component.
* For multiple components, it will be the md5 hash of the list of alphabetically ordered molecular\_structure\_hash of the components, separated by |.

Md5(firstHash|secondHash|...)

The molecular\_entity\_hash differs from molecular\_structure\_hash because it adds the molar fraction to the hashes. Thus the molecular\_entity\_hash is the md5 hash of the list of alphabetically ordered pairs of molecular\_component\_hash:molar fraction:

Md5(firstComponentHash:firstMolarFraction|[secondComponentHash:secondMolarFraction…])

Once all descriptors have been defined for a molecular entity, you can proceed with inserting the record in molecular\_entity table. Then, each individual component must be linked to that entity via the corresponding mapping table. This is where the molar fraction of each component will be stored.

**Note:** For a single small molecule, the molecular\_structure\_hash of the entity will be the same as the md5 hash of the sm\_molecule entry.

#### How to register your molecules

Your molecules are the most important molecules, so they deserve their own tables. Before you start, please review the Private section and schema section in the documentation. This will help you to set up the configuration appropriately depending on your needs. The tables will be defined in the public schema if you enabled the public schema only but they will be defined in the private schema **only** if you are using both public and private schema. This is an additional security to avoid potential leak in the public schema if you want your data to be kept private in the private schema.

The process to register the structure of small molecules has already been outline in the previous sections of Molecular entity. In this section, we will review how to identify your molecules and create libraries.

**Internal\_molecule table:** This table will contain the identifiers and the inventory of your internal molecules.

**Internal\_library table:** This table will provide the list of internal libraries.

**internal\_library\_molecular\_map:** This table will associate a library to its list of compounds.

A script is provided in PRIVATE\_SCRIPT/INTERNAL/load\_internal\_mol.php to load internal molecules. Please modify the file path in the copy section to make it work. The required format is a SMILES file as such:

SMILES IDENTIFIER[|INVENTORY]

Where INVENTORY is optional. If you choose to provide it, please ensure it is provided in mg (milligrams) and separated from the identifier with a | (pipe). This script will standardize your small molecules and load them in sm\_molecule (molecular structure), sm\_counterion (counterion), sm\_entry (pair molecule/counterion), and mapped to molecular\_entity. It will then proceed in loading internal\_molecule table with the inventory.

REVIEW SCRIPT

## Database structure

### Database table naming convention

#### Database table naming convention.

##### Table prefix

A table must always start with the scientific concept it represents, followed by a more precise definition. Additionally, it can have a suffix or prefix depending on the situation.

This is the list of scientific concept names related to genomic/proteomic:

|  |  |  |  |
| --- | --- | --- | --- |
| Prefix | Scientific concept | **Prefix** | Scientific concept |
| prot\_ | Protein | **Variant\_** | Variant |
| Gene/gn\_ | Gene | **So\_** | Sequence ontology |
| Chr\_ | Chromosome | **RNA\_** | RNA expression |
| Transcript\_/Tr\_ | Transcript | **ECO\_** | Evidence ontology |
| Taxon\_ | Taxonomy | **GO\_** | Gene ontology |
| Pw\_ | Pathway | **Prot\_** | Protein |
| Tr\_protseq | Translation mRNA/protein | **PTM\_** | Post Translational modification |

This is the list of scientific concept names related to drug/clinical trial ecosystem

|  |  |  |  |
| --- | --- | --- | --- |
| Prefix | Scientific concept | Prefix | Scientific concept |
| Clinical\_trial\_ | Clinical trials | Drug\_ | Drug |
| Clinical\_variant\_ | Clinical variant | Meddra\_ | MEDDRA |
| Side\_Effect\_ | Drug Side effects |  |  |

This is the list of scientific concept names related to publisher ecosystem

|  |  |  |  |
| --- | --- | --- | --- |
| Prefix | Scientific concept | **Prefix** | Scientific concept |
| Company\_ | Company/University | **Patent\_** | Patent |
| Pmid\_ | Publication | **Source\_** | Source of data |

This is the list of scientific concept names related to the assay ecosystem

|  |  |  |  |
| --- | --- | --- | --- |
| Prefix | Scientific concept | Prefix | Scientific concept |
| Activity\_ | Experimental data | **Bioassay\_onto\_** | Bioassay ontology |
| Assay\_ | Assay | **Target\_** | Target |

This is the list of scientific concept names related to the disease/anatomy ecosystem

|  |  |  |  |
| --- | --- | --- | --- |
| Prefix | Scientific concept | Prefix | Scientific concept |
| Cell\_ | Cell lines | **Disease**\_ | Disease |

This is the list of scientific concept names related to 3-D Structure ecosystem

|  |  |  |  |
| --- | --- | --- | --- |
| Prefix | Scientific concept | Prefix | Scientific concept |
| Xr\_ | Macromolecular structures |  |  |

This is the list of scientific concept names related to molecule ecosystem

|  |  |  |  |
| --- | --- | --- | --- |
| Prefix | Scientific concept | Prefix | Scientific concept |
| Molecular\_ | Group of molecules, molecular entities | **Lipid**\_ | Lipid |
|  |  | **SM**\_ | Small molecule |

This is the list of scientific concept names related to protein domain ecosystem

|  |  |  |  |
| --- | --- | --- | --- |
| Prefix | Scientific concept | Prefix | Scientific concept |
| IP\_ | Interpro |  |  |

For example, A more precise definition could be:

* Prot\_seq: Protein sequence
* Prot\_dom: Protein domain
* Prot\_AC: Protein accession
* Prot\_Name: Protein name

Note: **Mv\_** are only used for Materialized views or tables computed on the fly from a query

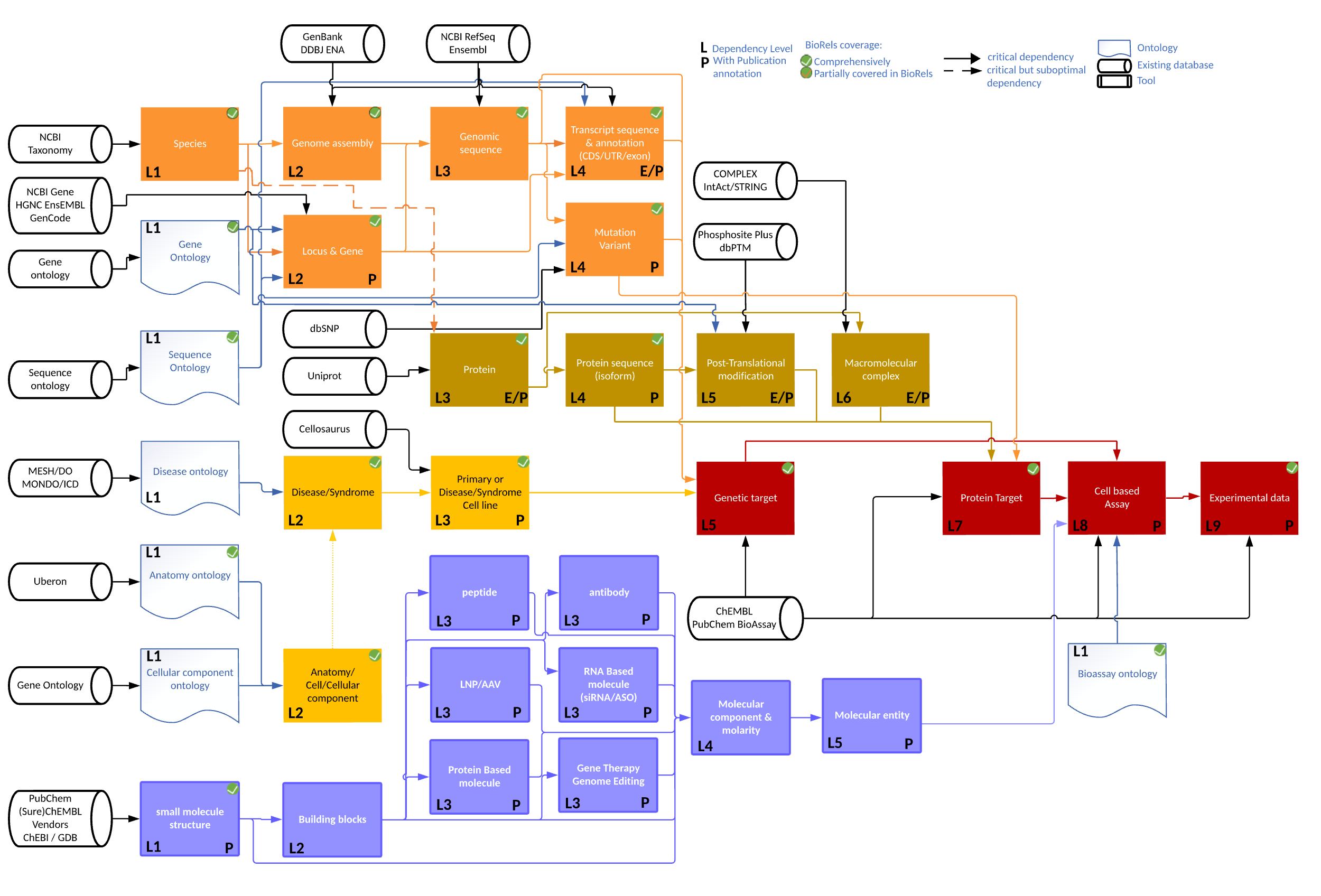
##### Table Suffix:

* \_entry: Tables representing a scientific concept. Example: gn for gene, prot for protein …
* \_syn: Synonyms of an \_entry table, gn\_syn for gene synonyms
* \_rel: Relationship between \_entry records: gn\_rel for orthologs
* \_extdb: External database identifier for an \_entry. Prot\_entry: external identifier for a given protein
* \_tree/\_hierarchy: Hierarchy between different \_entry. Taxon\_tree: hierarchical classification of taxons
* \_map: Connection between \_entry of 2 different scientific concepts or a scientific concept and its names
* \_stat: statistical or agglomeration table
* \_pos: Positions in a sequence. Transcript\_pos: each nucleotide of a given transcript
* \_history: Previous values of a given record

### Biorj File format

#### Biorj theory

Biorj is a data exchange file format for BioRels. Contrary to many other file formats, it does not follow a specific “format”, where fields are rigidly defined. Instead, it is based on a json representation of the database, more precisely of the scientific concept and its dependencies you wish to export. So, let’s dive into the theory a bit by defining some context.



Let’s take the example of exporting experimental data from an assay. As shown in the image above, experimental data operates at an L9 dependency level, meaning there are eight layers of critical dependencies that must be resolved before we can successfully export it. If you need a refresher on dependencies, please refer to the Understanding dependencies section.

The preceding level is the Cell-Based Assay, which contains metadata about the assay type. This level has critical dependencies on several other concepts, including the BioAssay Ontology (L1), Molecular Entity (L5), Protein Target (L7), and Genetic Target (L5). Each of these concepts also has its own critical dependencies, encompassing protein, disease, molecular structure, and gene concepts. Therefore, when exporting any experimental data, all related data from these critical dependencies must be extracted.

In terms of database schema and relationships, a critical dependency can be represented in two ways: either through a foreign key linking to the dependent table or via a mapping table that contains foreign keys to both related tables. In the Biorj format, which we will discuss later, we will focus on these critical dependency tables, referred to as Parents (P).

In addition, some concept can have additional metadata information. If we take the example of an assay, it can have publications associated to it. We usually refer to those as ***non-critical or related dependencies*** since they augment the data but are not critical. Thus, such information can also be extracted and save. In the Biorj format, those will be called children.

If we wish to export experimental data, we do not want to define all of the child and parents of an experimental data concept, which would include all 9 levels of dependencies and their non-critical dependencies. To alleviate some of this challenge, we can group tables into Blocks. A Publication block would provide the rules to extract the publication information, but also its authors, institutions, journal or abstract. Then, if one wants to extract a assay that has reported publications, we can just defines in the assay block that we want to also extract a publication block so that all publication information is also extracted.

#### Biorj rules

Now that we have explained the concepts, let’s look at the rule set defined in $TG\_DIR/BACKEND/SCRIPT/BIORJ/BIORJ\_RULES. We will first focus on an example with publications:

BLOCK pmid\_entry

PARENT pmid\_journal

CHILD pmid\_author\_map<pmid\_author<pmid\_instit

CHILD pmid\_abstract

END

A BLOCK is defined by a table name that would represent the main table for a scientific concept, in this case pmid\_entry for a publication. Publication has a critical dependency, which is the journal it is published in, this pmid\_journal is a PARENT of pmid\_entry. The abstract of a publication is a non-critical dependency, and therefore can be defined as a CHILD of pmid\_entry. Since there can be multiple authors in a publication, Biorels defines a mapping table – pmid\_author\_map – to bridge publications to their authors. In addition, an author is assigned to an institution (pmid\_instit).

Please note here the <. This character is very important as it defines the flow of the data. When you want to export a publication, BioRels will search in pmid\_entry table. Once it finds it, it will then search for that publication ID in pmid\_author\_map. The results from pmid\_author\_map will allow to retrieve pmid\_author records, which themselves will allow to retrieve pmid\_instit. Those retrieval rules are based on the foreign key constraints defined in BioRels.

BLOCK assay\_entry

PARENT assay\_cell<cell\_entry:E

PARENT assay\_cell<taxon

PARENT assay\_tissue<anatomy\_entry:E

PARENT taxon

PARENT source

PARENT assay\_confidence

PARENT assay\_type

PARENT assay\_target<taxon

PARENT assay\_target<assay\_target\_type

PARENT assay\_target>assay\_target\_genetic\_map<assay\_genetic<taxon

PARENT assay\_target>assay\_target\_genetic\_map<assay\_genetic<gene\_seq:E

PARENT assay\_target>assay\_target\_genetic\_map<assay\_genetic<transcript:E

PARENT assay\_target>assay\_target\_protein\_map<assay\_protein<prot\_seq<prot\_entry:E

PARENT assay\_target>assay\_target\_protein\_map<assay\_protein<gn\_entry:E

CHILD assay\_pmid<pmid\_entry:E

CHILD activity\_entry<source

CHILD activity\_entry<bioassay\_onto\_entry

CHILD activity\_entry<molecular\_entity:E

PARENT assay\_variant<prot\_seq<prot\_entry:E

Now let’s take the example of an assay. Similarly, we define a block with the main table for an assay: assay\_entry. An assay has multiple critical dependencies, such as the taxon, source, assay\_confidence, assay\_type. However, it also have more complicated critical dependencies, ones that requires more definitions. Assay\_cell is a table defining the type of cell lines used in assays, as provided by ChEMBL. Itself is critically dependent to cell\_entry table which is defining all cell lines. Please note that cell\_entry is followed by “**:E**”. This E stands for Entry and will trigger all the data export rules defined in the BLOCK cell\_entry. This allows to extract all the critical and related dependencies of cell\_entry without having to define all the rules again.

PARENT assay\_target>assay\_target\_genetic\_map<assay\_genetic<taxon

PARENT assay\_target>assay\_target\_genetic\_map<assay\_genetic<gene\_seq:E

PARENT assay\_target>assay\_target\_genetic\_map<assay\_genetic<transcript:E

This block is particularly interesting. Indeed, we can see here that an assay record has a critical dependency to assay\_target. However, assay\_target\_genetic\_map is a mapping table between assay\_target and assay\_genetic and as such, the foreign keys to those tables are located in assay\_target\_genetic\_map. Thus, the direction of the foreign key relationship is changed from < to >. Gene\_seq and transcript are both followed with **:E**, allowing to get their critically and non-critically dependency data by calling their respective blocks.

CHILD assay\_pmid<pmid\_entry:E

At last, you can see an example of a non-critical dependency with assay\_pmid listing all the publications associated with an assay. This will call the pmid\_entry block, thanks to **:E,** which will get all the publication’s metadata.

#### Biorj requirements and unique keys

To properly function, this process requires a few requirements to be met:

* The schema must be identical. This implies the table names and the column names
* Foreign keys must be identical. Their names however doesn’t matter
* The schema name doesn’t matter
* The unique keys for a given table must be identical

The list of foreign keys is generated on the fly during Biorj import and export. However, the list of unique keys must be defined in $TG\_DIR/BACKEND/SCRIPT/BIORJ/BIORJ\_RULES.

KEYS

activity\_entry molecular\_entity\_id|assay\_entry\_id|value|unit\_type

anatomy\_entry anatomy\_tag

anatomy\_extdb source\_id|anatomy\_extdb|anatomy\_entry\_id

anatomy\_syn anatomy\_entry\_id|syn\_type|syn\_value|source\_id

assay\_confidence description

Each line is made of two columns. The first column defines the table while the second column list all column names of the table, separated by |, which altogether makes the unique definition of a record for that table.

#### Adding rules to Biorj

Depending on your use case, you might need to add a few rules in Biorj\_rules file. To help, here are a few questions to answer to direct you:

* If you are adding a new data source that will create a new scientific concept => Add block
* If you are you modifying any column that can be used to uniquely identify a record? => Update key
* If you are you changing the critical dependencies of the main table? (foreign key) => Update rules
* If you are adding a new table that isn’t a new scientific concept => Update rules

##### Adding a Biorj block

If you are expanding on BioRels database schema, thank you for your contribution! You are now at the stage where you need to test the export/import of BioRels for your tables. If you have been adding a new scientific concept, you will need to create a new BLOCK.

BLOCK [TABLE\_NAME]

The format of a BLOCK starts with the BLOCK word followed by the main table of the scientific concept, i.e. that ones that uniquely define that concept. For instance, for a gene, it would gn\_entry; for a taxon, it would be taxon. Next, you will need to define the PARENT lines, which characterize the critical dependencies for this concept. To do so, you should look at the foreign keys defined in your main table to list all the referenced tables. If those tables are not the main table of scientific concepts, you must follow the path of foreign keys until you reach the main tables, or there are no more foreign keys. Main tables can be found by looking at the table names of the different BLOCKS defined in BIORJ\_RULES. If you reach a main table, you must add the suffix :E to include the metadata associated to it.

Next you need to define the CHILD lines, while provides the list of non-critical dependencies for this new scientific concept. Similarly, you must follow the path of foreign keys until you reach the main tables, or there are no more foreign keys.

The last line of the BLOCK should be END.

In addition, you will need to update the unique keys as described in the next section

##### Update Biorj keys

To perform properly, Biorj will need to know which columns in a given table uniquely defines a record. As many unique keys and foreign keys can be defined for a given table, we need to manually list which one is the proper one to use. The KEYS section in BIORJ\_RULES file follows a key->value pair where the key is the table name in column 1 and the value is the list of columns, separated by |, that uniquely defines a record.

If you have modified a table, please verify that the list of columns is correct.

If you have created a new table, please add the table with the list of columns at the end of the KEYS block, before the END line.

KEYS

activity\_entry molecular\_entity\_id|assay\_entry\_id|value|unit\_type

anatomy\_entry anatomy\_tag

anatomy\_extdb source\_id|anatomy\_extdb|anatomy\_entry\_id

anatomy\_syn anatomy\_entry\_id|syn\_type|syn\_value|source\_id

assay\_confidence description

[TABLE\_NAME] [LIST\_OF\_COLUMNS\_UNIQUELY\_DEFINING\_A\_RECORD\_SEP\_BY\_|]

##### Update rules

If you are modifying the foreign key relationships of a table, you will need to update the corresponding Biorj rules. To do so, please locate the corresponding scientific concept associated with the table and ensure the relationships are properly defined in Biorj rules.

## 

# Website

## Introduction

The website serves as one of the access points to the BioRels database and aim at surfacing the data in many ways. The website is divided into independent modules that can generate webpage, image, flat files or json content. Let’s see how the website directory is structured:

|  |  |
| --- | --- |
| Directory/File name | Description |
| module | All public modules |
| Module/ |  |
| private |  |
|  |  |
|  |  |
|  |  |
|  |  |

## 

The website is built on a set of modules, that can be categorize into two concepts. The first concept is a PORTAL, i.e. a scientific concept from which a user would like that visualize the data from this perspective (a gene, a protein, a compound …). A PORTAL should contain a menu, and a portal page. A PAGE is a specific module with the solely goal of fetching a specific set of data and representing it visually.

Here we present a few of the constraints associated to a module:

* A Module is uniquely defined by a MODULE\_NAME.
* All files necessary for a module are stored in a single directory, defined by a MODULE\_PATH (see below)
* All modules are accessible through the website via a unique WEBLINK\_MODULE
* A Module can take only one input value.
* A Module can take zero, one or many parameters
* A Module can call other module(s) to get either their data or HTML output.

The website is built on modules. Each module is defined in the GLOBAL\_CONFIG file and its files are stored in either the module directory or private/module.

Definition of a module in GLOBAL\_CONFIG:

**START** PAGE|PORTAL MODULE\_NAME

{PORTAL\_ID PORTAL\_NUMBER}

**LOC** MODULE\_PATH

**TAG** WEBLINK MODULE NAME *[PARAMETER]*

**FNAME** MODULE\_FILENAME\_PREFIX

**HTML\_TAG** HTML\_TAG\_NAME

WITH\_EXPORT true|false

**WITH\_DATA** true|false

**END** PAGE|PORTAL *MODULE\_NAME*

We differentiate between a PORTAL, and a PAGE which describes and presents a very specific subset of data from a given PORTAL perspective. Each Module defined by its MODULE\_NAME in the GLOBAL\_CONFIG must be unique as it will be called throughout the website.

START PAGE PROTEIN\_RESIDUE

LOC /protein/protein\_residue

TAG PROT\_RESIDUE REGEX:UNIRES

FNAME protein\_residue

HTML\_TAG protein\_residue

WITH\_EXPORT true

WITH\_DATA true

END PAGE PROTEIN\_RESIDUE

Here we define a new module called PROTEIN\_RESIDUE. Each word is separated by one or multiple tabs (doesn’t matter how many as log as no characters are in between).

LOC represents the location of the module directory in the root website module directory. Thus in this case, the location would be $TG\_DIR/WEBSITE/module/protein/protein\_residue.

The TAG defines how this module will be called and whether or not a parameter is need. In this instance, PROT\_RESIDUE

## Authentication and security

BioRels website allows to differentiate between public and private modules. If you wish to grant a specific access to a given user, you can proceed by sending the access through an http header. To do so, edit the WEBSITE\_CONFIG file and modify WITH\_HTTPD\_ACCESS to T (for True) instead of F (for False) and replace HTTPD\_ACCESS value to the http header you with to retrieve. Then, in require/php/core/r\_auth.php, in the SPECIFIC AUTHENTIFICATION BLOCK, the value associated with the given httpd header will be stored in $STRHTTPD. You can modify how to process the content of this variable to edit the access to either public or private module. If the same module name exist in both public and private and the user has access to both, then the private module will be used.

## Installation and configuration

### Requirements

## Additional configurations & use cases

### Creating a new portal

#### Modify WEBSITE\_CONFIG

Add in website\_config file a new portal block. A portal block always starts with START PORTAL followed by the portal name. PORTAL\_ID is optional. Then a set of tags defined how to trigger this portal. In this example, the GENE portal will be instantiated when a GENE/GENE\_SYMBOL, GENEID/GENE\_ID or ENSEMBL/ENSEMBL\_ID is provided. Each portal requires 2 specific module directories. The first module should be named \*\_portal and is the “welcome” page for this portal. The second module directory should be named \*\_portal\_menu and is the menu for this portal.

START   PORTAL  GENE

PORTAL\_ID   15

TAG     GENE    ([a-zA-Z0-9]{1,10})

TAG     GENEID  ([0-9]{1,10})

TAG     ENSEMBL REGEX:ENSEMBL

LOC             /genomic/genomic\_portal

FNAME           genomic\_portal

HTML\_TAG        genomic\_portal

WITH\_EXPORT     true

WITH\_DATA       true

END PORTAL  GENE

#### Create portal directories

In module directory, create a directory named as the portal, in which all modules related to this portal will be created.

From there, create two directories: \*\_portal and \*\_portal\_menu.

The easiest way to do so is to copy paste the genomic\_portal and genomic\_portal\_menu directories, to renames them and the subsequence files.

#### Define a color for the portal

Each portal has a different color to be easily recognizable.

In require/css/style.css, search for .portal\_col1. Create a new css class with the portal number matching the one in WEBSITE\_CONFIG

.portal\_col10 {

    background-color: #549670;

    color: white;

}

#### Define path to portal

Once done, we need to write the logic enabling the website to go to this portal.

Open php/core/r\_def\_page.php and add a CASE. The case value should match the portal name. A second switch will then list all the tags defined in the WEBSITE\_CONFIG block. A database function must be call to retrieve the minimal amount of information related to that portal and store it in $USER\_INPUT[‘PORTAL’][‘DATA’]

case 'GENE': // Portal name

        {

            switch($USER\_INPUT['PORTAL']['TYPE'])

            {

                case 'GENEID': // Tag defined in website config

                    $USER\_INPUT['PORTAL']['DATA']=gene\_portal\_geneID($USER\_INPUT['PORTAL']['VALUE'],$GLB\_CONFIG['GLOBAL']['TG\_DIR']);

                    if ($USER\_INPUT['PORTAL']['DATA']==array())$USER\_INPUT['PAGE']['NAME']='UNKNOWN\_REC';

                    break;

                case 'GENE': /// Tag defined in website config

                    $USER\_INPUT['PORTAL']['DATA']=gene\_portal\_gene($USER\_INPUT['PORTAL']['VALUE']);

                break;

            }

            break;

        }

# Troubleshooting

## jobs

#### Shell script does not exist.

If you receive a message looking like this:

JOB ID 10200 shell script does not exist /$TG\_DIR/BACKEND/SCRIPT/SHELL/wh\_datasource.sh

Then it means the shell wrapper script is missing and needs to be created. Please refer to section Adding Shell script.

#### Unable to find [JOB\_NAME] in GLB\_TREE

This issue can arise from multiple situations but exist due to the presence of JOB\_NAME in biorels\_timestamp table but not in CONFIG\_JOB or CONFIG\_USER.

Therefore, the first question to ask is: Is JOB\_NAME still relevant? No? Then delete it from biorels\_timestamp by doing so:

DELETE FROM [SCHEMA\_NAME].biorels\_timestamp where job\_name=’[JOB\_NAME]’

Still a relevant job? Then your issue is with the CONFIG files, either CONFIG\_JOB or CONFIG\_USER. Please make sure:

* The [JOB\_NAME] is the same in CONFIG\_JOB and CONFIG\_USER, in biorels\_timestamp table and is the same as the script name.
* No space before or after in the CONFIG files
* Tabs between values

# Supplementary information

## Space requirements for different cases

We tested different scenarios to ensure a reliable infrastructure. The additional advantage of it is that we can provide some estimates of how much space it will require. This is very dependent on selected data sources and how much overlap exists between those. For instance, there are a lot of redundancy in all 4 small molecule databases, leading to an overall less database space than if you loaded them in different databases.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data source | Disk space | Database space | Disk space | Database space |  |
| Clinvar | 5.2Gb |  | Human proteome | 4.2Gb | 4.1Gb |
| Ensembl  (1 genome) | 2-4Gb |  | Mouse Proteome | 650Mb |  |
| NCBI Gene | 9.Gb | 100Mb -13Gb | Swiss-Prot | 4.4Gb |  |
| GO | 33Mb | 30Mb | dbSNP | 11Tb |  |
| Publication | 505Gb | 9.6Gb  +4Gb (mapping) | Cellausorus | 95Mb |  |
| RefSeq | 5-7Gb |  | ChEMBL | 5.8Gb |  |
| SureCHEMBL | 48Gb |  | HMDB | 6.1Gb |  |
| Uniprot (ChEMBL) | 250Mb |  | Livertox | 456Mb |  |
| Reactome | 38Mb | 12Mb | GeneReviews | 1.4G |  |
| Clinical trial | 7.8Gb | 3.7Gb | OpenTargets | 29G |  |
| Taxonomy | 1.7Gb | 300Mb |  |  |  |