developer.skatelescope.org Documentation

Release 0.1.0-alpha

Marco Bartolini

Jun 03, 2020

CONTENTS:

This project is developing the Local Monitoring and Control (LMC) prototype for the Square Kilometre Array.

CHAPTER

ONE

GETTING STARTED

1.1 Set up your development environment

Two approaches to setting up a development environment are documented below. One is the "default" option currently documented in the SKA software developer portal. The other is our "recommended" approach.

- (The Default approach) In this approach, you will create an Ubuntu 18.04 virtual machine using VirtualBox (though you could do the same on a physical machine), then use an Ansible playbook to install Tango.
  - In this approach, your system has everything you need to run and test your code.
  - However we have found it to be quite brittle. You wouldn't want to fiddle with the settings, lest things go wrong. And when they do go wrong, they can be very hard to debug.
- (The Recommended approach) Develop against the SKA Docker images. In this approach, you will install Docker, and use the SKA Docker image for testing code.
  - In this approach, your system does not have Tango installed, so you cannot run and test your code locally.
  - Because you don't be running your code locally, you have flexibility in how you set your local system up. The only real requirement is Docker.
  - You will be testing your code in a Docker container running the standard SKA docker image, so you don't need to worry that your code might not port to other SKA environments.
  - The Docker image will be kept up to date; you don't need to worry about updating it yourself.

  The approach documented below uses an Ubuntu 20.04 image, with Visual Studio Code (VScode) as an IDE. VScode integrates very well with Docker.

1.1.1 The SKA software developer portal way

1. Follow the instructions on the Tango Development Environment set up page.
2. Set up your itango docker container to mount your host working directory. This will allow you to launch locally hosted code within the itango container. To do this, edit `/usr/src/ska-docker/docker-compose/itango.yml` and add the following lines under the itango service definition:

```
volumes:
  - ${HOME}:/hosthome:rw
```

3. Clone our GitLab repo.
4. Verify your setup:

```
$ cd /usr/src/ska-docker/docker-compose
$ make start itango #not needed if it already shows in "make status"
$ docker exec -it -e PYTHONPATH=/hosthome/ska-logging:/hosthome/lmc-base-classes/
↪src \
  itango python3 \
  /hosthome/ska-low-mccs/src/ska/mccs/MccsMaster.py -?
usage :  MccsMaster instance_name [-v[trace level]] [-nodb [-dlist <device name␣
↪list>]]
Instance name defined in database for server MccsMaster :
$ docker exec -it -e PYTHONPATH=/hosthome/ska-logging:/hosthome/lmc-base-classes/
↪src \
  itango tango_admin --add-server MccsMaster/01 MccsMaster lfaa/master/01
$ docker exec -it -e PYTHONPATH=/hosthome/ska-logging:/hosthome/lmc-base-classes/
↪src \
  itango python3 \
  /hosthome/ska-low-mccs/src/ska/mccs/MccsMaster.py 01
1|2020-03-13T05:27:15.844Z|INFO|MainThread|write_loggingLevel|SKABaseDevice.py
↪#490|tango-device:lfaa/master/01|Logging level set to LoggingLevel.INFO on␣
↪Python and Tango loggers
1|2020-03-13T05:27:15.845Z|INFO|MainThread|update_logging_handlers|SKABaseDevice.
↪py#169|tango-device:lfaa/master/01|Logging targets set to []
1|2020-03-13T05:27:15.846Z|INFO|MainThread|init_device|SKABaseDevice.py#399|tango-
↪device:lfaa/master/01|No Groups loaded for device: lfaa/master/01
1|2020-03-13T05:27:15.846Z|INFO|MainThread|init_device|SKABaseDevice.py#401|tango-
↪device:lfaa/master/01|Completed SKABaseDevice.init_device
Ready to accept request
```

1.1.2 (Recommended) The Docker way

1. You will need an Ubuntu 20.04 physical or virtual machine.
   - These instructions assume Ubuntu 20.04, but you could adapt them to a different Ubuntu version, a different Linux flavour, or even a different operating system such as Windows. (If you adapt these instructions to a different system, please consider contributing to this documentation.)
2. Install Docker CE. Unfortunately you can't just *sudo apt install docker* because that would install a Canonical build of Docker named Docker.io, and this is not recommended. We'll need to work a little harder to install Docker CE. We can use *apt* but first we need to add the Docker apt repository, and in order to do that we will need to install the Docker repo public key, and these steps will themselves require installation of packages:

```
$ sudo apt install apt-transport-https ca-certificates curl gnupg-agent software-
↪properties-common
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
$ sudo add-apt-repository "deb [arch-amd64] https://download.docker.com/linux/
↪ubuntu #(lsb_release -cs) stable"
$ sudo apt-get update
$ sudo apt install docker-ce docker-ce-cli
```

3. Test your install:

```
$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:6a65f928fb91fcfbc963f7aa6d57c8eeb426ad9a20c7ee045538ef34847f44f1
Status: Downloaded newer image for hello-world:latest
```

(continues on next page)

(continued from previous page)

```
Hello from Docker!
This message shows that your installation appears to be working correctly.
...
```

4. At this point you can only run this command as sudo, because you are not a member of the docker group. The docker group is created but it is empty. Add yourself to the docker group:

```
$ sudo usermod -aG docker $USER
```

5. If you are running on a virtual machine, you should restart the VM now. If you are on a physical Ubuntu machine, you must at least log out and log back in. Then verify that you can run docker without sudo:

```
$ docker run hello-world
```

6. Is git installed? Try

```
$ git --version
```

   and if the command is not found then

```
$ sudo apt install git
```

7. Set up git:

```
$ git config --global user.name "Your Name"
$ git config --global user.email "youremail@domain.com"
```

   This is probably a good time to set up commit-signing too. Follow the instructions at the SKA Working with Git page.

8. Clone our repo:

```
$ cd ~
$ git clone https://gitlab.com/ska-telescope/ska-low-mccs.git
```

9. Install Visual Studio Code (hencefort "VScode"). This step is easy: just install it via the "Ubuntu Software" app.

10. Open VScode. Choose "Open folder..." and select the folder for our repo. You should see the contents of our repo open into your sidebar.

    - If you don't: there is a column of icons along the left-hand side that controls which sidebar you are seeing. Click on the first one. *Now* you should set the contents of our repo in the sidebar.

11. Click on the "Extensions" sidebar icon (it's the one that looks like a square jigsaw puzzle.) Search for and install "Remote-Containers".

12. Once the extension is installed, you should see a pop-up box telling you that it has detected a *.devcontainers* folder in our repo, and asking if you would like to reload the repo in a remote container. Choose yes. You'll see a popup message that it is "Starting with Dev Container".

    - If you left it too long and the pop-up disappeared, then <Ctrl-Shift-P> is your friend: it opens a search box for all of the many commands supported by VScode. Type "Remote" and you will find an option along the lines of "Rebuild and reopen in container".

    - The first time you do this, it will take a very long time, because the Docker image has to be downloaded. Once downloaded, the image will be cached, so it will be much faster in future.

- If you click on the message box, it will open a terminal showing you that things are happening. Go have a cup of tea.

13. You're ready to develop!

    - The other sidebar you need to know about is the git sidebar. This sidebar helps you keep track of git status and perform git commands. For example, to make a commit, simply stage the edited files that you want to commit (the "+" button), provide a message in the message box, and hit the commit (tick) button. For more complex git stuff like stashing, rebasing, etc, it might be possible to do it through the GUI, but you might still find it easier to do it in the terminal.

CHAPTER

TWO

API

2.1 MCCS Master

2.2 MCCS Subarray

2.3 MCCS Station

2.4 MCCS Station Beam

2.5 MCCS Tile

2.6 MCCS Antenna

2.7 control_model

2.8 utils

ska.low.mccs.utils.**tango_raise**(*msg*, *reason='API_CommandFailed'*, *severity=tango.ErrSeverity.ERR*, *_origin=None*)

Helper function to provide a concise way to throw *tango.Except.throw_exception*

Example:

```
class MyDevice(Device):
    @command
    def some_command(self):
        if condition:
            pass
        else:
            tango_throw("Condition not true")
```

Parameters

- **msg** (*[type]*) – [description]
- **reason** (*str, optional*) – the tango api DevError description string, defaults to "API_CommandFailed"

- **severity** (*tango.ErrSeverity*, optional) – the tango error severity, defaults to *tango.ErrSeverity.ERR*
- **_origin** (*str, optional*) – the calling object name, defaults to None (autodetected) Note that autodetection only works for class methods not e.g. decorators

ska.low.mccs.utils.**call_with_json**(*func*, ***kwargs*)

Allows the calling of a command that accepts a JSON string as input, with the actual unserialised parameters.

Parameters

- **func** – the function to call
- **kwargs** – parameters to be jsonified and passed to func

Ptype func callable

Ptype kwargs any

Returns the return value of func

Example Suppose you need to use MccsMaster.Allocate() to command a master device to allocate certain stations and tiles to a subarray. Allocate() accepts a single JSON string argument. Instead of

parameters={"id": id, "stations": stations, "tiles": tiles} json_string=json.dumps(parameters) master.Allocate(json_string)

save yourself the trouble and

call_with_json(master.Allocate, id=id, stations=stations, tiles=tiles)

class ska.low.mccs.utils.**json_input**(*schema_path=None*)

Method decorator that parses and validates JSON input into a python object. The wrapped method is thus called with a JSON string, but can be implemented as if it had been passed an object.

If the string cannot be parsed as JSON, an exception is raised.

Parameters **schema_path** – an optional path to a schema against which the JSON should be validated. Not working at the moment, so leave it None.

Ptype string

Raises

- **FileNotFoundException** – if no file is found at the schema path provided
- **json.JSONDecodeError** – if the file at the specified schema path is not valid JSON

Example Conceptually, MccsMaster.Allocate() takes as arguments a subarray id, an array of stations, and an array of tiles. In practice, however, these arguments are encoded into a JSON string. Implement the function with its conceptual parameters, then wrap it in this decorator:

@json_input def MccsMaster.Allocate(id, stations, tiles):

The decorator will provide the JSON interface and handle the decoding for you.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX