

---

# **CWLKernel**

*Release 0.0.4*

**Jul 29, 2020**



---

# Contents

---

<b>1</b>	<b>Installation</b>	<b>3</b>
<b>2</b>	<b>Contents</b>	<b>5</b>
2.1	For Users . . . . .	5
2.2	For Developers . . . . .	5
2.2.1	Custom Magic Commands Extensions . . . . .	5
2.2.2	Basic Example: Custom Magic Command . . . . .	5
2.2.3	Advanced Example: Custom Magic Command . . . . .	6
2.2.4	Code Details . . . . .	10
<b>3</b>	<b>Indices and tables</b>	<b>17</b>
	<b>Python Module Index</b>	<b>19</b>
	<b>Index</b>	<b>21</b>



---

CWLKernel is an Kernel for Jupyter Notebook to enable users to execute **CWL**.

---

```
cwlVersion: v1.1
class: CommandLineTool
baseCommand: echo
inputs:
  message:
    type: string
    inputBinding:
      position: 1
outputs:
  example_output:
    type: stdout
```

```
{
  "basename": "f811c63eed22dec4eec6f02280820eabf16fa779",
  "checksum": "sha1$47a013e660d408619d894b20806b1d5086aab03b",
  "class": "File",
  "http://commonwl.org/cwltool#generation": 0,
  "id": "example_output",
  "location": "file:///private/tmp/CWLKERNEL_DATA/runtime_data/
↪f811c63eed22dec4eec6f02280820eabf16fa779",
  "nameext": "",
  "nameroot": "f811c63eed22dec4eec6f02280820eabf16fa779",
  "size": 13
}
```



# CHAPTER 1

---

## Installation

---

To install the kernel, download it from [github](#) and run the python setup.py install.





## 2.1 For Users

Check the examples in [github](#).

## 2.2 For Developers

### 2.2.1 Custom Magic Commands Extensions

The kernel is designed to be extendable. The methodology for creating extensions is inspired by [IPython's Magic Commands](#). Developers can create new custom magic commands.

Jupyter Notebook's cells that contain magic commands cannot contain CWL code. The magic command format is the following: `% [command] [argument_string]`. The argument string is multiline and contains all the content after the magic command until the end of the cell or the next magic command. For example, at the following snippet, the first time the `argument_string` will contain as a single string, the string from `arg1` until the bar and at the second example, the `argument_string` will be an empty string.

Kernel's configuration is based on the system's environment variables and it is managed by the `cwlkernel.CWLExecuteConfigurator.CWLExecuteConfigurator`. The `CWLKERNEL_MAGIC_COMMANDS_DIRECTORY` variable holds the path that the kernel will search for python scripts to execute them. Check the [Basic Example: Custom Magic Command](#).

### 2.2.2 Basic Example: Custom Magic Command

In the presented example we want to create a magic command which prints the message hello world. Firstly, the directory of the custom magic commands should be configured.

```
mkdir -p ~/.cwlkernel/startup/
cd ~/.cwlkernel/startup/
export CWLKERNEL_MAGIC_COMMANDS_DIRECTORY=$(pwd)
echo $CWLKERNEL_MAGIC_COMMANDS_DIRECTORY
```

Inside the directory, we create the following file *hello.py*.

```
1 from cwlkernel.CWLKernel import CWLKernel
2
3 @CWLKernel.register_magic()
4 def hello(kernel: CWLKernel, argument_string: str):
5     kernel.send_response(
6         kernel.iopub_socket,
7         'stream',
8         {'name': 'stdout', 'text': 'hello world'})
9     )
```

The decorator is required to register the magic command to the kernel. Every magic command should have that signature. In case that the magic command does not accept any arguments, like in this case, the argument string will be just an empty string. Now, we can open a jupyter notebook and run the following command.

If we want to build command with more arguments with complicated structure, the usage of `argparse` is suggested. For example, the aforementioned example could be changed to:

```
1 from cwlkernel.CWLKernel import CWLKernel
2 import argparse
3
4 @CWLKernel.register_magic()
5 def hello(kernel: CWLKernel, argument_string: str):
6     parser = argparse.ArgumentParser()
7     parser.add_argument(
8         'messages', nargs='*', type=str, default=['hello world'])
9     )
10    args = parser.parse_args(argument_string.split())
11    for message in args.messages:
12        kernel.send_response(
13            kernel.iopub_socket,
14            'stream',
15            {'name': 'stdout', 'text': message + '\n'})
16    )
```

### 2.2.3 Advanced Example: Custom Magic Command

For custom magic commands with state and complex logic the object-oriented strategy is suggested. To do that, you have to create a class to encapsulate the logic inside. The state has to be defined as a class attribute. The method's functionality, which is mapped to magic command, should be defined as a static method and registered as a magic command with the decorator.

In the following tutorial, we present how to use some of Jupyter's features to build interactive commands. Jupyter Notebook has a pub-sub communication channel to communicate with the kernel. There are multiple types of messages that the kernel can send to the notebook. For more information check [Jupyter's documentation for messaging](#).

In the presented tutorial we will use the *display\_data* & *update\_display\_data* message types to illustrate how we can build interactive magic commands. Let's suppose that we want to build magic commands to create & visualise a graph. Also, we want instead of printing the image multiple times to update the initial one.

- `add_node`: add a new node in the graph

- `add_edge`: add a new edge in the graph
- `bind_view`: initialise the graph and bind the image display

To do that we will use the `networkx` library and the `matplotlib`.

We assume that you have already set up a directory to add custom magic commands, as it is described in the *Basic Example: Custom Magic Command*. In that directory, let's create a file `interactive.py`. In order to implement the requirements, we will create a class named `BindGraph`. The class has a state, the graph, that we want to visualise, the attribute named `G`, and a `data_id` which is required for updating the data to the notebook.

```
class BindGraph:
    G = None
    data_id = '1234'
```

---

### Tip: Technical Recommendation

The kernel is not aware of the sequence in which the jupyter notebook's cells are, but the kernel receives them in the order that the user executes them. For example, if in the jupyter notebook we have in the following cells `% command1` and `% command2`, the user, during his development, may execute them in different/wrong order. For use cases in which magic commands have common states, the usage of [builder pattern](#) is suggested to be considered.

---

Firstly, we want to create a function for displaying new images or updating existing ones. To do that, we will use the API provided by jupyter notebook. In that function, we send from the kernel to the notebook a message including a `display_id`. This id is needed to be able to update the image when we request it. So, we will define the following staticmethod:

```
@staticmethod
def display_image(kernel: CWLKernel, image_html_tag: str, update: bool = False):
    if update:
        message_type = 'update_display_data'
    else:
        message_type = 'display_data'
    kernel.send_response(
        kernel.iopub_socket,
        message_type,
        {
            'data': {
                'text/html': image_html_tag,
                'text/plain': f"{image_html_tag}"
            },
            'metadata': {},
            'transient': {
                'display_id': BindGraph.data_id
            }
        }
    )
```

Then we want to define the `bind_view` magic command. That command has to initialise an empty graph and visualise the empty image.

```
@staticmethod
@CWLKernel.register_magic()
def bind_view(kernel: CWLKernel, arg: str):
    BindGraph.G = nx.Graph()
    image = BindGraph.get_image()
    BindGraph.display_image(kernel, image)
```

The `get_image` is a staticmethod that we defined to convert the graph to an HTML image tag.

```
@staticmethod
def get_image():
    nx.draw(BindGraph.G, with_labels=True)
    image_stream = BytesIO()
    plt.savefig(image_stream)
    image_base64 = base64.b64encode(image_stream.getvalue()).decode()
    plt.clf()
    mime = 'image/png'
    image = f"<img src='data:{mime}; base64, {image_base64}' alt='Graph'>"
    return image
```

The methods for adding node & edge are very similar. For both of the cases, firstly we update the graph based on the user's argument and then we generate the new image and we send a message for update. Finally, we also send a message under the cell that the user requested to execute the command to inform him.

```
@staticmethod
@CWLKernel.register_magic()
def add_node(kernel: CWLKernel, arg: str):
    BindGraph.G.add_node(arg)
    image = BindGraph.get_image()
    BindGraph.display_image(kernel, image, update=True)
    kernel.send_text_to_stdout('Done!\n')

@staticmethod
@CWLKernel.register_magic()
def add_edge(kernel: CWLKernel, arg: str):
    edges = arg.split()
    BindGraph.G.add_edge(*edges)
    image = BindGraph.get_image()
    BindGraph.display_image(kernel, image, update=True)
    kernel.send_text_to_stdout('Done!\n')
```

Finally, the full code will look like that:

```
import base64
import networkx as nx
import matplotlib.pyplot as plt
from cwlkernel.CWLKernel import CWLKernel
from io import BytesIO

class BindGraph:
    G = None
    data_id = '1234'

    @staticmethod
    def display_image(kernel: CWLKernel, image_html_tag: str, update: bool = False):
        if update:
            message_type = 'update_display_data'
        else:
            message_type = 'display_data'
        kernel.send_response(
            kernel.iopub_socket,
            message_type,
            {
                'data': {
```

(continues on next page)

(continued from previous page)

```

        "text/html": image_html_tag,
        "text/plain": f"{image_html_tag}"
    },
    'metadata': {},
    'transient': {
        'display_id': BindGraph.data_id
    }
}
)

@staticmethod
@CWLKernel.register_magic()
def add_node(kernel: CWLKernel, arg: str):
    BindGraph.G.add_node(arg)
    image = BindGraph.get_image()
    BindGraph.display_image(kernel, image, update=True)
    kernel.send_text_to_stdout('Done!\n')

@staticmethod
@CWLKernel.register_magic()
def add_edge(kernel: CWLKernel, arg: str):
    edges = arg.split()
    BindGraph.G.add_edge(*edges)
    image = BindGraph.get_image()
    BindGraph.display_image(kernel, image, update=True)
    kernel.send_text_to_stdout('Done!\n')

@staticmethod
def get_image():
    nx.draw(BindGraph.G, with_labels=True)
    image_stream = BytesIO()
    plt.savefig(image_stream)
    image_base64 = base64.b64encode(image_stream.getvalue()).decode()
    plt.clf()
    mime = 'image/png'
    image = f''
    return image

@staticmethod
@CWLKernel.register_magic()
def bind_view(kernel: CWLKernel, arg: str):
    BindGraph.G = nx.Graph()
    image = BindGraph.get_image()
    BindGraph.display_image(kernel, image)

@staticmethod
def display_image(kernel: CWLKernel, image_html_tag: str, update: bool = False):
    if update:
        message_type = 'update_display_data'
    else:
        message_type = 'display_data'
    kernel.send_response(
        kernel.iopub_socket,
        message_type,
        {
            'data': {
                "text/html": image_html_tag,

```

(continues on next page)

```

        "text/plain": f"{image_html_tag}"
    },
    'metadata': {},
    'transient': {
        'display_id': BindGraph.data_id
    }
}
)

```

## 2.2.4 Code Details

### Source Code Modules

#### cwlkernel Module

#### CoreExecutor

```

class cwlkernel.CoreExecutor.CoreExecutor (file_manager:          cwlkernel.IOManager.IOFileManager,
                                           provenance_directory: Optional[pathlib.Path])

```

**execute** (*provenance=False*) → Tuple[uuid.UUID, Dict[KT, VT], Optional[Exception], Optional[cwltool.provenance.ResearchObject]]

**Parameters** **provenance** – Execute with provenance enabled/disabled.

**Returns** Run ID, dict with new files, exception if there is any.

**set\_data** (*data: List[str]*) → List[str]

**set\_workflow\_path** (*workflow\_str: str*) → str

**Parameters** **workflow\_str** – the cwl

**Returns** the path where we executor stored the workflow

```

classmethod validate_input_files (yaml_input: Dict[KT, VT], cwd: pathlib.Path) → NoReturn

```

```

class cwlkernel.CoreExecutor.JupyterFactory (root_directory: str, executor: Optional[cwltool.executors.JobExecutor]
                                             = None, loading_context: Optional[cwltool.context.LoadingContext]
                                             = None, runtime_context: Optional[cwltool.context.RuntimeContext]
                                             = None)

```

```

class cwlkernel.CoreExecutor.ProvenanceFactory (workflow_uri_path: str, root_directory: str,
                                                stove_provenance_directory: str,
                                                executor: Optional[cwltool.executors.JobExecutor]
                                                = None, loading_context: Optional[cwltool.context.LoadingContext]
                                                = None, runtime_context: Optional[cwltool.context.RuntimeContext]
                                                = None)

```

## CWLExecuteConfigurator

```
class cwlkernel.CWLExecuteConfigurator.CWLExecuteConfigurator
```

```
    properties = {'CWLKERNEL_BOOT_DIRECTORY': ('/tmp/CWLKERNEL_DATA', <function CWLExecuteConfigurator.boot_directory>)
```

## kernel\_magics

```
class cwlkernel.kernel_magics.ExecutionMagics
```

```
    static execute (kernel: cwlkernel.CWLKernel.CWLKernel, execute_argument_string: str)
```

```
        Execute registered tool by id. % execute [tool-id] [yaml input ...]
```

```
        @param kernel: the kernel instance @param execute_argument_string: a multiple line string contains in the first line the tool id and in the next lines the input parameters in yaml syntax @return: None
```

```
    static execute_with_provenance (kernel: cwlkernel.CWLKernel.CWLKernel, execute_argument_string: str)
```

```
    static suggest_execution_id (query_token: str, *args, **kwargs) → List[str]
```

```
class cwlkernel.kernel_magics.MagicSnippetBuilder
```

```
    static snippet (kernel: cwlkernel.CWLKernel.CWLKernel, command: str)
```

```
        Submit a cwl workflow incrementally. Usage: % snippet add [...] % snippet add [...] % snippet build
```

```
        @param kernel: @param command: @return:
```

```
class cwlkernel.kernel_magics.ManualWorkflowComposer
```

```
    static new_workflow (kernel: cwlkernel.CWLKernel.CWLKernel, workflow_id: str)
```

```
    static new_workflow_add_input (kernel: cwlkernel.CWLKernel.CWLKernel, args: str)
```

```
    static new_workflow_add_output_source (kernel: cwlkernel.CWLKernel.CWLKernel, args: str)
```

```
    static new_workflow_add_step (kernel: cwlkernel.CWLKernel.CWLKernel, ids: str)
```

```
    static new_workflow_add_step_in (kernel: cwlkernel.CWLKernel.CWLKernel, args: str)
```

```
    static new_workflow_build (kernel: cwlkernel.CWLKernel.CWLKernel, *args)
```

```
class cwlkernel.kernel_magics.Scatter
```

```
    classmethod parse_args (args_line) → Tuple[str, str]
```

```
    parser = ArgumentParser(prog='sphinx-build', usage=None, description=None, formatter_class=ArgumentParser
```

```
    static scatter (kernel: cwlkernel.CWLKernel.CWLKernel, args_line: str)
```

```
    scatter_template = {'class': 'Workflow', 'cwlVersion': None, 'inputs': None, 'outputs': None}
```

```
cwlkernel.kernel_magics.compile_executed_steps_as_workflow (kernel: cwlkernel.CWLKernel.CWLKernel, args: str)
```

```
    Compose a workflow from executed workflows.
```

```
    @param kernel: @param args: @return:
```

```
cwlkernel.kernel_magics.data (kernel: cwlkernel.CWLKernel.CWLKernel, *args)
    Display all the data which are registered in the kernel session.

cwlkernel.kernel_magics.display_data (kernel: cwlkernel.CWLKernel.CWLKernel, data_name:
    str) → None
    Display the data generated by workflow. Usage % displayData [data id]
    @param kernel: the kernel instance @param data_name: the data id @return None

cwlkernel.kernel_magics.display_data_csv (kernel: cwlkernel.CWLKernel.CWLKernel,
    data_name: str)

cwlkernel.kernel_magics.display_data_image (kernel: cwlkernel.CWLKernel.CWLKernel,
    data_name: str)

cwlkernel.kernel_magics.edit (kernel: cwlkernel.CWLKernel.CWLKernel, args: str) → Op-
    tional[Dict[KT, VT]]

cwlkernel.kernel_magics.github_import (kernel: cwlkernel.CWLKernel.CWLKernel, url: str)

cwlkernel.kernel_magics.logs (kernel: cwlkernel.CWLKernel.CWLKernel, limit=None)

cwlkernel.kernel_magics.magics (kernel: cwlkernel.CWLKernel.CWLKernel, arg: str)

cwlkernel.kernel_magics.sample_csv (kernel: cwlkernel.CWLKernel.CWLKernel, args: str)

cwlkernel.kernel_magics.system (kernel: cwlkernel.CWLKernel.CWLKernel, commands: str)
    Execute bash commands in the Runtime Directory of the session.
    @param kernel: @param commands: @return:

cwlkernel.kernel_magics.view_tool (kernel: cwlkernel.CWLKernel.CWLKernel, workflow_id:
    str)

cwlkernel.kernel_magics.visualize_graph (kernel: cwlkernel.CWLKernel.CWLKernel,
    tool_id: str)
    Visualize a Workflow
```

## IOManager

```
class cwlkernel.IOManager.IOFileManager (root_directory: str)

    append_files (files_to_copy: List[str], relative_path: str = '.', metadata: Optional[Dict[KT, VT]] =
        None) → List[str]

    clear ()

    files_counter

    get_files () → List[str]

    get_files_registry () → Dict[KT, VT]

    get_files_uri () → urllib.parse.ParseResult

    read (relative_path: str) → bytes

    remove (path: str)

    write (relative_path: str, binary_data: bytes, metadata=None) → str

class cwlkernel.IOManager.ResultsManager (root_directory: str)
```



**get\_last\_result\_by\_id** (*result\_id: str*) → Optional[str]

The results manager may have multiple results with the same id, from multiple executions. That function will return the path of the last result @param result\_id id to the Results manager. If the result\_id has the format of path then the last goes to the id and the previous one to the produced by [\_produced\_by]/[result\_id] @return: the path of last result with the requested id or None

## CWLLogger

**class** cwlkernel.CWLLogger.CWLLogger (*root\_directory*)

**classmethod** **collect\_infrastructure\_metrics** () → NamedTuple

**classmethod** **get\_hostname** () → str

**classmethod** **get\_running\_kernels** () → List[int]

**Returns** A list with the process ids of running kernels

**load** (*limit=None*) → Iterator[Dict[KT, VT]]

**save** ()

**to\_dict** ()

## CWLBuilder

**class** cwlkernel.CWLBuilder.CWLBuilder

**build** () → cwlkernel.cwlrepository.CWLComponent.WorkflowComponent

**class** cwlkernel.CWLBuilder.CWLSnippetBuilder

**append** (*code: str, indent: int = 0*) → None

**build** () → cwlkernel.cwlrepository.CWLComponent.WorkflowComponent

**clear** ()

**get\_current\_code** () → str

## CWLKernel

**class** cwlkernel.CWLKernel.CWLKernel (\*\**kwargs*)

Jupyter Notebook kernel for CWL.

**banner** = 'Common Workflow Language'

**do\_complete** (*code: str, cursor\_pos: int*)

Override in subclasses to find completions.

**do\_execute** (*code: str, silent=False, store\_history: bool = True, user\_expressions=None, allow\_stdin: bool = False*) → Dict[KT, VT]

Execute user code. Must be overridden by subclasses.

**get\_past\_results** () → List[str]

**get\_pid** () → Tuple[int, int]

**Returns** The process id and his parents id.

**history**

Returns a list of executed cells in the current session. The first item has the value “magic”/”register” and the second the code

**implementation** = 'CWLKernel'

**implementation\_version** = '0.0.4'

**language\_info** = {'file\_extension': '.cwl', 'mimetype': 'text/x-cwl', 'name': 'yaml'

**language\_version** = '1.1'

**class register\_magic** (*magics\_name: Optional[str] = None*)

Registers magic commands. That method should be used as a decorator to register custom magic commands.

**class register\_magics\_suggester** (*magic\_command\_name: str*)

Decorator for registering functions for suggesting commands line arguments

**results\_manager****runtime\_directory**

**send\_error\_response** (*text*) → None

Sends a response to the jupyter notebook’s stderr. @param text: The message to display @return: None

**send\_json\_response** (*json\_data: Union[Dict[KT, VT], List[T]]*) → None

Display a Dict or a List object as a JSON. The object must be json dumpable to use that function. @param json\_data: Data to print in Jupyter Notebook @return: None

**send\_text\_to\_stdout** (*text: str*)

**workflow\_repository**

## CWLLoggerStorageManager

**class** cwlkernel.CWLLoggerStorageManager.CWLLoggerStorageManager (*root\_directory*)

**get\_storage\_path** () → str

**load** (*limit=None*) → Iterator[Dict[KT, VT]]

**save** (*logs*) → str

## cwlkernel repository Module

### CWLComponent

**class** cwlkernel.cwlrepository.CWLComponent.CWLTool (*workflow\_id: str, command\_line\_tool: Dict[KT, VT]*)

**command\_line\_tool**

**compose\_requirements** () → Dict[KT, VT]

**inputs****outputs**

```

to_dict () → Dict[KT, VT]
to_yaml () → str
class cwlkernel.cwlrepository.CWLComponent.CWLWorkflow (workflow_id: str, workflow: Optional[Dict[KT, VT]] = None)

    add (component: cwlkernel.cwlrepository.CWLComponent.WorkflowComponent, step_name: str, run_reference: Optional[str] = None) → None
    add_input (workflow_input: Dict[KT, VT], step_id: str, in_step_id: str)
    add_output_source (output_ref: str, type_of: str)
    add_step_in_out (connect: Union[str, dict], step_in_name: str, step_in: str, step_out: Optional[str] = None, step_out_id: Optional[str] = None)
    compose_requirements () → Dict[KT, VT]
    inputs
    outputs
    remove (component: cwlkernel.cwlrepository.CWLComponent.WorkflowComponent) → None
    steps
    to_dict () → Dict[KT, VT]
    to_yaml () → str
    validate ()

class cwlkernel.cwlrepository.CWLComponent.WorkflowComponent (workflow_id: str, component: Optional[Dict[KT, VT]])

    compose_requirements () → Dict[KT, VT]
    get_input (input_id: str) → Optional[Dict[KT, VT]]
    get_output (output_id: str) → Optional[Dict[KT, VT]]
    id
    inputs
    outputs
    to_dict () → Dict[KT, VT]
    to_yaml () → str

class cwlkernel.cwlrepository.CWLComponent.WorkflowComponentFactory

    get_workflow_component (yaml_string: str) → cwlkernel.cwlrepository.CWLComponent.WorkflowComponent

```

## CWLRepository

```

exception cwlkernel.cwlrepository.cwlrepository.MissingIdError
class cwlkernel.cwlrepository.cwlrepository.WorkflowRepository (directory: pathlib.Path)

```

```
classmethod get_instance () → __SingletonWorkflowRepository__
```

## CHAPTER 3

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### C

`cwlkernel.CoreExecutor`, 10  
`cwlkernel.CWLBuilder`, 13  
`cwlkernel.CWLExecuteConfigurator`, 11  
`cwlkernel.CWLKernel`, 13  
`cwlkernel.CWLLogger`, 13  
`cwlkernel.CWLLoggerStorageManager`, 14  
`cwlkernel.cwlrepository.CWLComponent`,  
14  
`cwlkernel.cwlrepository.cwlrepository`,  
15  
`cwlkernel.IOManager`, 12  
`cwlkernel.kernel_magics`, 11





- 
- A**
- add() (*cwlkernel.cwlrepository.CWLComponent.CWLWorkflow* method), 15
  - add\_input() (*cwlkernel.cwlrepository.CWLComponent.CWLWorkflow* method), 15
  - add\_output\_source() (*cwlkernel.cwlrepository.CWLComponent.CWLWorkflow* method), 15
  - add\_step\_in\_out() (*cwlkernel.cwlrepository.CWLComponent.CWLWorkflow* method), 15
  - append() (*cwlkernel.CWLBuilder.CWLSnippetBuilder* method), 13
  - append\_files() (*cwlkernel.IOManager.IOFileManager* method), 12
- B**
- banner (*cwlkernel.CWLKernel.CWLKernel* attribute), 13
  - build() (*cwlkernel.CWLBuilder.CWLBuilder* method), 13
  - build() (*cwlkernel.CWLBuilder.CWLSnippetBuilder* method), 13
- C**
- clear() (*cwlkernel.CWLBuilder.CWLSnippetBuilder* method), 13
  - clear() (*cwlkernel.IOManager.IOFileManager* method), 12
  - collect\_infrastructure\_metrics() (*cwlkernel.CWLLogger.CWLLogger* class method), 13
  - command\_line\_tool (*cwlkernel.cwlrepository.CWLComponent.CWLTool* attribute), 14
  - compile\_executed\_steps\_as\_workflow() (in module *cwlkernel.kernel\_magics*), 11
  - compose\_requirements() (*cwlkernel.cwlrepository.CWLComponent.CWLTool* method), 14
  - compose\_requirements() (*cwlkernel.cwlrepository.CWLComponent.CWLWorkflow* method), 15
  - compose\_requirements() (*cwlkernel.cwlrepository.CWLComponent.WorkflowComponent* method), 15
  - CoreExecutor (class in *cwlkernel.CoreExecutor*), 10
  - CWLBuilder (class in *cwlkernel.CWLBuilder*), 13
  - CWLExecuteConfigurator (class in *cwlkernel.CWLExecuteConfigurator*), 11
  - CWLKernel (class in *cwlkernel.CWLKernel*), 13
  - cwlkernel.CoreExecutor* (module), 10
  - cwlkernel.CWLBuilder* (module), 13
  - cwlkernel.CWLExecuteConfigurator* (module), 11
  - cwlkernel.CWLKernel* (module), 13
  - cwlkernel.CWLLogger* (module), 13
  - cwlkernel.CWLLoggerStorageManager* (module), 14
  - cwlkernel.cwlrepository.CWLComponent* (module), 14
  - cwlkernel.cwlrepository.cwlrepository* (module), 15
  - cwlkernel.IOManager* (module), 12
  - cwlkernel.kernel\_magics* (module), 11
  - CWLKernel.register\_magic* (class in *cwlkernel.CWLKernel*), 14
  - CWLKernel.register\_magics\_suggester* (class in *cwlkernel.CWLKernel*), 14
  - CWLLogger* (class in *cwlkernel.CWLLogger*), 13
  - CWLLoggerStorageManager* (class in *cwlkernel.CWLLoggerStorageManager*), 14
  - CWLSnippetBuilder* (class in *cwlkernel.CWLBuilder*), 13
  - CWLTool* (class in *cwlkernel.cwlrepository.CWLComponent*), 14
  - CWLWorkflow* (class in *cwlkernel*
-

- nel.cwlrepository.CWLComponent*), 15
- ## D
- `data()` (in module *cwlkernel.kernel\_magics*), 11
- `display_data()` (in module *cwlkernel.kernel\_magics*), 12
- `display_data_csv()` (in module *cwlkernel.kernel\_magics*), 12
- `display_data_image()` (in module *cwlkernel.kernel\_magics*), 12
- `do_complete()` (*cwlkernel.CWLKernel.CWLKernel* method), 13
- `do_execute()` (*cwlkernel.CWLKernel.CWLKernel* method), 13
- ## E
- `edit()` (in module *cwlkernel.kernel\_magics*), 12
- `execute()` (*cwlkernel.CoreExecutor.CoreExecutor* method), 10
- `execute()` (*cwlkernel.kernel\_magics.ExecutionMagics* static method), 11
- `execute_with_provenance()` (*cwlkernel.kernel\_magics.ExecutionMagics* static method), 11
- `ExecutionMagics` (class in *cwlkernel.kernel\_magics*), 11
- ## F
- `files_counter` (*cwlkernel.IOManager.IOFileManager* attribute), 12
- ## G
- `get_current_code()` (*cwlkernel.CWLBuilder.CWLSnippetBuilder* method), 13
- `get_files()` (*cwlkernel.IOManager.IOFileManager* method), 12
- `get_files_registry()` (*cwlkernel.IOManager.IOFileManager* method), 12
- `get_files_uri()` (*cwlkernel.IOManager.IOFileManager* method), 12
- `get_hostname()` (*cwlkernel.CWLLogger.CWLLogger* class method), 13
- `get_input()` (*cwlkernel.cwlrepository.CWLComponent.WorkflowComponent* method), 15
- `get_instance()` (*cwlkernel.cwlrepository.cwlrepository.WorkflowRepository* class method), 15
- `get_last_result_by_id()` (*cwlkernel.IOManager.ResultsManager* method), 12
- `get_output()` (*cwlkernel.cwlrepository.CWLComponent.WorkflowComponent* method), 15
- `get_past_results()` (*cwlkernel.CWLKernel.CWLKernel* method), 13
- `get_pid()` (*cwlkernel.CWLKernel.CWLKernel* method), 13
- `get_running_kernels()` (*cwlkernel.CWLLogger.CWLLogger* class method), 13
- `get_storage_path()` (*cwlkernel.CWLLoggerStorageManager.CWLLoggerStorageManager* method), 14
- `get_workflow_component()` (*cwlkernel.cwlrepository.CWLComponent.WorkflowComponentFactory* method), 15
- `github_import()` (in module *cwlkernel.kernel\_magics*), 12
- ## H
- `history` (*cwlkernel.CWLKernel.CWLKernel* attribute), 14
- ## I
- `id` (*cwlkernel.cwlrepository.CWLComponent.WorkflowComponent* attribute), 15
- `implementation` (*cwlkernel.CWLKernel.CWLKernel* attribute), 14
- `implementation_version` (*cwlkernel.CWLKernel.CWLKernel* attribute), 14
- `inputs` (*cwlkernel.cwlrepository.CWLComponent.CWLTool* attribute), 14
- `inputs` (*cwlkernel.cwlrepository.CWLComponent.CWLWorkflow* attribute), 15
- `inputs` (*cwlkernel.cwlrepository.CWLComponent.WorkflowComponent* attribute), 15
- `IOFileManager` (class in *cwlkernel.IOManager*), 12
- ## J
- `JupyterFactory` (class in *cwlkernel.CoreExecutor*), 10
- ## L
- `language_info` (*cwlkernel.CWLKernel.CWLKernel* attribute), 14
- `language_version` (*cwlkernel.CWLKernel.CWLKernel* attribute), 14
- `load()` (*cwlkernel.CWLLogger.CWLLogger* method), 13
- `load()` (*cwlkernel.CWLLoggerStorageManager.CWLLoggerStorageManager* method), 14

logs () (in module cwlkernel.kernel\_magics), 12

## M

magics () (in module cwlkernel.kernel\_magics), 12

MagicSnippetBuilder (class in cwlkernel.kernel\_magics), 11

ManualWorkflowComposer (class in cwlkernel.kernel\_magics), 11

MissingIdError, 15

## N

new\_workflow () (cwlkernel.kernel\_magics.ManualWorkflowComposer static method), 11

new\_workflow\_add\_input () (cwlkernel.kernel\_magics.ManualWorkflowComposer static method), 11

new\_workflow\_add\_output\_source () (cwlkernel.kernel\_magics.ManualWorkflowComposer static method), 11

new\_workflow\_add\_step () (cwlkernel.kernel\_magics.ManualWorkflowComposer static method), 11

new\_workflow\_add\_step\_in () (cwlkernel.kernel\_magics.ManualWorkflowComposer static method), 11

new\_workflow\_build () (cwlkernel.kernel\_magics.ManualWorkflowComposer static method), 11

## O

outputs (cwlkernel.cwlrepository.CWLComponent.CWLTool attribute), 14

outputs (cwlkernel.cwlrepository.CWLComponent.CWLWorkflow attribute), 15

outputs (cwlkernel.cwlrepository.CWLComponent.WorkflowComponent attribute), 15

## P

parse\_args () (cwlkernel.kernel\_magics.Scatter class method), 11

parser (cwlkernel.kernel\_magics.Scatter attribute), 11

properties (cwlkernel.CWLExecuteConfigurator.CWLExecuteConfigurator attribute), 11

ProvenanceFactory (class in cwlkernel.CoreExecutor), 10

## R

read () (cwlkernel.IOManager.IOFileManager method), 12

remove () (cwlkernel.cwlrepository.CWLComponent.CWLWorkflow method), 15

remove () (cwlkernel.IOManager.IOFileManager method), 12

results\_manager (cwlkernel.CWLKernel.CWLKernel attribute), 14

ResultsManager (class in cwlkernel.IOManager), 12

runtime\_directory (cwlkernel.CWLKernel.CWLKernel attribute), 14

## S

sample\_csv () (in module cwlkernel.kernel\_magics), 12

save () (cwlkernel.CWLLogger.CWLLogger method), 13

save () (cwlkernel.CWLLoggerStorageManager.CWLLoggerStorageManager method), 14

Scatter (class in cwlkernel.kernel\_magics), 11

scatter () (cwlkernel.kernel\_magics.Scatter static method), 11

scatter\_template (cwlkernel.kernel\_magics.Scatter attribute), 11

send\_error\_response () (cwlkernel.CWLKernel.CWLKernel method), 14

send\_json\_response () (cwlkernel.CWLKernel.CWLKernel method), 14

send\_text\_to\_stdout () (cwlkernel.CWLKernel.CWLKernel method), 14

set\_data () (cwlkernel.CoreExecutor.CoreExecutor method), 10

set\_workflow\_path () (cwlkernel.CoreExecutor.CoreExecutor method), 10

snippet () (cwlkernel.kernel\_magics.MagicSnippetBuilder static method), 11

steps (cwlkernel.cwlrepository.CWLComponent.CWLWorkflow attribute), 15

suggest\_execution\_id () (cwlkernel.kernel\_magics.ExecutionMagics static method), 11

system () (in module cwlkernel.kernel\_magics), 12

## T

to\_dict () (cwlkernel.CWLLogger.CWLLogger method), 13

to\_dict () (cwlkernel.cwlrepository.CWLComponent.CWLTool method), 14

to\_dict () (cwlkernel.cwlrepository.CWLComponent.CWLWorkflow method), 15

to\_dict () (cwlkernel.cwlrepository.CWLComponent.WorkflowComponent method), 15

to\_yaml () (cwlkernel.cwlrepository.CWLComponent.CWLTool method), 15

to\_yaml () (cwlkernel.cwlrepository.CWLComponent.CWLWorkflow method), 15

`to_yaml()` (*cwlkernel.cwlrepository.CWLComponent.WorkflowComponent method*), 15

## V

`validate()` (*cwlkernel.cwlrepository.CWLComponent.CWLWorkflow method*), 15

`validate_input_files()` (*cwlkernel.CoreExecutor.CoreExecutor class method*), 10

`view_tool()` (*in module cwlkernel.kernel\_magics*), 12

`visualize_graph()` (*in module cwlkernel.kernel\_magics*), 12

## W

`workflow_repository` (*cwlkernel.CWLKernel.CWLKernel attribute*), 14

`WorkflowComponent` (*class in cwlkernel.cwlrepository.CWLComponent*), 15

`WorkflowComponentFactory` (*class in cwlkernel.cwlrepository.CWLComponent*), 15

`WorkflowRepository` (*class in cwlkernel.cwlrepository.cwlrepository*), 15

`write()` (*cwlkernel.IOManager.IOFileManager method*), 12