Scientific Workflows with FireWorks

January 24, 2022

wikitolearn
collaborative textbooks

This book is the result of a collaborative effort of a community of people like you, who believe that knowledge only grows if shared.
We are waiting for you!

Get in touch with the rest of the team by visiting http://join.wikitolearn.org

Contents

Chapter 1

Setup

1.1 Python

For this tutorial it is recommended to use the anaconda python distribution. Download anaconda (Python 3.6 version) from https://www.continuum.io/downloads for Linux x86_64 and follow the installation instructions on the download page. Make sure that python from this installation is in your `$PATH`. In addition the following python packages must be installed:

```
pip install --upgrade pip
pip install pjson
pip install pyaml
pip install future
pip install python-igraph
```

Hint: If later, during usage of the igraph library, an error like this:

```
ImportError: /home/gks/anaconda3/lib/python3.6/site-
   packages/igraph/_igraph.cpython-36m-x86_64-linux-gnu.so
   : undefined symbol:
   _ZNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEC1Ev
```

occurs, then the libgcc package has to be installed/upgraded:

```
conda install libgcc
```

1.2 MongoDB

If your system is Ubuntu and you have administrator permissions you can install MongoDB system-wide with the following command:

```
sudo apt-get install mongodb
```

The server is automatically started after the installation is completed.

For general instructions please read the installation manual https://docs.mongodb.com/manual/administration/install-on-linux/

1.3 Install FireWorks

The following instructions are for installation in your `$HOME` directory. You might like to create another directory for this tutorial and replace `$HOME` with the relevant path.

```
cd $HOME
git clone https://github.com/ikondov/fireworks
cd fireworks
python setup.py develop --user
export PYTHONPATH=$PWD:$PYTHONPATH
export PATH=$HOME/.local/bin:$PATH
```

1.4 Install Tutorial

```
cd $HOME
git clone \href{https://gitlab.com/ikondov/gridka-school-
   fireworks}{https://gitlab.com/ikondov/gridka-school-
   fireworks}
cd gridka-school-fireworks
export PYTHONPATH=$PWD/lib:$PYTHONPATH
export PATH=$PWD/bin:$PATH
```

1.5 Further Packages

In order to visualize the workflows graphically two packages must be installed:

```
sudo apt-get install graphviz evince
```

The packages ImageMagick and Eye of GNOME (eog) are necessary for Exercise 3:

```
sudo apt-get install imagemagick eog
```

If editors like `vi` or `nano` are not preferred, more advanced editors may be installed, e.g.:

```
sudo apt-get install gedit emacs
```

For the web GUI of lpad a web browser must be installed, e.g.:

```
sudo apt-get install firefox
```

Chapter 2

Basic Procedures

2.1 Compose Fireworks and Workflows

Formats and editors

Fireworks and workflows can be defined in three different general-purpose languages: Python, JSON and YAML. There is no domain-specific language in FireWorks and thus no specialized editor. This is why, a normal text editor is sufficient.

All workflow definitions in all exercises will be based on JSON and YAML. Exercise 5 will introduce to writing a custom Firetask for which Python will be used. Again, JSON and YAML will be used to define the workflows using custom Firetasks. For each exercise there are one or more initial examples in the **exercises/demos** folder. We recommend trying these examples before starting solving the problems.

NOTE: Most of the examples here will be presented in YAML (more readable and concise). If you feel more comfortable with editing JSON you can use the converters `yaml2json` and `json2yaml` provided in the **bin** folder or/and the provided JSON versions of the demos and solutions.

Workflow structure

The building blocks of a workflow are Fireworks. A Firework is the minimum possible piece of the workflow executed by the rocket launcher (see below). The major part of the workflow description is the list of Fireworks, `fws`. A Firework has an ID `fw_id`, a name `name`, and a specification `spec` containing a list of Firetasks and Firework specific data. In addition, a dictionary named `links` with dependencies between the Fireworks has to be specified. Further attributes are `metadata` and `name`.

Every Firetask includes the Firetask name `_fw_name` and definitions of its parameters. Firetask is *atomic* i.e. executed at once without further subdivisions. The Firetasks of one Firework are executed strictly one after another in the order of their specification and share the same job working directory and the files in it.

Here is a short example for a workflow demonstrating the usage of the `PythonFunctionTask`:

```
fws:
- fw_id: 1
  name: Grind coffee
  spec:
    _tasks:
    - _fw_name: PythonFunctionTask
      function: auxiliary.print_func
      inputs: [coffee beans]
      outputs: coffee powder
    coffee beans: best selection
- fw_id: 2
  name: Brew coffee
  spec:
    _tasks:
    - _fw_name: PythonFunctionTask
      function: auxiliary.print_func
      inputs: [coffee powder, water]
      outputs: pure coffee
    water: workflowing water
links:
  '1': [2]
metadata: {}
name: Simple coffee workflow
```

Open a text editor, such as `vi`, `nano`, `gedit` or `emacs`, and save the example above as **workflow.yaml**. To convert to JSON you can use the following command:

```
yaml2json < workflow.yaml > workflow.json
```

2.2 Add Fireworks to LaunchPad

The LaunchPad is a database where the workflows are stored during their full life cycle. It is hosted on a resource named FireServer.

NOTE: For simplicity, in this tutorial the FireServer is the same host on which you are logged on.

When used productively the LaunchPad contains many workflows in different states. To distinguish between different workflows, the query commands can specify e.g. the Firework ID from the relevant workflow on the LaunchPad or perform a mongo-like queries. To avoid the need to apply filters to the queries, we will clean up the LaunchPad from previous Fireworks at the beginning of each exercise in this tutorial with this command:

```
lpad reset
```

To add a workflow to the LaunchPad:

```
lpad add workflow.yaml
```

Alternatively in JSON format:

```
lpad add workflow.json
```

2.3 Validate Workflows

Formal verification is done with adding a workflow to the LaunchPad. However, missing links or data dependencies, and circular dependencies are not detected at this stage and the errors appear at run time. To also check for such errors the *-c* or *–check* flags can be used when adding the workflow to the LaunchPad:

```
lpad add -c workflow.json
```

If a workflow has been added without such a check, it can be checked later with:

```
lpad check_wflow -i <firework ID>
```

NOTE: The correctness check is recommended for all exercises in this tutorial.

2.4 Visualize Workflows

Already added workflows can be converted into DOT format and viewed graphically:

```
lpad check_wflow -i <firework ID> [--view_control_flow]
   [--view_data_flow] [-f <DOT_FILE>]
```

After the dot file is produced it can be converted to PDF and the workflow graph can be viewed:

```
dot -Tpdf -o workflow.pdf workflow.dot
evince workflow.pdf
```

2.5 Execution

The workflow engine of FireWorks is called FireWorker. Multiple FireWorkers can be running on different resources where individual Fireworks can be executed by the rocket launcher `rlaunch` which has three modes of operation: *singleshot*, *rapidfire* and *multi*.

NOTE: For simplicity, in this tutorial the FireWorker is the same host as the FireServer.

To only execute one Firework from the LaunchPad which is in *READY* state the following command is used:

```
rlaunch singleshot
```

To run all Fireworks in *READY* state in a sequence:

```
rlaunch rapidfire
```

NOTE: Every Firework changes its state to *READY* after all its parent Fireworks are completed (state *COMPLETED*) and the states of linked child Fireworks are updated as soon as a Firework if completed. This means that any workflow will be run until there are no more Fireworks in *READY* state.

NOTE: In *singleshot* mode `rlaunch` runs the Firework in the directory where it is started. In rapidfire mode `rlaunch` creates separate sub-directories for each Firework at run time.

To suppress verbose information on the screen the *-s* flag can be added:

```
rlaunch -s rapidfire
```

2.6 Monitoring

To query workflows available on the LaunchPad use the command `lpad get_wflows`:

```
lpad get_wflows [-d <more|all>]
```

To query individual Fireworks use the command:

```
lpad get_fws [-i <firework ID> [-d <more|all>]]
```

NOTE: The query from the command line is recommended in this tutorial.

Alternatively the web GUI can be used:

```
lpad webgui
```

NOTE: Make sure that a web browser is configured in your terminal session and an X server is running on your machine and it is configured and tunneled properly in your terminal session.

2.6.1 Remove a workflow from LaunchPad

A selection of workflows can be deleted from LaunchPad using the `lpad delete_wflows` command. For example to delete workflows including Fireworks with given IDs:

```
lpad delete_wflows -i <firework IDs>
```

2.6.2 Failure and restart

If execution of a Firework fails for some reason, its state changes to `FIZZLED`. The failure reason can be found in the launch section of the Firework using the command:

```
lpad get_fws -i <firework ID> -d all
```

If the reason is external and the error is solved then the Firework can be rerun with:

```
lpad rerun_fws -i <firework ID>
```

Chapter 3

Exercises

3.1 Exercise 1: Managing Control Flow

With this exercise we will learn to describe the dependencies in control flow between Fireworks, exploit possible concurrencies and use the standard Firetask `ScriptTask`.

3.1.1 Problem 1.1

Change to directory **exercises/work/1_control_flow**.

Copy the workflow files from the folder **exercises/problems/1_control_flow**. For example, if you decide to use YAML:

```
cp ../../problems/1_control_flow/*.yaml .
```

One by one, check the three sequential workflows **f1_pitstop_seq_wrong_1.yaml**, **f1_pitstop_seq_wrong_2.yaml**, and **f1_pitstop_seq_wrong_3.yaml** for errors, and then find and correct the errors.

Add the workflow to the LaunchPad and query its state.

Execute the workflow with `rlaunch singleshot`, i.e. run one Firework at a time.

After running each single Firework, monitor the output and the states of the Fireworks until the workflow is completed.

3.1.2 Problem 1.2

Repeat the steps of **Problem 1.1** for the parallel version of the workflow:

f1_pitstop_par_wrong_1.json, **f1_pitstop_par_wrong_2.json** and **f1_pitstop_par_wr**

3.2 Exercise 2: Managing Data Flow

The purpose of this exercise is to learn how to pass data between Fireworks and describe data dependencies using the custom Firetask `PythonFunctionTask`:

```
- _fw_name: PythonFunctionTask
  function: any_module.any_function
  inputs:
  - first argument
  - second argument
  outputs:
  - data to forward
```

The keys specified in the `inputs` list must be available in the Firework `spec` at task run time. These are passed as positional arguments to the function. The returned outputs will be stored in the specs of the current Firework and of the child Fireworks under the keys specified in the `outputs` list.

3.2.1 Problem 2.1

Change to folder **exercises/work/2_data_flow**. Copy the provided template **exercises/inputs/2_data_flow/template.[json|yaml]** and complete it so that the script **exercises/problems/2_data_flow/recruiting-script.py** is implemented as a workflow. Check the workflow, add it to LaunchPad and run it in *singleshot* mode watching the changes in the Fireworks with the workflow run.

3.2.2 Problem 2.2

Copy the different solutions **recruiting-[012].[json|yaml]** from the folder **exercises/problems/2_data_flow**, and detect and correct the errors and run the workflow in *rapidfire* mode. Compare the corrected versions to each other and to your solution for Problem 2.1. Compare the results of two instances of the same workflow. Why do they differ?

3.3 Exercise 3: Manage Data in Files and Command line

The purpose of this exercise is to learn how to pass data between Fireworks as files and process these data using the custom `CommandLineTask`. The built-in `ScriptTask` used in **Exercise 1** allows to run a script but provides no methods to move data between Fireworks and no handling of command line options, flags input and output as workflow data.

Given is a set of reusable operations implemented using the `convert` and `montage` commands from the *ImageMagick* package with different sets of command line flags. These are:

- Rotate $+/-90°$, $180°$
- Horizontal mirror
- Vertical mirror

- Montage
- Swirl
- Animate

The Fireworks corresponding to these operations can be found in **exercises/demos/3_files_and_commands**. The provided solutions in **exercises/solutions/3_files_and_commands** are suitable for the input for letter "A" in **exercises/inputs/3_files_and_commands/A**.

3.3.1 Problem 3.1

The input image files are some parts of 2×2 tiled capital letters. Some of the input tiles are rotated or mirrored vertically or horizontally and some missing tiles can be recovered by the same operations using the symmetry. The task is to recover all tiles and put them together to reconstruct the image of the selected letter.

Select a set of input images (**piece-1.png** and **piece-2.png**) for a letter from the folder **exercises/inputs/3_files_and_commands**. Then use the provided Fireworks and compose a workflow to reconstruct the selected letter by adjusting the image processing parameters, inputs and outputs. Verify, add and run the workflow and check the resulting image.

3.3.2 Problem 3.2

Given an image as input, "swirl" the image at the angles 90°, 180°, 270° and 360° producing four new images. Arrange the original and the resulting images in the sequence:

$0° \rightarrow 90° \rightarrow 180° \rightarrow 270° \rightarrow 360° \rightarrow 270° \rightarrow 180° \rightarrow 90° \rightarrow 0°$

and make an animation. As input you can use either the image file **letter.png** from **exercises/inputs/3_files_and_commands** or the reconstructed image from **Problem 3.1**.

3.4 Exercise 4: Extending a Workflow

Workflows in FireWorks can be extended at any time of their life cycle, particularly in states *READY*, *RUNNING* and *COMPLETED*. If the appended Firework or workflow is appended as a detour then the child Fireworks may not be be *RUNNING* or *COMPLETED*.

3.4.1 Problem 4.1

Using the extend the workflow from the solution of **Problem 3.1** (**image_reconstruct.json**) with the workflow from **Problem 3.2** (**image_swirl.json**) to reuse the image produced by the former as input in the latter workflow. For this, first copy the

workflow **image_swirl.json** from **exercises/solutions/3_files_and_commands** to a file **image_swirl_montaged.json**. Then do the following changes:

- Change all Firework IDs to become negative integers. This is required by the `append_wflow` method.
- Add the following Firework to the list of Fireworks (for JSON adapt the code correspondingly):

```
- fw_id: -6
  name: Pass filename
  spec:
    _tasks:
    - _fw_name: PythonFunctionTask
      function: auxiliary.print_func
      inputs: [montaged image]
      outputs: [montaged image]
```

  and:

```
'-6': [-1, -2, -3, -4, -5]
```

  to the `links` dictionary. Why is this step necessary?

- Modify it so that the input `original image` in all Fireworks is passed from a parent Firework:

```
original image: {source: montaged image}
```

- Save the thus prepared sub-workflow file as **image_swirl_montaged.[yaml|json]**.

Now identify the Firework ID of the Firework providing the final image as output:

```
lpad get_fws -n "Put the four pieces together"
```

Alternatively, the workflow can be found by name and the last Firework can be found in the more detailed printout:

```
lpad get_wflows -n "Image reconstruction" -d more
```

Finally, use the command:

```
lpad append_wflow -i <ID> -f image_swirl_montaged.json
```

to extend the workflow with the sub-workflow. A new query shows that the workflow is now marked from *COMPLETED* to *RUNNING*:

```
lpad get_wflows -s "RUNNING"
```

Run the extended workflow.

3.5 Exercise 5: Writing a Firetask

This exercise bases on an extended version of the example from **Exercise 2**. It is implemented as a python script **exercises/problems/5_author_firetask/recruiting-script.py**. It has two new aspects:

1. All parameters are loaded at the beginning from a JSON document **parameters.json**.
2. The steps `candidates_apply()` and `screen_candidates()` are repeated as long as the size of the candidate list is smaller than `number_to_invite`.

3.5.1 Anatomy of Firetasks

All Firetasks are classes derived from the `FiretaskBase` class. Many built-in Firetasks are available in the upstream Fireworks, such as the `ScriptTask`, `PyTask`, `FileTransferTask`. Additional Firetasks can be written for both generic and specific purposes.

The skeleton of a Firetask looks like this:

```
from fireworks.core.firework import FiretaskBase, FWAction
from fireworks.utilities.fw_utilities import
   explicit_serialize

@explicit_serialize
class MyFiretask(FiretaskBase):
    """ My new Firetask """

    _fw_name = 'MyFiretask'
    required_params = ['par1', 'par2']
    optional_params = ['optional par']

    def run_task(self, fw_spec):
        """
        This is the method called upon Firetask execution.
    It has access
        to the spec through the fw_spec parameter. The
   required and the
        optional parameters are accessed through self, i.e
   . they are class
        attributes. Optionally this function returns a
   FWAction object to pass
        data to next Firetasks and Fireworks and/or to
   dynamically modify the
        workflow.
        """

        print(self['par1'], self['par2']) # print the
   values of required parameters
```

```
        print(self.fw_spec['name']) # print the name of
    the Firework

        if self.get('optional par'):
            actions = [
                'update_spec': {
                    'par1': self['par1'],
                    'optional par': self['optional par']
                }
            ]
        else:
            actions = []

        return FWAction(*actions)
```

Here a Firework using this Firetask:

```
- fw_id: 1
  name: Sample firework
  spec:
    _tasks:
    - _fw_name: {{custom_tasks.MyFiretask}}
      par1: data.json
      par2: data.yaml
      optional par: {}
```

NOTE: the python module `custom_tasks` must be in $PYTHONPATH in order to be correctly registered. For this reason keep the file **custom_tasks.py** either in **lib** or in **exercises/work/5_author_firetask** where you start the `lpad` and `rlaunch` commands. In the latter case you should add that directory to $PYTHONPATH with:

```
export PYTHONPATH=`pwd`:$PYTHONPATH
```

3.5.2 Problem 5.1: Data Loader Firetask

Write a `DataLoaderTask` to load the necessary parameters from the JSON document. If necessary the input data structure can be split into more than one file. The resulting workflow, which is given in **exercises/problems/5_author_firetask/dataloader.jso** must be successfully running with no further modifications.

Hint: You can use the `load()` method from the `json` package to load JSON documents as list or dictionaries and then return a `FWAction` object with `update_spec` and the structure (see above example).

3.5.3 Problem 5.2: Conditional Repeater Firetask

Write a `RepeatIfLengthLesser` Firetask that implements the while loop in the script. The Firetask should integrate into the workflow available in **dataloader+repeater.json** without further adaptations.

Hint: You can use the `load_object` function from `fireworks.utilities.fw_serializers` to construct a `Firework` object and the `detours` keyword argument of `FWAction` to insert the Firework dynamically:

```
firework = Firework(
    tasks=[load_object(task) for task in fw_spec['_tasks
   ']],
    spec=fw_spec,
    name='repeat '+self['measure']
)
return FWAction(detours=firework)
```

Chapter 4

Text and image sources, contributors, and licenses

4.1 Text

- **Course:Scientific Workflows with FireWorks/Setup/Python** *Source:* https://en.wikitolearn.org/Course%3AScientific_Workflows_with_FireWorks/Setup/Python?oldid=9978 *Contributors:* Mapelli Dario and Elnaz
- **Course:Scientific Workflows with FireWorks/Setup/MongoDB** *Source:* https://en.wikitolearn.org/Course%3AScientific_Workflows_with_FireWorks/Setup/MongoDB?oldid=9976 *Contributors:* Mapelli Dario and Elnaz
- **Course:Scientific Workflows with FireWorks/Setup/Install FireWorks** *Source:* https://en.wikitolearn.org/Course%3AScientific_Workflows_with_FireWorks/Setup/Install_FireWorks?oldid=9972 *Contributors:* Elnaz
- **Course:Scientific Workflows with FireWorks/Setup/Install Tutorial** *Source:* https://en.wikitolearn.org/Course%3AScientific_Workflows_with_FireWorks/Setup/Install_Tutorial?oldid=9974 *Contributors:* Elnaz
- **Course:Scientific Workflows with FireWorks/Setup/Further Packages** *Source:* https://en.wikitolearn.org/Course%3AScientific_Workflows_with_FireWorks/Setup/Further_Packages?oldid=9968 *Contributors:* Elnaz
- **Course:Scientific Workflows with FireWorks/Basic Procedures/Compose Fireworks and Workflows** *Source:* https://en.wikitolearn.org/Course%3AScientific_Workflows_with_FireWorks/Basic_Procedures/Compose_Fireworks_and_Workflows?oldid=9387 *Contributors:* Elnaz
- **Course:Scientific Workflows with FireWorks/Basic Procedures/Add Fireworks to LaunchPad** *Source:* https://en.wikitolearn.org/Course%3AScientific_Workflows_with_FireWorks/Basic_Procedures/Add_Fireworks_to_LaunchPad?oldid=9388 *Contributors:* Elnaz
- **Course:Scientific Workflows with FireWorks/Basic Procedures/Validate Workflows** *Source:* https://en.wikitolearn.org/Course%3AScientific_Workflows_with_FireWorks/Basic_Procedures/Validate_Workflows?oldid=9389 *Contributors:* Elnaz
- **Course:Scientific Workflows with FireWorks/Basic Procedures/Visualize Workflows** *Source:* https://en.wikitolearn.org/Course%3AScientific_Workflows_with_FireWorks/Basic_Procedures/Visualize_Workflows?oldid=9390 *Contributors:* Elnaz
- **Course:Scientific Workflows with FireWorks/Basic Procedures/Execution** *Source:* https://en.wikitolearn.org/Course%3AScientific_Workflows_with_FireWorks/Basic_Procedures/Execution?oldid=9391 *Contributors:* Elnaz
- **Course:Scientific Workflows with FireWorks/Basic Procedures/Monitoring** *Source:* https://en.wikitolearn.org/Course%3AScientific_Workflows_with_FireWorks/Basic_Procedures/Monitoring?oldid=9392 *Contributors:* Elnaz

- **Course:Scientific Workflows with FireWorks/Exercises/Exercise 1: Managing Control Flow** *Source:* https://en.wikitolearn.org/Course%3AScientific_Workflows_with_FireWorks/Exercises/Exercise_1%3A_Managing_Control_Flow?oldid=9393 *Contributors:* Elnaz
- **Course:Scientific Workflows with FireWorks/Exercises/Exercise 2: Managing Data Flow** *Source:* https://en.wikitolearn.org/Course%3AScientific_Workflows_with_FireWorks/Exercises/Exercise_2%3A_Managing_Data_Flow?oldid=9520 *Contributors:* Elnaz
- **Course:Scientific Workflows with FireWorks/Exercises/Exercise 3: Manage Data in Files and Command line** *Source:* https://en.wikitolearn.org/Course%3AScientific_Workflows_with_FireWorks/Exercises/Exercise_3%3A_Manage_Data_in_Files_and_Command_line?oldid=9395 *Contributors:* Elnaz
- **Course:Scientific Workflows with FireWorks/Exercises/Exercise 4: Extending a Workflow** *Source:* https://en.wikitolearn.org/Course%3AScientific_Workflows_with_FireWorks/Exercises/Exercise_4%3A_Extending_a_Workflow?oldid=9400 *Contributors:* Elnaz
- **Course:Scientific Workflows with FireWorks/Exercises/Exercise 5: Writing a Firetask** *Source:* https://en.wikitolearn.org/Course%3AScientific_Workflows_with_FireWorks/Exercises/Exercise_5%3A_Writing_a_Firetask?oldid=9789 *Contributors:* Riccardo Iaconelli, Mapelli Dario and Elnaz

4.2 Images

4.3 Content license

- [Project:Copyright Creative Commons Attribution Share Alike 3.0 & GNU FDL]
- Creative Commons Attribution-Share Alike 3.0