

---

# CONTROL-CORE

*Release 0.1*

CONTROL-CORE

Aug 18, 2023



# CONTENTS

<b>1</b>	<b>Contents</b>	<b>3</b>
1.1	Installation . . . . .	3
1.2	Usage . . . . .	6
1.3	Adding RTC Module to Raspberry Pi . . . . .	10
1.4	Concore editor . . . . .	12
1.5	The Concore Action . . . . .	15
1.6	The Concore Action Dev docs . . . . .	18
1.7	Shared Memory communication in Concore . . . . .	23
1.8	Concore.hpp . . . . .	26
	<b>Index</b>	<b>29</b>



**CONTROL-CORE** is a framework for Closed-Loop peripheral neuromodulation control systems

Check out the [Installation](#) section for further information on installing the CONTROL-CORE framework. Check out the [Usage](#) section for further information on writing `concore` programs.

---

**Note:** This project is under active development.

---

If you use the **CONTROL-CORE** framework in your research, please cite the below paper:

- Kathiravelu, P., Arnold, M., Fleischer, J., Yao, Y., Awasthi, S., Goel, A. K., Branen, A., Sarikhani, P., Kumar, G., Kothare, M. V., and Mahmoudi, B. **CONTROL-CORE: A Framework for Simulation and Design of Closed-Loop Peripheral Neuromodulation Control Systems**. In IEEE Access. March 2022. <https://doi.org/10.1109/ACCESS.2022.3161471>



## CONTENTS

## 1.1 Installation

We have refactored and released the `concore` protocol as an open-source project.

### 1.1.1 From the Public Repository

If you are using `concore` (without committing), the only installation steps required are to clone the `concore` public repository and install the `concore` dependencies as below.

First, clone the `concore` repository from <https://github.com/ControlCore-Project/concore>.

```
$ cd concore
$ pip install -r requirements.txt
```

If you like to contribute your changes back to `concore` open-source framework, please fork the repository and send pull requests. We will review and merge your pull requests.

### 1.1.2 From the Private Repository

`concore` is now a public repository, as we refactor and make the code available to public. Therefore, the below information is mostly becoming obsolete.

To check out `concore` from the private repository, certain steps must be made to checkout the source code, since GitHub has discontinued password-based cloning.

First, check whether your computer has SSH configured.

```
$ ls -al ~/.ssh
```

If you have already configured, you will see files similar to the below.

```
id_rsa.pub
id_ecdsa.pub
id_ed25519.pub
```

If not, make sure to configure SSH and create new keys before proceeding to the next step.

If no keys found in the above step, please follow these steps.

```
$ ssh-keygen -t ed25519 -C "your_email@example.com"
> Generating public/private algorithm key pair.
> Enter a file in which to save the key (/Users/you/.ssh/id_algorithm): [Press enter]
> Enter passphrase (empty for no passphrase): [Type a passphrase]
> Enter same passphrase again: [Type passphrase again]
```

You may just click *Enter* for all of the above. Now you have created the keys, if you did not have already.

Now that you have the keys, start the ssh agent.

```
$ eval "$(ssh-agent -s)"
> Agent pid 59566
```

Add your SSH private key to the ssh-agent and store your passphrase in the keychain. If you created your key with a different name, or if you are adding an existing key that has a different name, replace `id_ed25519` in the command with the name of your private key file.

First finding the name:

```
$ ls -al ~/.ssh
```

Then,

```
$ ssh-add -K ~/.ssh/id_ed25519
```


Now that you have added your key, you can proceed to the next step of adding the key to your GitHub profile. In the upper-right corner of any page, click your profile photo, then click Settings.

Open and copy the contents of `~/.ssh/id_ed25519.pub` (or a similar `.pub` file from the `.ssh` directory).


```
$ cat ~/.ssh/id_ed25519.pub
```

In the upper-right corner of any page in GitHub after you have logged, click your profile photo, then click Settings.




**octocat**  
Octocat


×


 Edit status


---


 Your profile


---


 Your repositories


 Your projects


 Your codespaces

 Your organizations


 Your enterprises


 Your stars


 Your sponsors

 Your gists


---


 Upgrade

 Feature preview

 **Settings**

---

 GitHub Docs

 GitHub Support

---

Sign out

In the user settings sidebar, click SSH and GPG keys.

Click New SSH key or Add SSH key.

In the “Title” field, add a descriptive label for the new key. Paste your key (the content of `~/.ssh/id_ed25519.pub` or a similar file that you copied in the previous step) into the “Key” field.

Click Add SSH key. Click Add SSH key. Now, you are ready to checkout the private GitHub repository with the below commands, as long as you are already added to the respective repository.

```
$ git clone git@github.com:ControlCore-Project/concore20.git
$ cd concore20
$ pip install -r requirements.txt
```

## 1.2 Usage

### 1.2.1 Introduction

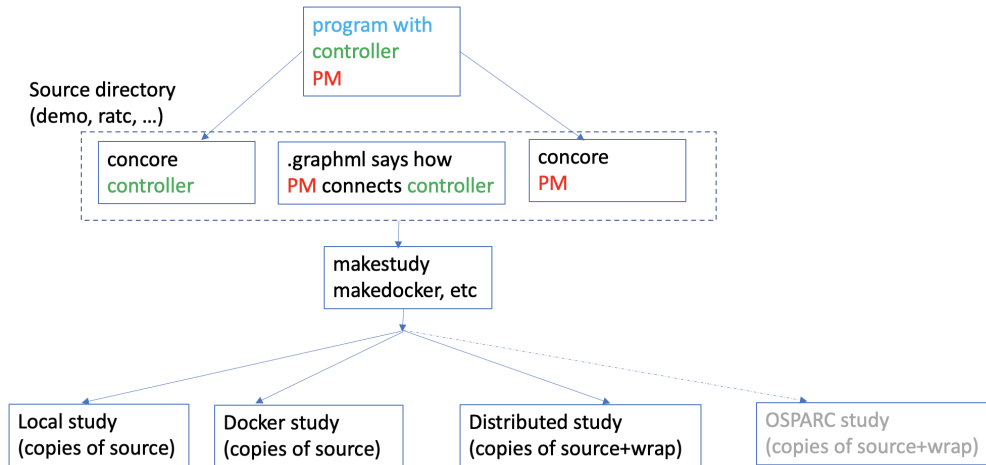
CONTROL-CORE is a framework for peripheral neuromodulation control systems, which consist of models of organs called physiological models (PMs) and controllers that interact with PMs. The controller and the PM are represented by workflow diagrams (.graphml) that indicate how they are connected together. Nodes in the workflow diagram are programs that are written using the `concore` protocol. Edges in the workflow diagram show how the nodes are interconnected. To create a system with `concore`, you need to write at least two programs (or use existing programs) that are referenced in the workflow diagram. These programs need to use the `concore` methods to receive and transmit neurostimulation data and response.

To develop workflows in the CONTROL-CORE framework, one must go through two steps. First, to develop the programs following the `concore` protocol. Second, is to create a workflow from the created programs.

### 1.2.2 Writing `concore` Programs

The `concore` protocol requires the developer (the application developer who develops programs to run on the CONTROL-CORE framework) to write a new `concore` study consisting of two or more interacting programs or split one into separate and programs.

If you are migrating an existing program into `concore`, the splitting approach is the one you need to take. If you are writing programs from scratch, it makes more sense to write programs such as a PM and a controller as independent programs and construct a `concore` study from these programs.



Now, let's look into how to split an existing program to use concore as specified above, with a minimal example.

### 1.2.3 Adapting your program to use concore protocol

First, let's consider the below simple program, that does not adhere to the concore protocol, and appears as a “Combined program” with both PM and controller methods in it.

#### Combined program (non-concore)

The above simple code represents your existing program that does not adhere to concore protocol. That means, it consists of and methods in a single integrated program.

Now, let's see how to break this into two different concore programs, each representing and You must have noticed we have been consistently using colors in our code samples. These colors have a meaning.

Code segments that represent the methods are in .

Code segments that represent the methods are in .

Code segments that are specific to your application, and not specific to your PM or controller are in . These segments will likely end up in your both concore PM and controller programs as we will see shortly.

#### Separated into concore programs

Let's convert the above program to use concore now. concore specific code segments are in black in the two concore programs (controller and PM) displayed below.

##### The Controller

The respective concore controller, saved as controller.py:

##### The PM

The concore PM, saved as pm.py:

The concore Git repository comes with some samples. The above controller.py and pm.py can be found in the demo directory. The demo directory also comes with other sample controller and pm implementations, and workflows that connect them - stored as GraphML files.

## 1.2.4 Building Workflows with concore

CONTROL-CORE leverages [DHGWorkflow](#) to create such workflows graphically. DHGWorkflow is a browser-based lightweight workflow composer, which lets us visually create directed hypergraphs (DHGs) and save them as GraphML files. concore consists of a parser that interprets the GraphML files created by DHGWorkflow into workflows consisting of concore programs that interact with each other in a DHG.

concore comes with a set of samples in the demo directory. As a learning exercise, these samples do not have the actual physiological models and controllers. Rather, they demonstrate the concore protocol with minimal complexity.

Let's run a sample workflow!

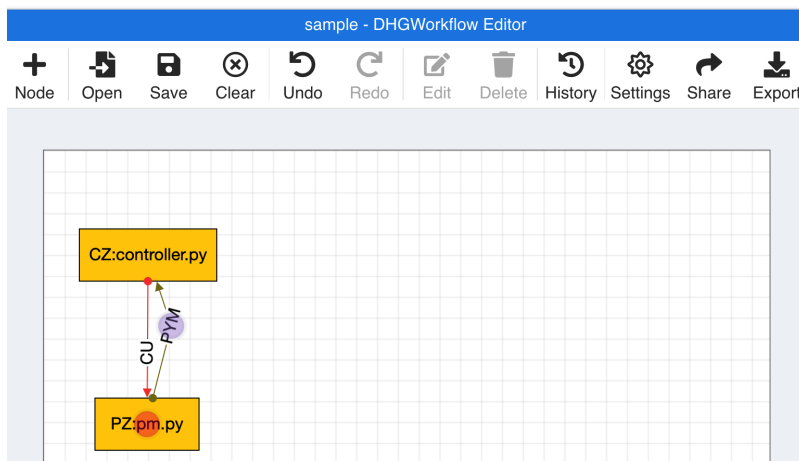
First, use your favorite editor to create controller and pm or use the existing controller.py and pm.py in the demo directory (elaborated in the above section). Similarly, a sample workflow with the above sample controller.py and pm.py is also saved in the demo directory as sample.graphml in the demo directory. The demo directory also has several other controller, pm, and workflow sample implementations.

Let's go to the demo directory to compose and edit the workflows.

```
$ cd demo
```

Then, use the editgraph command to pop up the browser that opens DHGWorkflow. Then create a graphml file, similar to the one demonstrated below, by dragging and dropping nodes and edges. Make sure to label the nodes and edges correctly, making sure to use the correct case (the programs are case sensitive). The below commands are specific to POSIX (Linux/MacOS) environments. If you use Windows, please make sure to use \ instead of /.

```
../editgraph
```



Use the “Save As” option and type “sample1.” That will save the workflow as sample1.graphml in your Downloads directory.

Now, use the getgraph command to copy the graphml files from your Downloads directory to the current directory.

```
../getgraph
```

This copies “sample1.graphml” to current directory “demo”.

Now, go back to main concore directory to build the study that uses the programs referenced by the workflow.

```
cd ..
```

Run the `makestudy` command of `concore` which creates files and folders necessary for the workflow execution.

```
./makestudy demo/sample1
```

This would create a “sample1” directory with a first copy of the source files in “src”.

Go to the sample1 directory.

```
cd sample1
```

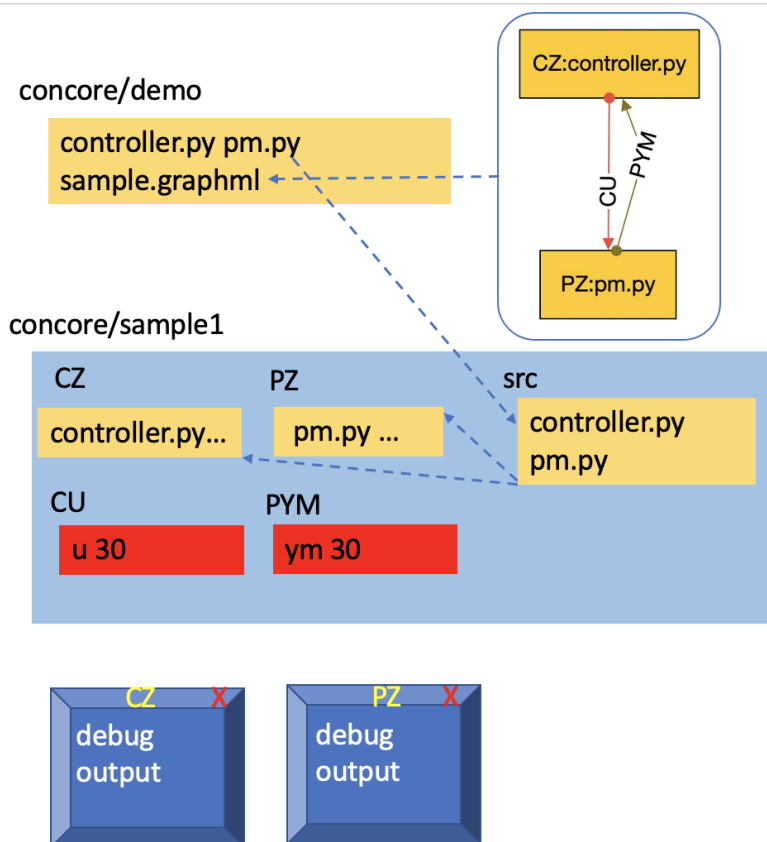
Now, run the build command, which functions like a compiler.

```
./build
```

This,

- creates CZ directory corresponding to node and copies `controller.py` into CZ.
- creates PZ directory corresponding to node and copies `pm.py` into PZ.
- creates CU and PYM directories corresponding to the edges of the graph.

The below diagram demonstrates the files and folders after running the build command. The dashed arrows indicate the multiple copies of files that are being used.



Now, set a maxtime attribute to list the number of iterations to be made by the controller and the PM.

```
./maxtime 30
```

This copies 30 into edges so both programs finish at the same time

Next, use `debug` command to execute the workflow.

```
./debug
```

The debug windows pop up.

Remember to close the debug windows with mouse, once the results are obtained.

Instead of debug, you may use the run command to execute the workflows without debug windows.

```
./run
```

At last, clean up the resources with the below commands.

```
./stop
```

Executing a stop command is always needed for docker. But it is optional in this example as we did not use Docker.

```
./clear
```

Clearing is also always needed for Docker or if rerunning. Again optional in this example.

Finally, you must destroy the sample1 directory, making sure to stop and clear first in Docker executions before using the destroy command.

```
cd ..
```

```
./destroy sample1
```

You may see a few error messages and warnings. They are expected and can be safely ignored.

## 1.3 Adding RTC Module to Raspberry Pi

The **Real-Time Clock** (RTC) module is a device that keeps time even when the power is off or the battery is removed.

By default, the Raspberry Pi is intended to be connected to the Internet via Ethernet or Wi-Fi to update the time automatically from global NTP (Network Time Protocol) servers.

However, an RTC module is necessary for running Concore on a Raspberry Pi without a network connection, ensuring the preservation of accurate time even during power outages.

### 1.3.1 Hardware Requirements

1. Raspberry Pi
2. Micro USB power supply
3. SD card with Raspbian OS installed
4. USB keyboard and mouse
5. HDMI-capable monitor
6. Real-Time Clock module for Raspberry Pi (e.g., DS3231)

### 1.3.2 Connections

Connect the RTC module(DS3231) to the Raspberry Pi's GPIO pins as follows:

- RTC module to Raspberry Pi GPIO Pins:
  - VCC ———> Pin No. 1 (3.3V)
  - Data ———> Pin No. 3 (GPIO2)
  - Clock ———> Pin No. 5 (GPIO3)
  - NC ———> Pin No. 7 (GPIO4)
  - GND ———> Pin No. 9 (GND)
- Additionally:
  - Connect a USB keyboard and mouse to the Raspberry Pi.
  - Connect a monitor to the Raspberry Pi's HDMI port.
  - Connect the power supply.

### 1.3.3 Setting Up and Testing I2C

Before using the RTC module, you need to set up the I2C interface on your Raspberry Pi:

1. Option A: Run the following command in the terminal::

```
sudo raspi-config
```

Select “Advanced” and then enable I2C.

2. Option B: Select “Menu” > “Preferences” > “Raspberry Pi Configuration” > “Interfaces” > “Enabled I2C”.

After configuring I2C, reboot the Raspberry Pi. To verify that I2C is working correctly, run the following commands in the terminal::

```
sudo apt-get install python-smbus i2c-tools
sudo i2cdetect -y 1
```

You should observe the address of the RTC module (e.g., 0x68) being displayed, indicating that the I2C configuration is functioning properly.

### 1.3.4 Setting RTC Time on Raspbian Jessie

- Run the following command in the terminal::

```
sudo nano /boot/config.txt
```

Add the line “dtoverlay=i2c-rtc,ds3231” at the end of the file.

- Reboot RaspberryPi
- Run the following command in the terminal::

```
sudo i2cdetect -y 1
```

The output should display “UU” instead of the RTC module's address (e.g., 0x68), indicating that the kernel driver is working.

- Disable the “fake hwclock” to prevent interference with the RTC::

```
sudo apt-get -y remove fake-hwclock
sudo update-rc.d -f fake-hwclock remove
```

- Disable the “fake hwclock” to prevent interference with the RTC::

```
sudo hwclock -D -r
```

Note that the RTC module may initially have the wrong time and needs to be set once.

- Connect the Raspberry Pi to Ethernet or Wi-Fi to allow it to sync the correct time from the Internet.
- Run:

```
# to write the time
sudo hwclock -w

# to read the time
sudo hwclock -r
```

Ensure that the RTC module remains connected so that the time is saved. From now on, the Raspberry Pi will automatically sync the time from the RTC module at boot, even without an Internet connection.

## 1.4 Concore editor

### 1.4.1 Introduction

Concore Editor is a fork of **DHGWorkflow**, visual Directed Hypergraph Workflow Composer, finetuned to operate as a frontend for concore.

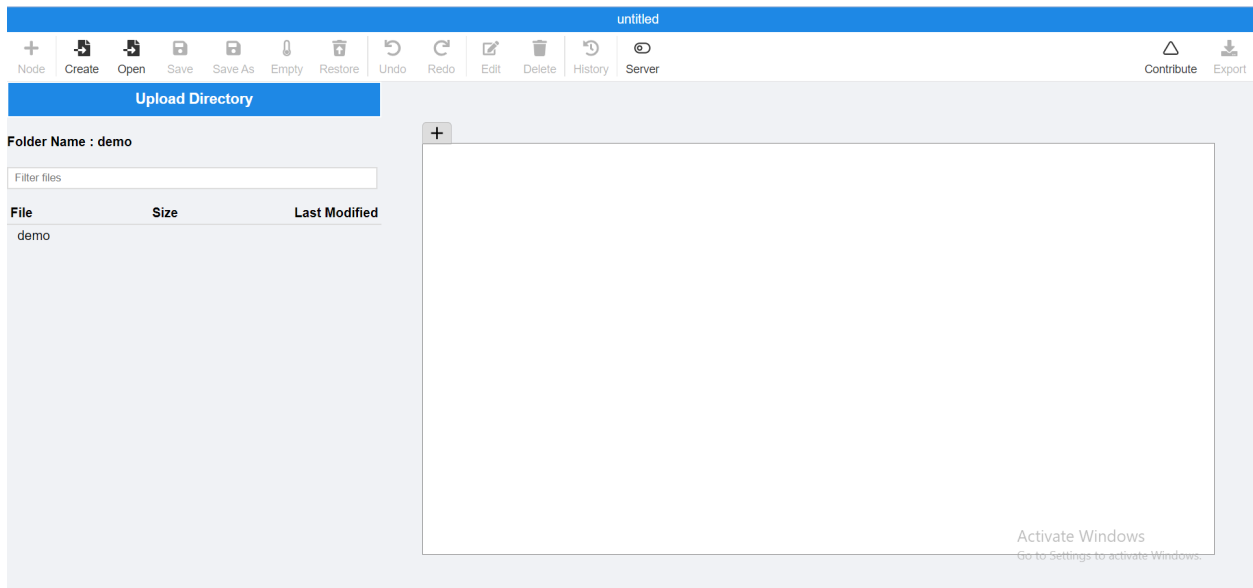
Key Features of the concore Editor:

Export-import graph as a graphml file. Export graph as JPEG/PNG Undo-Redo Actions Drag Drop Nodes Create Edges easily Bend Edges and many more!

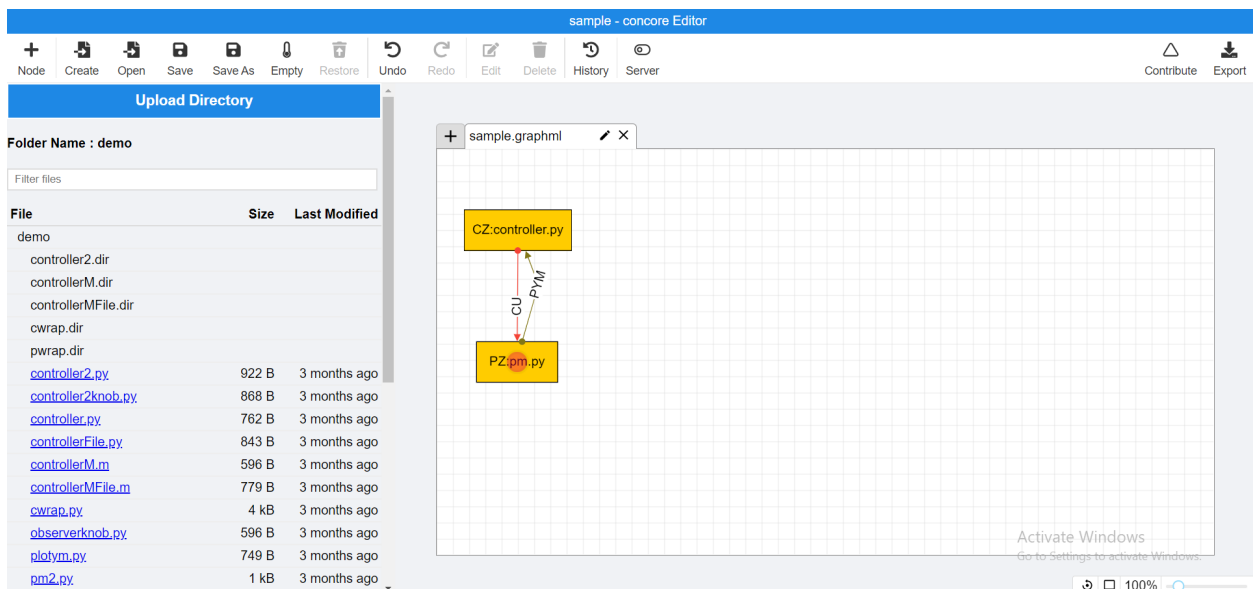
### 1.4.2 How to use?

- Open **concore editor** by clicking on <https://controlcore-project.github.io/concore-editor/> .
- Click on Upload Directory button and upload the directory containing **.graphml** files.





- Now, click on the directory which was uploaded and then click on the `.graphml` to open it in workspace.



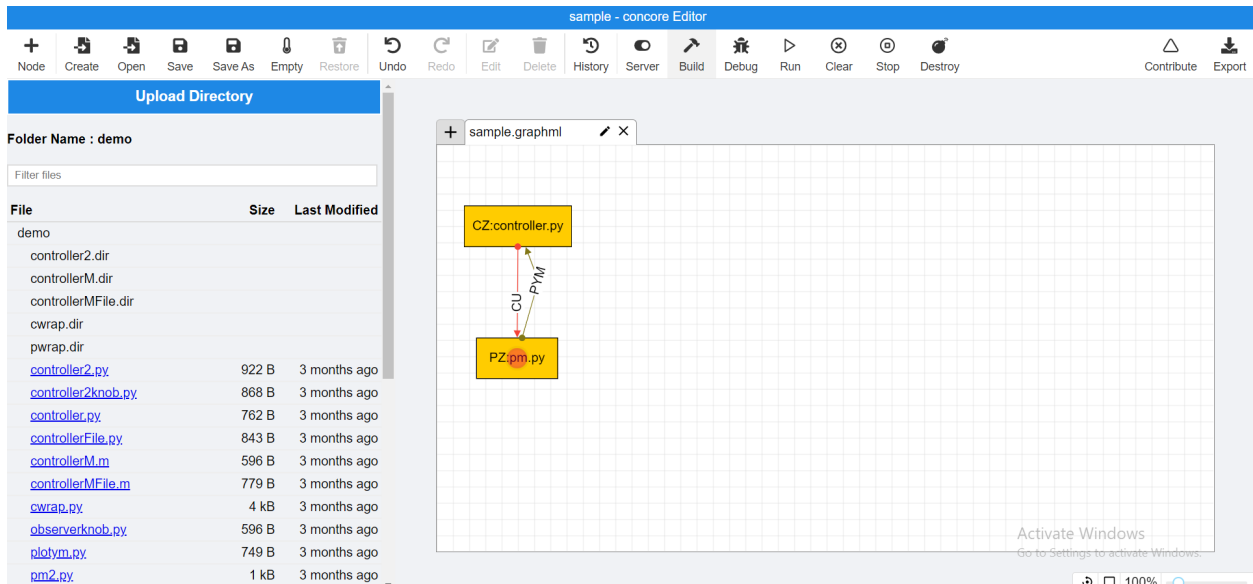
- Edit the `.graphml` as wanted and save it.
- To build the `.graphml` file, concore fri(server) needs to be switched on. To do so:

First, clone the concore repository from <https://github.com/ControlCore-Project/concore>.

```
$ cd concore
$ pip install -r requirements.txt
$ cd fri
$ cd server
$ python main.py
```

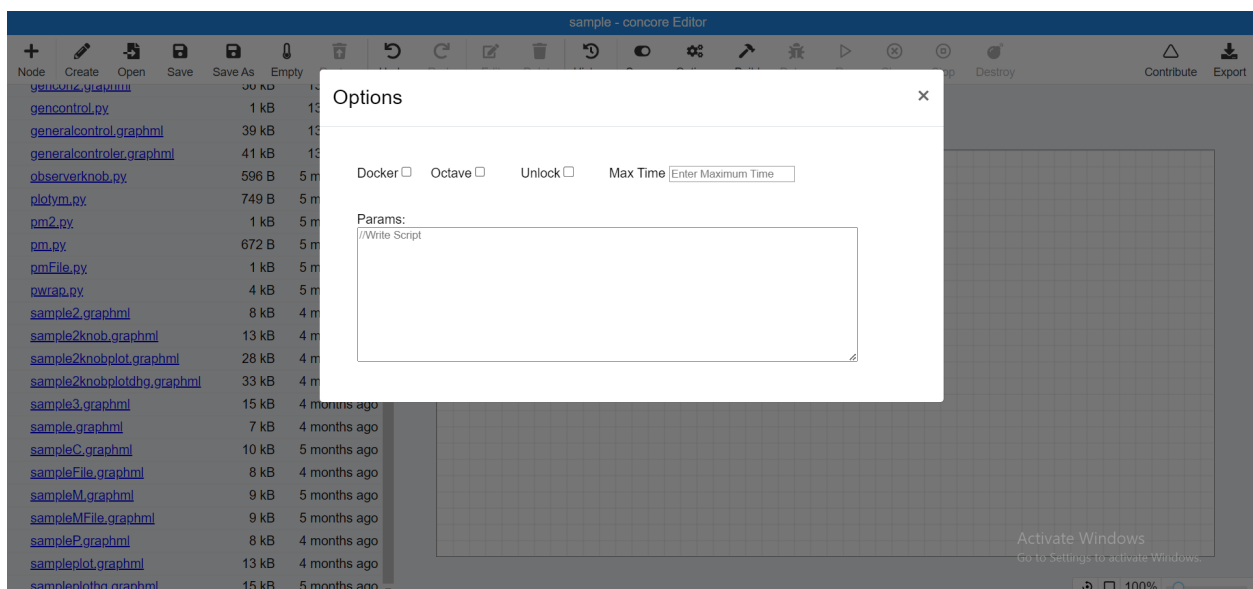
It will start the Flask server.

- After switching on the server, click on the **server** button in concore editor's tab bar. You will see all the concore functions appearing there.

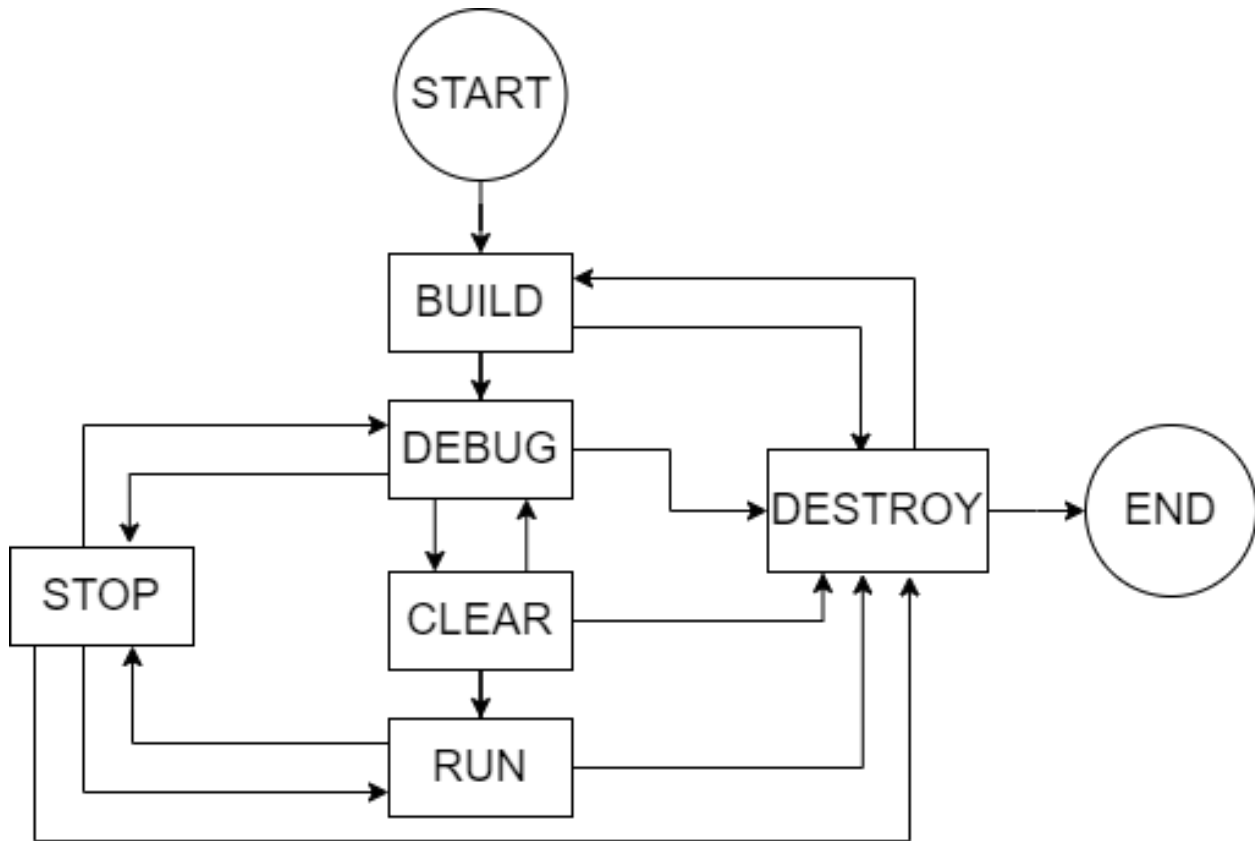


- There is an options button which can be seen. It provides the flexibility to take input from user about how a particular study should be build.
  - Checkmark docker checkbox if study to be build as docker image.
  - Checkmark octave checkbox if uploaded graphml contains matplotlib extension and has to be build via octave.
  - Enter the time for which you want the study to debug in maxtime field.
  - Enter customize parameters in params if there are any.

**NOTE:** The options can be changed only before executing build or clear.



- Last step is to perform concore functions on opened .graphml in order given in below flowchart.



**Note:** Executing concore functions in different order can result in errors and breakages.

## 1.5 The Concore Action

### 1.5.1 Introduction

This feature makes it easy for users to share their studies, examples, and program files. With this feature, you can contribute your study without having to use GitHub.

### 1.5.2 How to use?

#### 1.5.3 1. Creating a new Study

Using concore editor:

- Open **concore editor** by clicking on <https://controlcore-project.github.io/concore-editor/>
- Click on **Contribute** button at top-right of the editor
- To contribute examples, concore fri(server) needs to be switched on. To do so refer <https://control-core.readthedocs.io/en/latest/concoreeditor.html#how-to-use>
- **Fill in the details as follow:-**

**These are the necessary arguments**

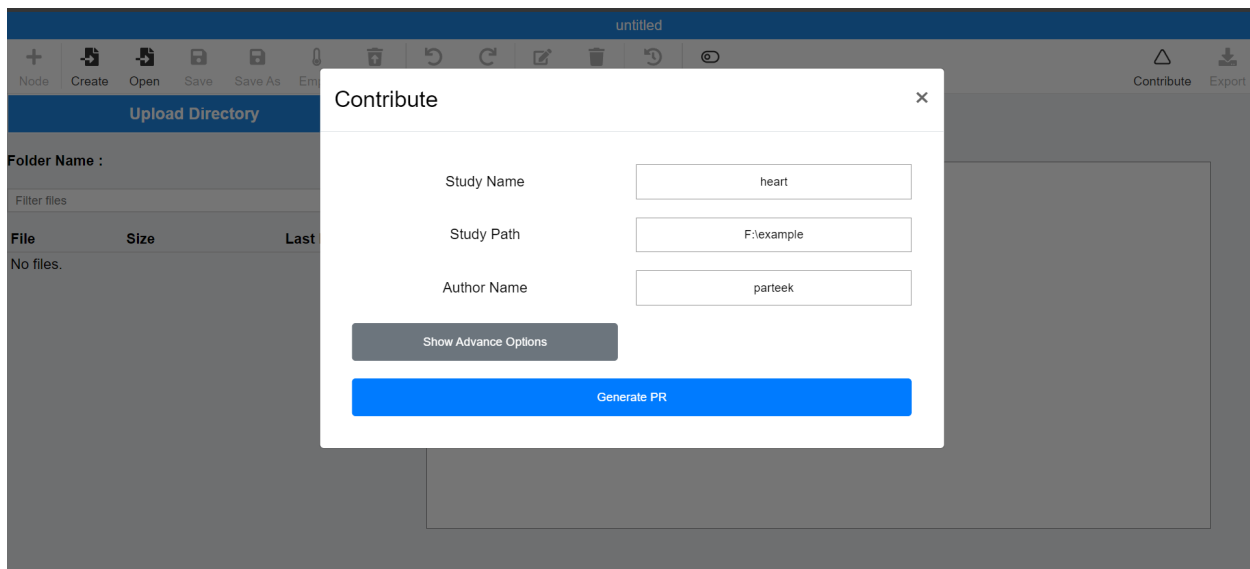
- Study Name - title of the study
- Study Path - full directory path where the study is located as shown below.
- Author Name - name of the person who created that study

**You can also provide the optional arguments by clicking Show Advance Options button**

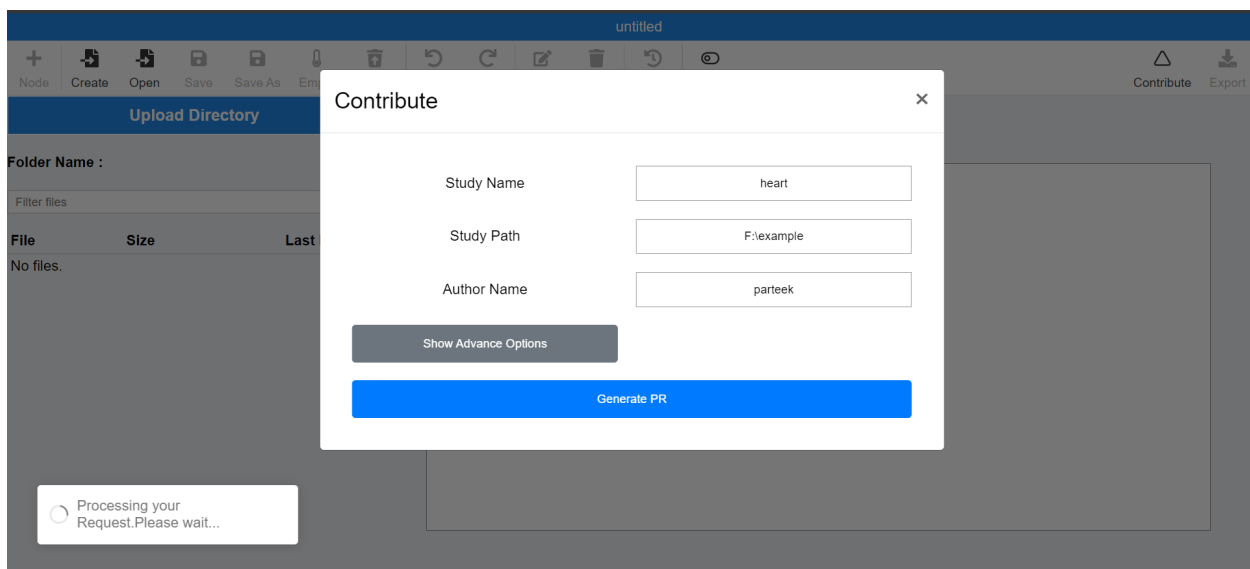
- Branch Name - the branch name you want
- Title of Study - a short title which will displayed as title of pull request at github
- Description of Study - a description which will be displayed as body of the pull request at github

Note: If you are not familiar with Github, we recommend you to skip the optional fields

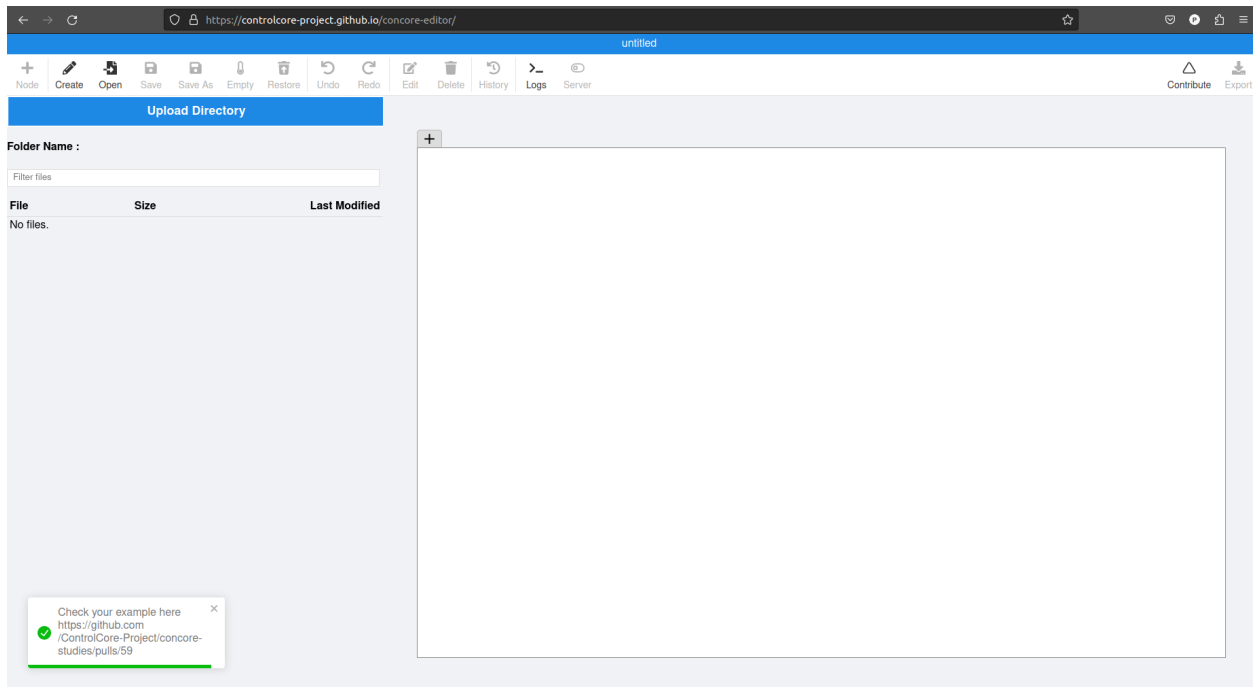
Tip: To copy the file path , follow this right-click on the file > click on copy path



- Click on Generate PR button and wait until it processes your request



- After it gets completed, you can check at the link given in the message



### 1.5.4 Using command-line tools:

- pass the above mentioned arguments to the `contribute` command

```
./contribute <Study-Name> <Full-Path-To-Study> <Author-Name> <Branch-Name> <PR-Title>
<PR-Body>
```

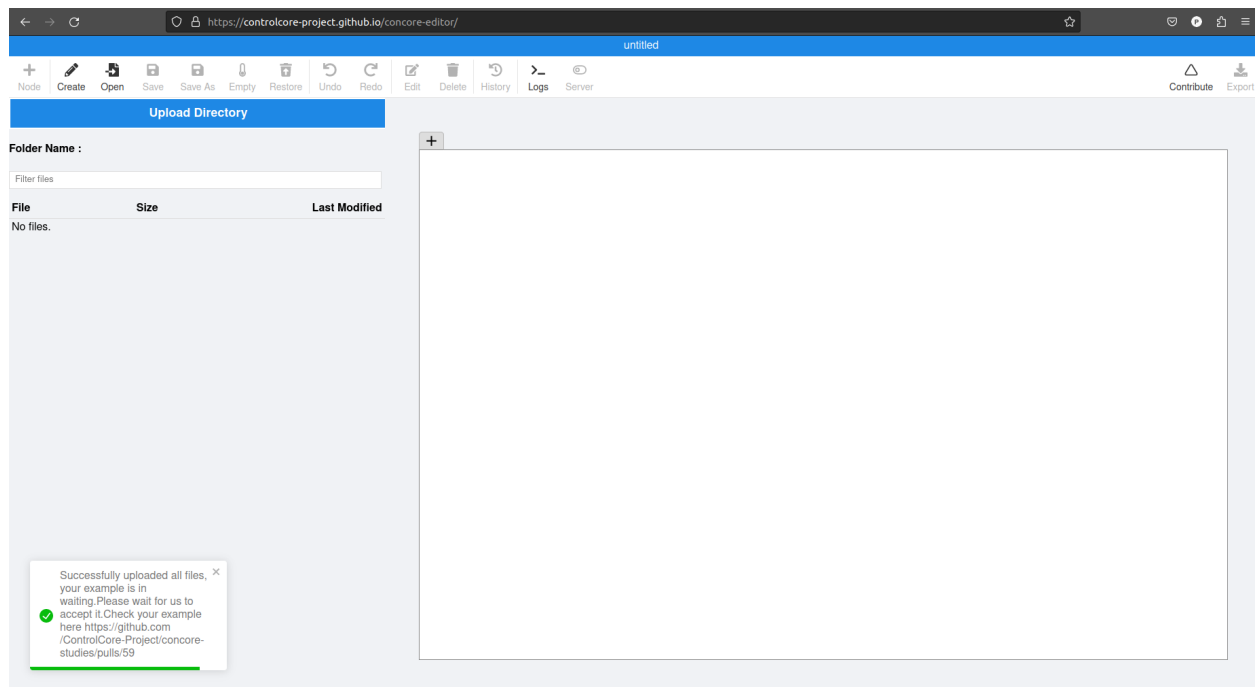
- e.g:

```
./contribute heart F:\example parteeek heart-study "Added heart study"
```

It will create a Pull request authored by parteeek, on a new branch named heart-study with title “Added heart study”

### 1.5.5 2. Updating existing Study

- Make the changes in your local study
- Again create the PR either using `concore-editor` or CLI. Make sure the values for Author name, Study Name and Branch name (if entered before) should be the same as you entered before while contributing study



Note: Successful submission doesn't mean that your study is added to our repository, It will take time for us to review and accept the study. So, you can mention your email in the description field mentioned above so that if your study got accepted then we will let you know.

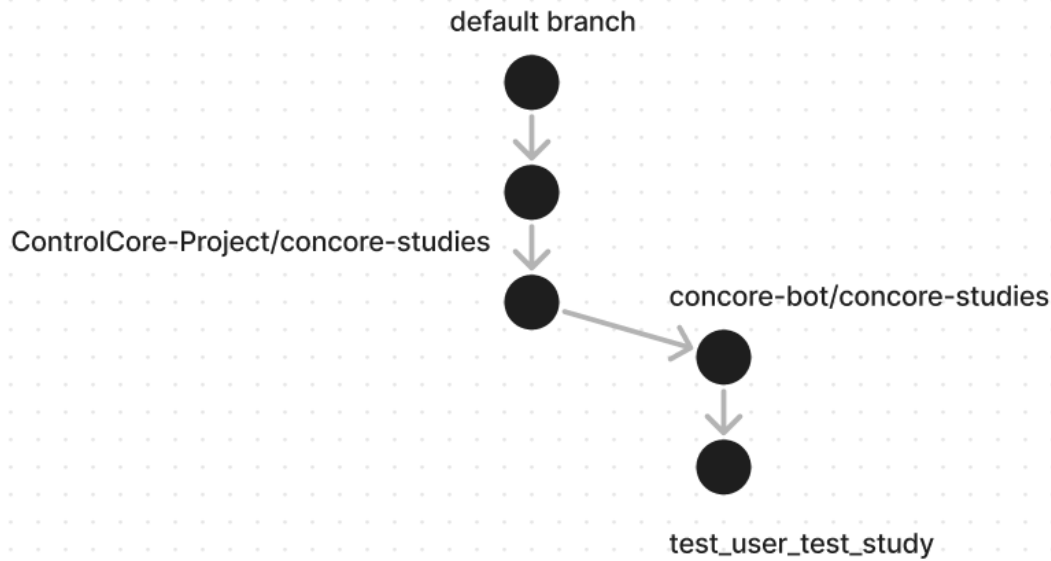
## 1.6 The Concore Action Dev docs

### 1.6.1 Introduction

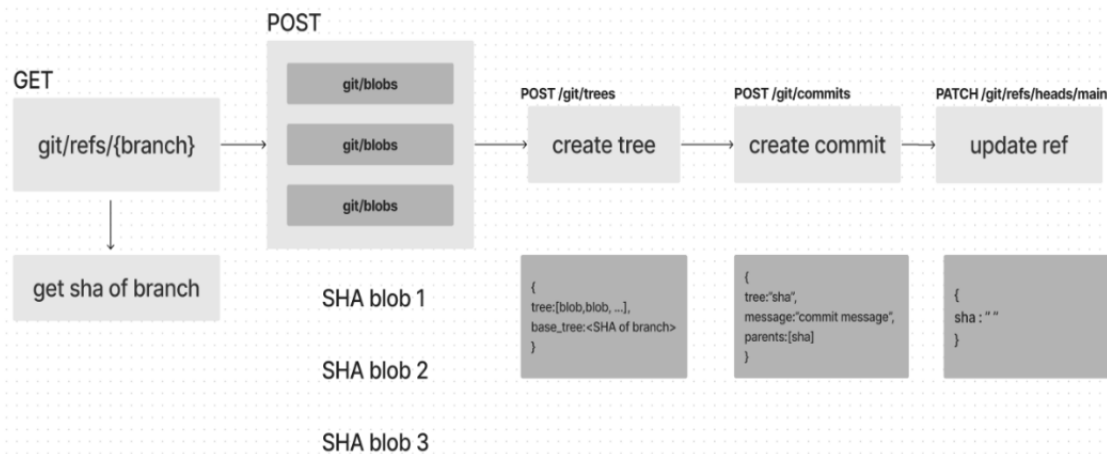
The current implementation uses Github REST API to push code to github by authenticating with github fine grained access token (refer <https://github.blog/2022-10-18-introducing-fine-grained-personal-access-tokens-for-github/>)

### 1.6.2 How it works

- On triggering contribute action in fri server, it executes the `contribute.py` script
- **Let's assume the parameters provided are:**
  - Study Name - test study
  - Study Path - F:\example
  - Author Name - test user
- It will create a new branch named `AuthorName_StudyName` i.e `test_user_test_study` in the bot repository `concore-bot/concore-studies` which references the default branch in `ControlCore-Project/concore-studies`



- Now creating Github blob (refer <https://docs.github.com/en/rest/git/blobs> ) for each file in the directory here F:\example. After creating blobs , create a new Git tree (refer <https://docs.github.com/en/rest/git/trees>)
- After creating Github tree, change the reference of the newly created branch to the that tree



- Now the changes have been pushed to github, and for creating pull request the `pull_request.yml` workflow is runned which creates pull request to upstream repo using persoanl access token of upstream repo

Note: You need to store the personal access token of upstream repo as github secret in bot repo

### 1.6.3 Creating token for bot

This token is used to create branches and push studies to bot repo. After pushing studies to bot repo it dispatch `pull_request.yml` workflow.

- Open the ControlCore-Project account at github and navigate as follow:  
settings > Developer settings > Personal access token > Fine-grained access token > Generate New token  
Or  
click <https://github.com/settings/personal-access-tokens/new>
- Fill token name, description expiration time
- Under the Repository access section select **Only select repositories** to select the repository for which you want to provide the access
- Under the Permissions , provide the read-write permissions for **Contents** and **Actions** in the Repository permission
- Click **Generate token** button
- Then copy the generated token and hash it using this website in base64 encoding <http://www.unit-conversion.info/texttools/base64/> . Note this hashing is important as Github automatically revoke any token present in the code.
- Place the token in `contribute.py` script at <https://github.com/ControlCore-Project/concore/blob/dev/contribute.py#L7>

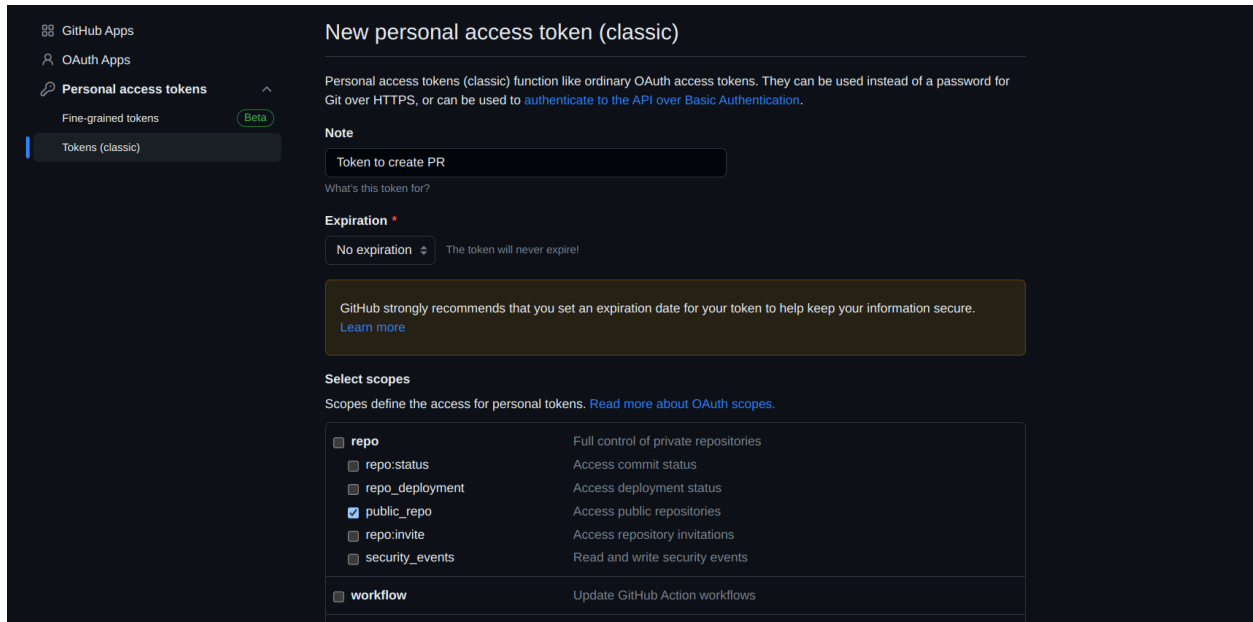
Note: This token has max life time of 1 year and needs to be updated after this duration

### 1.6.4 Creating token for workflow

This token is used to create pull request to upstream repo using bot account as author of pull request.

- Open `concore-bot` account at Github
- Create a Personal access token of bot account at <https://github.com/settings/tokens/new>
- Fill the details as shown below:





GitHub Apps

OAuth Apps

Personal access tokens

Fine-grained tokens

Tokens (classic)

## New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

**Note**

Token to create PR

What's this token for?

**Expiration \***

No expiration ⌵ The token will never expire!

GitHub strongly recommends that you set an expiration date for your token to help keep your information secure. [Learn more](#)

**Select scopes**

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input type="checkbox"/> <b>repo</b>	Full control of private repositories
<input type="checkbox"/> repo:status	Access commit status
<input type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input type="checkbox"/> repo:invite	Access repository invitations
<input type="checkbox"/> security_events	Read and write security events
<input type="checkbox"/> <b>workflow</b>	Update GitHub Action workflows

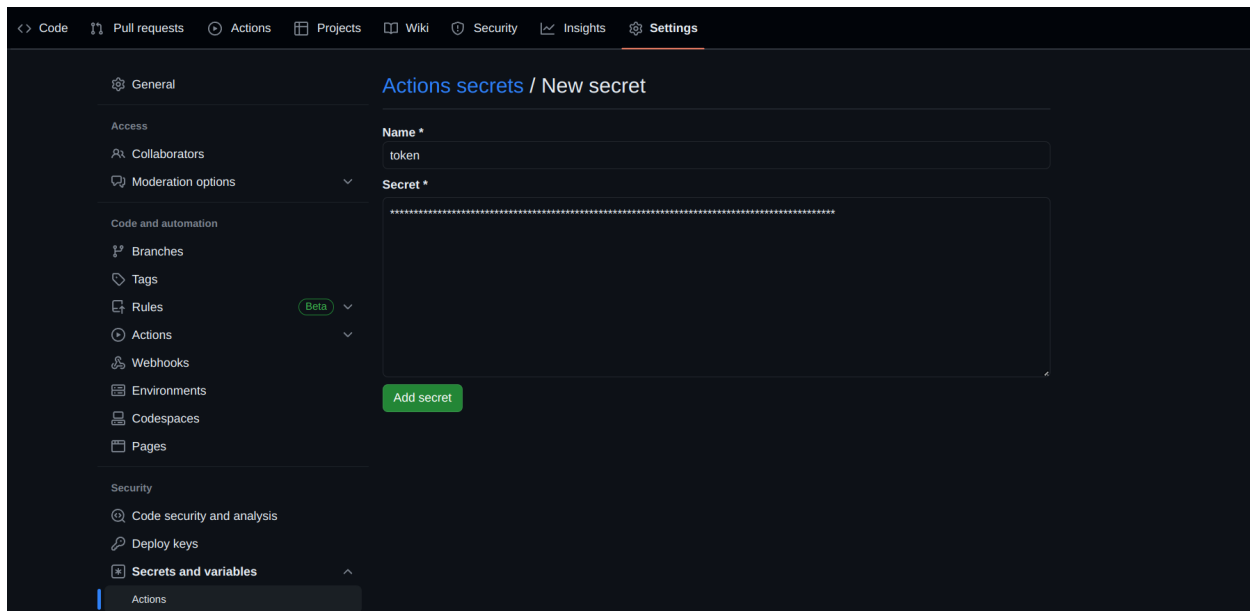
- Click Generate token at bottom of the page , then Copy the token



<input type="checkbox"/> <b>user</b>	Update ALL user data
<input type="checkbox"/> read:user	Read ALL user profile data
<input type="checkbox"/> user:email	Access user email addresses (read-only)
<input type="checkbox"/> user:follow	Follow and unfollow users
<input type="checkbox"/> <b>delete_repo</b>	Delete repositories
<input type="checkbox"/> <b>write:discussion</b>	Read and write team discussions
<input type="checkbox"/> read:discussion	Read team discussions
<input type="checkbox"/> <b>admin:enterprise</b>	Full control of enterprises
<input type="checkbox"/> manage_runners:enterprise	Manage enterprise runners and runner groups
<input type="checkbox"/> manage_billing:enterprise	Read and write enterprise billing data
<input type="checkbox"/> read:enterprise	Read enterprise profile data
<input type="checkbox"/> <b>audit_log</b>	Full control of audit log
<input type="checkbox"/> read:audit_log	Read access of audit log
<input type="checkbox"/> <b>codespace</b>	Full control of codespaces
<input type="checkbox"/> codespace:secrets	Ability to create, read, update, and delete codespace secrets
<input type="checkbox"/> <b>project</b>	Full control of projects
<input type="checkbox"/> read:project	Read access of projects
<input type="checkbox"/> <b>admin:pgp_key</b>	Full control of public user GPG keys
<input type="checkbox"/> write:pgp_key	Write public user GPG keys
<input type="checkbox"/> read:pgp_key	Read public user GPG keys
<input type="checkbox"/> <b>admin:ssh_signing_key</b>	Full control of public user SSH signing keys
<input type="checkbox"/> write:ssh_signing_key	Write public user SSH signing keys
<input type="checkbox"/> read:ssh_signing_key	Read public user SSH signing keys

Generate token Cancel

- Add this token as Github secret in bot repo at <https://github.com/concore-bot/concore-studies/settings/secrets/actions/new> with name token

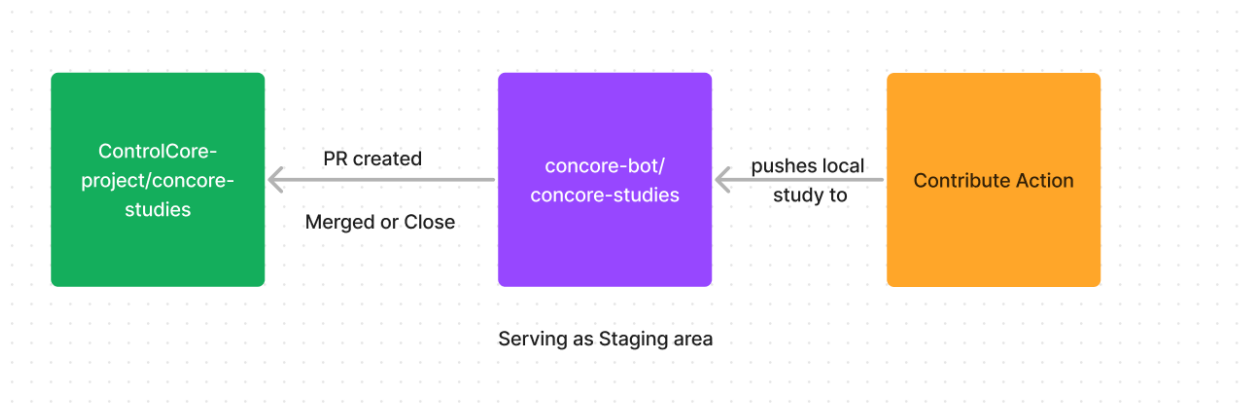


- Click Add secret

Note: This token can have unlimited life span so it is just one-time setup

### 1.6.5 Architecture

- When contribute action is triggered it pushes local study to concore-bot/concore-studies repository



- After pushing study to Github , it creates a pull request for that study to the ControlCore-Project/concore-studies repository
- Until the Pull request is merged or closed this branch serves as staging area

Optional:- If you want to explore why we came up with this approach , please refer [https://docs.google.com/document/d/1DdmP051q0b90QoiQ4RMH-3080gxrGWJuHc9QV\\_A1W0U/edit?usp=sharing](https://docs.google.com/document/d/1DdmP051q0b90QoiQ4RMH-3080gxrGWJuHc9QV_A1W0U/edit?usp=sharing)

## 1.7 Shared Memory communication in Concore

**Shared memory** is a form of inter-process communication (IPC) in computer systems where multiple processes can access and exchange data directly from a common memory space. It allows processes to communicate and share information efficiently without the need for intermediate data storage like files.

### 1.7.1 Advantages of Shared Memory over File Sharing

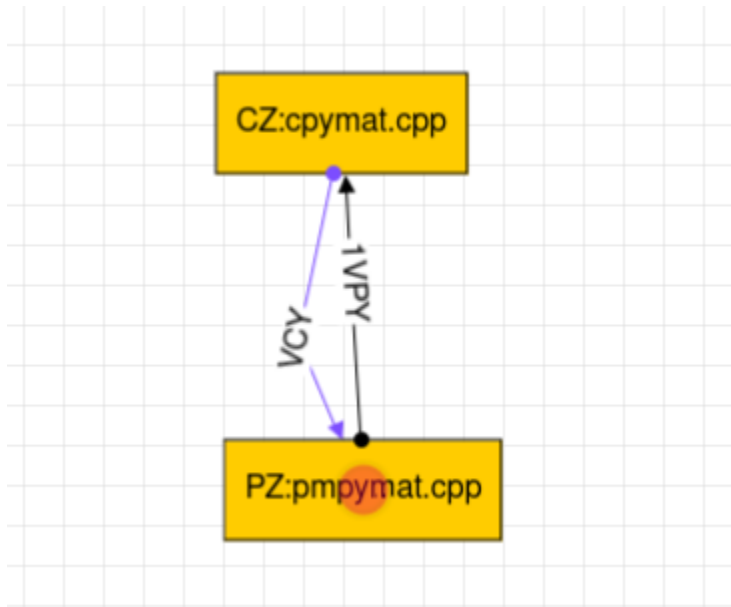
- **Speed:** Shared memory is typically faster than file sharing because data is directly accessible in memory, eliminating the overhead of reading and writing to files. In file sharing, data needs to be written to disk, and the file system overhead can slow down data access.
- **Efficiency:** Since there is no need to read or write data to and from disk, shared memory reduces the CPU and I/O overhead, resulting in more efficient data exchange between processes.
- **Simplicity:** Shared memory is straightforward to implement and use compared to file sharing, where you need to handle file open/close, read/write operations, and ensure proper data integrity.
- **Scalability:** Shared memory can be more scalable in certain situations as it reduces the need for the operating system to manage file I/O, especially when dealing with a large number of processes accessing the same data frequently.
- **Reduced Disk Space Usage:** In file sharing, temporary files might need to be created, and they consume disk space. Shared memory doesn't require creating any extra files, reducing disk space usage.

### 1.7.2 Usage or Activate Shared Memory in Concore

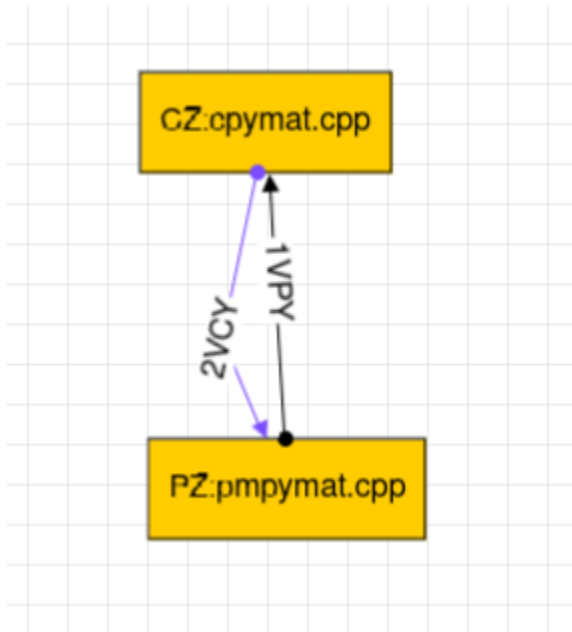
- Need to pass information in the edge in graphml.
- To achieve this, ensure that each edge starts with a positive number greater than 0.
- By following these guidelines, Concore will automatically facilitate data exchange between the relevant components using shared memory.
- Additionally, remember **not to repeat the same positive number** as a prefix for different edge names.

### 1.7.3 Cases

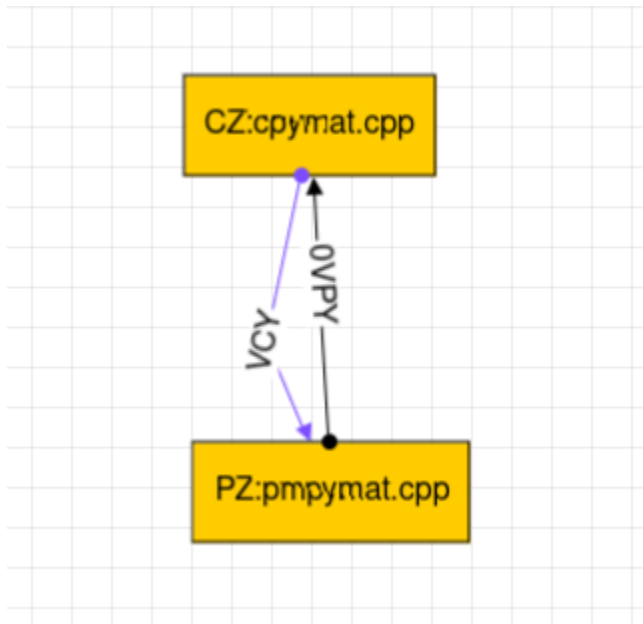
- Case 1:- PZ writes in Shared Memory (key = 1) and reads from file. CZ writes in file and read from same Shared Memory.



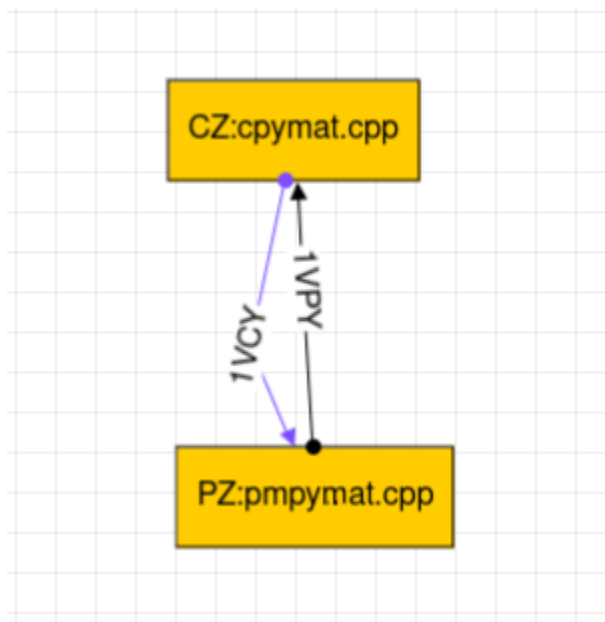
- Case 2:- PZ writes in Shared Memory (key = 1) and reads from Shared Memory (key = 2). CZ writes in Shared Memory (key = 2) and read from Shared Memory (key = 1).



- Case 3:- CZ and PZ use file method for communication.



- Case 4:- Not a valid case.



## 1.8 Concore.hpp

class **Concore**

Class representing the *Concore* implementation in C++.

### Public Functions

inline **Concore**()

Constructor for *Concore* class. Initializes the iport and oport maps by parsing the respective files. It also creates or attaches to the shared memory segment if required.

inline **~Concore**()

Destructor for *Concore* class. Detaches and removes the shared memory segment if shared memory created.

inline key\_t **ExtractNumeric**(const std::string &str)

Extracts the numeric part from a string.

#### Parameters

**str** – The input string.

#### Returns

The numeric part of the string. Returns -1 if the string does not contain a numeric part.

inline void **createSharedMemory**(key\_t key)

Creates a shared memory segment with the given key.

#### Parameters

**key** – The key for the shared memory segment.

inline void **getSharedMemory**(key\_t key)

Retrieves an existing shared memory segment with the given key. Waits until the shared memory segment is created by the writer process.

#### Parameters

**key** – The key for the shared memory segment.

inline map<string, int> **mapParser**(string filename)

Parses a file containing port and number mappings and returns a map of the values.

#### Parameters

**filename** – The name of the file to parse.

#### Returns

A map of port names and their corresponding numbers.

inline bool **unchanged**()

function to compare and determine whether file content has been changed.

#### Returns

true if the content has not changed, false otherwise.

inline vector<double> **parser**(string f)

Parses a string and extracts a vector of double values.

#### Parameters

**f** – The input string to parse.

**Returns**

A vector of double values extracted from the input string.

```
inline vector<double> read(int port, string name, string initstr)
```

deviate the read to either the SM (Shared Memory) or FM (File Method) communication protocol based on iport and oport.

**Parameters**

- **port** – The port number.
- **name** – The name of the file.
- **initstr** – The initial string

**Returns**

```
inline vector<double> read_FM(int port, string name, string initstr)
```

Reads data from a specified port and name using the FM (File Method) communication protocol.

**Parameters**

- **port** – The port number.
- **name** – The name of the file.
- **initstr** – The initial string.

**Returns**

a string of file content

```
inline vector<double> read_SM(int port, string name, string initstr)
```

Reads data from the shared memory segment based on the specified port and name.

**Parameters**

- **port** – The port number.
- **name** – The name of the file.
- **initstr** – The initial string to use if the shared memory is not found.

**Returns**

string of file content

```
inline void write(int port, string name, vector<double> val, int delta = 0)
```

deviate the write to either the SM (Shared Memory) or FM (File Method) communication protocol based on iport and oport.

**Parameters**

- **port** – The port number.
- **name** – The name of the file.
- **val** – The vector of double values to write.
- **delta** – The delta value (default: 0).

```
inline void write(int port, string name, string val, int delta = 0)
```

deviate the write to either the SM (Shared Memory) or FM (File Method) communication protocol based on iport and oport.

**Parameters**

- **port** – The port number.

- **name** – The name of the file.
- **val** – The string to write.
- **delta** – The delta value (default: 0).

inline void **write\_FM**(int port, string name, vector<double> val, int delta = 0)

write method, accepts a vector double and writes it to the file

#### Parameters

- **port** – The port number.
- **name** – The name of the file.
- **val** – The string to write.
- **delta** – The delta value (default: 0).

inline void **write\_FM**(int port, string name, string val, int delta = 0)

write method, accepts a string and writes it to the file

#### Parameters

- **port** – The port number.
- **name** – The name of the file.
- **val** – The string to write.
- **delta** – The delta value (default: 0).

inline void **write\_SM**(int port, string name, vector<double> val, int delta = 0)

Writes a vector of double values to the shared memory segment based on the specified port and name.

#### Parameters

- **port** – The port number.
- **name** – The name of the file.
- **val** – The vector of double values to write.
- **delta** – The delta value (default: 0).

inline void **write\_SM**(int port, string name, string val, int delta = 0)

Writes a string to the shared memory segment based on the specified port and name.

#### Parameters

- **port** – The port number.
- **name** – The name of the file.
- **val** – The string to write.
- **delta** – The delta value (default: 0).

inline vector<double> **initval**(string f)

Initializes the system with the given input values.

#### Parameters

- **f** – The input string containing the values.

#### Returns

A vector of double values representing the initialized system state.



## C

Concore (C++ class), 26  
 Concore::~~Concore (C++ function), 26  
 Concore::Concore (C++ function), 26  
 Concore::createSharedMemory (C++ function), 26  
 Concore::ExtractNumeric (C++ function), 26  
 Concore::getSharedMemory (C++ function), 26  
 Concore::initval (C++ function), 28  
 Concore::mapParser (C++ function), 26  
 Concore::parser (C++ function), 26  
 Concore::read (C++ function), 27  
 Concore::read\_FM (C++ function), 27  
 Concore::read\_SM (C++ function), 27  
 Concore::unchanged (C++ function), 26  
 Concore::write (C++ function), 27  
 Concore::write\_FM (C++ function), 28  
 Concore::write\_SM (C++ function), 28