[DEPRECATED] Guide: Adding a new software (and make it executable from Cytomine-WebUI)

How it works

This page describes how to add a new software to Cytomine and make it executable from Cytomine-WebUI. Two main steps are involved in this process :

- The software is added to the Cytomine-Core (see basic principles in How to add a new software)
- The middleware used (RabbitMQ) to distribute messages between Cytomine-Core and the software is set up (see: Package Middleware)

These steps are transparent to the end-user.

After reading this page, if you are interested to know how to execute a software on specific regions of interest of an image from the Explore view, please read the [HOWTO] How to add a shortcut for a software (JobTemplate).

Adding a new software and register it as a RabbitMQ Job service

Currently, there are two ways to add you software to Cytomine :

1. With the Java Client (Cytomine Java client on GitHub)
2. With the Python client (Cytomine Python client on GitHub)

1.Using the Java Client

When you install Cytomine using our installation procedure, it comes with several analysis softwares that are pre-installed. These were actually defined in the Cytomine-Java-Client, in particular this class.
Let's follow an example (based on the Tissue Detect algorithm implemented in Python) to see how it works :

Adding a new software using the Java client

```
public static void addSoftwareTissueDetect(Cytomine cytomine) throws Exception {
        try{
            Software software = cytomine.addSoftware("TissueDetect", "createRabbitJobWithArgsService",
"ValidateAnnotation",
                    "python algo/detect_sample/detect_sample.py --cytomine_host $host --cytomine_public_key
$publicKey --cytomine_private_key $privateKey " +
                            "--cytomine_base_path /api/ " +
                            "--cytomine_working_path /software_router/algo/detect_sample/ " +
                            "--cytomine_id_software $cytomine_id_software " +
                            "--cytomine_id_project $cytomine_id_project " +
                            "--cytomine_predict_term $cytomine_predict_term " +
                            "--cytomine_max_image_size $cytomine_max_image_size " +
                            "--cytomine_erode_iterations $cytomine_erode_iterations " +
                            "--cytomine_dilate_iterations $cytomine_dilate_iterations " +
                            "--cytomine_athreshold_blocksize $cytomine_athreshold_blocksize " +
                            "--cytomine_athreshold_constant $cytomine_athreshold_constant " +
                            "--verbose true ");


            // set by server
            cytomine.addSoftwareParameter("cytomine_id_software", "Number", software.getId(), "", true, 400,
null, null, null, true);
            cytomine.addSoftwareParameter("cytomine_id_project", "Number", software.getId(), "", true, 500,
null, null, null, true);

            // set by user
            cytomine.addSoftwareParameter("cytomine_predict_term", "Domain", software.getId(), "", true, 600, "
/api/project/$currentProject$/term.json", "name", "name");
            cytomine.addSoftwareParameter("cytomine_max_image_size", "Number", software.getId(), "2048", true,
700);
            cytomine.addSoftwareParameter("cytomine_erode_iterations", "Number", software.getId(), "3", true,
800);
            cytomine.addSoftwareParameter("cytomine_dilate_iterations", "Number", software.getId(), "3", true,
900);
            cytomine.addSoftwareParameter("cytomine_athreshold_blocksize", "Number", software.getId(), "951",
true, 1000);
            cytomine.addSoftwareParameter("cytomine_athreshold_constant", "Number", software.getId(), "5",
true, 1100);

        } catch (CytomineException e) {
            log.error(e);
        }
    }
```

The "Software" object

The first step is to declare a "Software" object (using addSoftware function) with :

- a name (here "TissueDetect")
- a service (here "createRabbitJobWithArgsService")
- a complete command to launch the software (here "python algo/detect_sample/detect_sample.py .....")

The service

The service to use depends on how your software is constructed. At the moment, you have two possibilities :

1. You start your software with a command like :

First option

```
python myFolder/myAlgo/myAlgo.py --option1 optionValue1 --option2 optionValue2
```

In this case, you have to use "createRabbitJobWithArgsService".

2. Or with a commande like :

Second option

```
groovy -cp myFolder/myAlgo/dependencies.jar myFolder/myAlgo/myAlgo.groovy option1 option2
```

In this case, you have to use "createRabbitJobService".

The command

In the software_router Docker container, your will have to put your software code into a directory named "algo".
Your command should therefore starts with "algo/myScriptName/myScript.py".

After that, your software will probably use data gathered from Cytomine. If so, you have to add these parameters to your command line so that the script knows which Cytomine server and keys to communicate with:

```
--cytomine_host $host --cytomine_public_key $publicKey --cytomine_private_key $privateKey
```

The rest of the command line is up to you and your software needs.

Adding software parameters

Now you have to declare on Cytomine-Core the parameters required by your software (e.g. it can include parameters specific to your image analysis tasks e.g. threshold values for segmentation, model parameters for machine learning, etc.)..
Here's a example :

Adding a software parameter

```
cytomine.addSoftwareParameter("parameter_name", "parameter_type", software.getId(), "defaultValue", true,
OrderInYourCommand);
```

In this example, the "parameter_name" will then be asked to the end-users in the "Run Job" dialog box, the entered value will be communicated by Cytomine-Core to RabbitMq and your analysis script through the $parameter_name variable present in the command line.

Take a look at the exemple above (or anywhere in the SoftwareExample.java class) to insert your own parameters.

Auto-set software parameters

Some software parameters doesn't need to be add. They will be set by the Cytomine-Core module automatically.
These softwares are :

- host
- privateKey
- publicKey
- cytomine_id_software
- cytomine_id_project

So, these parameters will not be asked to the end-users.

Run the project

The last step is to add a line on the main class of the Java client and run it.
The class Execute.java is the main class, you have to add a line like this one :

```
SoftwareExample.mySoftware(cytomine);
```

Your software is now added to the Cytomine Core and you can launch it from the web interface (see screenshots at the end of this page).

2. Using the Python client

You can follow the same principles using the Python client.

Your software has to be localized by Docker/RabbitMQ. If you software runs on the same computer than the Cytomine-Core, then you simply have to copy your software code into the software_router container of that computer, in the algos/ subdirectory.
For example, you can do:

```
sudo docker-enter software_router
cd /software_router/algos/
mkdir my_algorithm/
exit
```

Then, let's assume you have a single source file (script.py) for you software, copy it from your local directory to the container using:

```
sudo docker cp script.py software_router:/software_router/algo/my_algorithm/script.py
```

Now you have to add your software once to the Cytomine-Core database.
We suggest to create an add_software.py file (see a full example below) that will use the add_software and add_software_parameters functions:

```
software = conn.add_software("My_algorithm", "createRabbitJobWithArgsService","ValidateAnnotation",
execute_command)
```

where execute_command is a variable that contains the string of the command line to execute your script.py (the example is inspired by the detect_sample python software), e.g.:

```
execute_command = "python algo/my_algorithm/script.py --cytomine_host $host --cytomine_public_key $publicKey --
cytomine_private_key $privateKey " + "--cytomine_base_path /api/ " + "--cytomine_working_path /software_router
/algo/detect_sample/ " + "--cytomine_id_software $cytomine_id_software " + "--cytomine_id_project
$cytomine_id_project " + "--cytomine_id_image $cytomine_id_image " + "--cytomine_predict_term
$cytomine_predict_term " + "--cytomine_max_image_size $cytomine_max_image_size " + "--cytomine_erode_iterations
$cytomine_erode_iterations " + "--cytomine_dilate_iterations $cytomine_dilate_iterations " + "--
cytomine_athreshold_blocksize $cytomine_athreshold_blocksize " + "--cytomine_athreshold_constant
$cytomine_athreshold_constant " + "--verbose true "
```

In this command, the following arguments are mandatory (in addition to your own software parameters):

"--cytomine_host $host "
"--cytomine_public_key $publicKey "
"--cytomine_private_key $privateKey "
"--cytomine_base_path /api/ "
"--cytomine_id_software $cytomine_id_software "
"--cytomine_id_project $cytomine_id_project "

For logging you also have to add after the add_software call these two lines:

```
conn.add_software_parameter("cytomine_id_software", software.id, "Number",0, True, 400, True)
conn.add_software_parameter("cytomine_id_project", software.id, "Number", 0, True, 500, True)
```

If some of your software parameters are related to some Cytomine "objects", you can specify a service to display user-friendly information, e.g. the following command will then allow to have a drop-down list to select on which image to apply your algorithm, and you will see in the Cytomine-WebUI drop-down list images and thumbnails (rather than imageids) in the "Run Job" execution dialog box (see screenshot below):

```
conn.add_software_parameter("cytomine_id_image", software.id, "Domain", "", True, "", False, "/api/project
/$currentProject$/imageinstance.json", "instanceFilename", "instanceFilename")
```

Here is a full add_software.py example that will add the "my_algorithm" algorithm (it requires the script.py file to be in the docker container as explained above. In this example the script.py is based on the same code than https://github.com/cytomine/Cytomine-python-datamining/blob/master/cytomine-applications/detect_sample/detect_sample.py and it can be tested on the DEMO-SEGMENTATION-TISSUE toy data project). You have to adapt variables to your cytomine_host, keys, location of your script file in the docker container, location of your cytomine_working_path in the docker container:

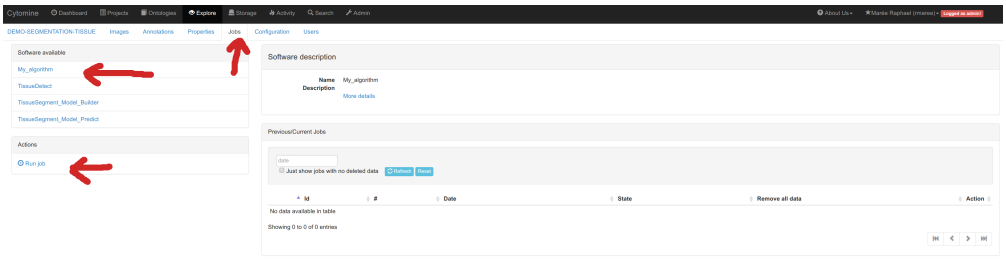
```
import cytomine
import sys
#Connect to cytomine, edit connection values
cytomine_host=""  # Cytomine core URL
cytomine_public_key=""  # Your public key
cytomine_private_key=" # Your private key
id_project=XXX  #Integer value corresponding to Cytomine project identifier where to add the software

#Connection to Cytomine Core
conn = cytomine.Cytomine(cytomine_host, cytomine_public_key, cytomine_private_key, base_path = '/api/',
working_path = '/tmp/', verbose= True)

execute_command = "python algo/my_algorithm/script.py --cytomine_host $host --cytomine_public_key $publicKey --
cytomine_private_key $privateKey " + "--cytomine_base_path /api/ " + "--cytomine_working_path /software_router
/algo/my_algorithm/ " + "--cytomine_id_software $cyto
mine_id_software " + "--cytomine_id_project $cytomine_id_project " + "--cytomine_id_image $cytomine_id_image "
+ "--cytomine_predict_term $cytomine_predict_term " + "--cytomine_max_image_size $cytomine_max_image_size " +
"--cytomine_erode_iterations $cytomine_erode_iterations " + "--
cytomine_dilate_iterations $cytomine_dilate_iterations " + "--cytomine_athreshold_blocksize
$cytomine_athreshold_blocksize " + "--cytomine_athreshold_constant $cytomine_athreshold_constant " + "--verbose
true "
#define software parameter template
software = conn.add_software("My_algorithm", "createRabbitJobWithArgsService","ValidateAnnotation",
execute_command)
conn.add_software_parameter("cytomine_max_image_size", software.id, "Number", 2048, True, 10, False)
conn.add_software_parameter("cytomine_erode_iterations", software.id, "Number", 3, True, 30, False)
conn.add_software_parameter("cytomine_dilate_iterations", software.id, "Number", 3, True, 40, False)
conn.add_software_parameter("cytomine_athreshold_constant", software.id, "Number", 5, True, 50, False)
conn.add_software_parameter("cytomine_athreshold_blocksize", software.id, "Number", 951, True, 60, False)

#for logging (set by server)
conn.add_software_parameter("cytomine_id_software", software.id, "Number",0, True, 400, True)
conn.add_software_parameter("cytomine_id_project", software.id, "Number", 0, True, 500, True)
# for user-friendly drop-down menu
conn.add_software_parameter("cytomine_id_image", software.id, "Domain", "", True, "", False, "/api/project
/$currentProject$/imageinstance.json", "instanceFilename", "instanceFilename")
conn.add_software_parameter("cytomine_predict_term", software.id, "Domain", "", True, "", False, "/api/project
/$currentProject$/term.json", "name", "name")
#add software to a given project
addSoftwareProject = conn.add_software_project(id_project,software.id)
```

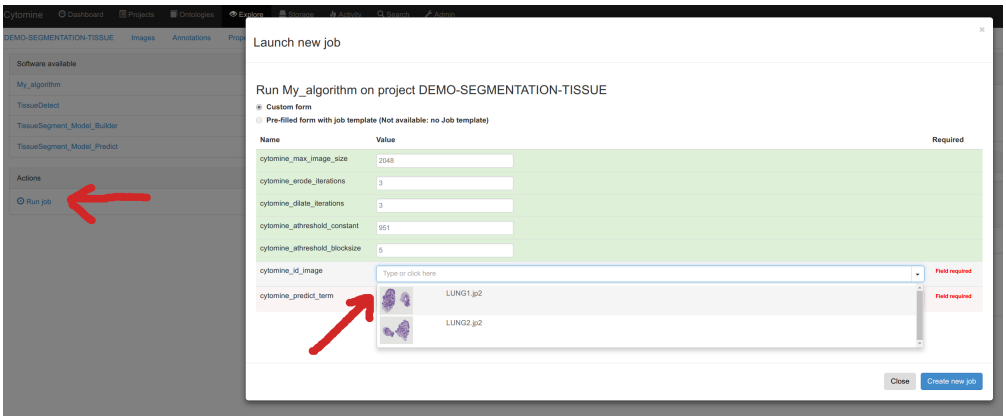
Execute the add_software.py script to add the software to your Cytomine Core:

```
python add_software.py
```

You will then be able to execute your software throught the Cytomine-WebUI. In practice, Cytomine-Core and RabbitMQ will append the parameter values encoded by the end-user in Cytomine-WebUI dialog box through the command-line arguments that executes the script contained in the software_router container.

Please notice in the second screenshot the user-friendly drop-down image list for Images and the list of ontology term names):



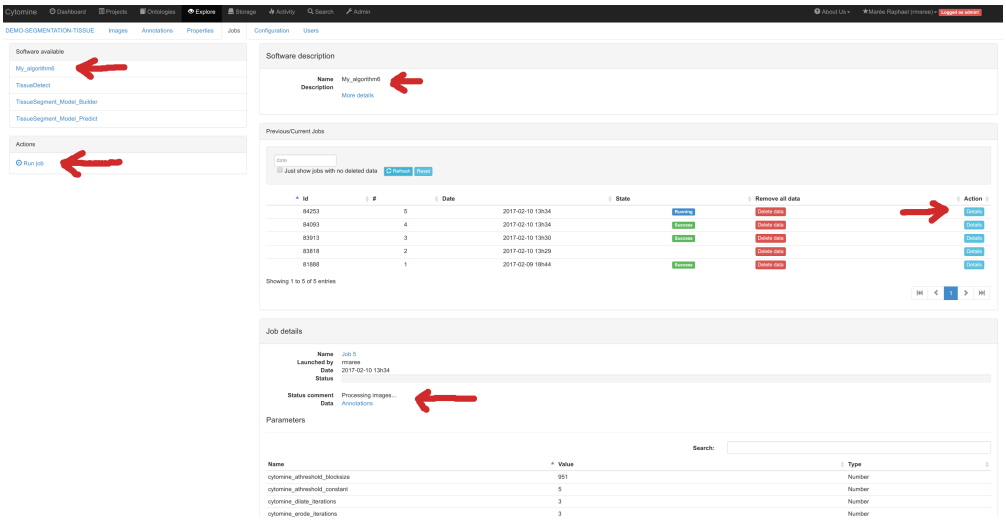


2. Logs and debugging

It is recommended that in your software you include some communications with Cytomine-Core to give information to the end-user about the status of its running job (these status messages will be displayed in the Jobs tab in the Cytomine-WebUI). It can be done by using the update_job_status, e.g. in Python (see other examples in our Cytomine-Datamining applications):

```
progress_msg = "Analyzing image %s (%d / %d )..." %(id_image,i,len(images))
job = conn.update_job_status(job, status = job.RUNNING, progress = progress, status_comment = progress_msg)
```

These status messages will be displayed in the Cytomine-WebUI Job tab:



However, for debugging, these status messages might not be enough. To see the full execution logs, you can enter into the software_router docker container and see the outputs of your job.

```
sudo docker-enter software_router
cd /software_router/algo/   ;; code
cd /software_router/algo/logs/
ls -altr

root@6041e74c788b:/software_router/algo/logs# ls -altr
total 620
-rw-r--r--  1 root root 349657 Feb 10 12:15 queueSoftwareTissueDetect-10-2-2017_12-15-40-423.log
-rw-r--r--  1 root root     98 Feb 10 12:29 queueSoftwareMy_algorithm-10-2-2017_12-29-09-101.log
drwxr-xr-x 18 root root   4096 Feb 10 12:30 ..
drwxr-xr-x  2 root root   4096 Feb 10 12:30 .
-rw-r--r--  1 root root 266401 Feb 10 12:30 queueSoftwareMy_algorithm-10-2-2017_12-30-44-096.log
```