ARTIQ induction - Basics

Charles Baynham

2024-02-05

---

1 ARTIQ Basics

1.1 Setup

1. Clone the https://gitlab.com/aion-physics/code/artiq/artiq_training repository. Make your own branch and push it regularly.

2. Follow the readme instructions to set up Nix and a few other convenience features.

3. Open the repository in Visual Studio Code and install the "python" and "direnv" extensions.

Infobox

We have glossed over the details of what just happened. For the curious, this repository is a Nix "flake" since it contains a `flake.nix` file. This imports other flakes and uses them to build a software environment which contains ARTIQ and various other things. The bulk of the work is done by the `PyAION` package which is both a flake and a python package - `artiq_training` calls a Nix function in `PyAION`.

This involved fetching various packages from Gitlab and other places, as well as python packages from the PyPI servers and pre-built packages from the AION binary cache. To avoid you having to rebuild all this stuff from source, the AION `cachix` binary cache has been configured, to which GitLab automatically pushes built packages when anyone uploads a new commit.

Then, `direnv` has automatically entered that environment when you `cd` into the `artiq_training` folder, allowing you to use ARTIQ commands. This is equivalent to using `nix develop` to enter a new shell.

Finally, by installing the `direnv` extension into Visual Studio Code, you allow the `Python` extension to find all our python package, giving you code suggestions, documentation and error detection to make your life easier.

1.2 Basics and Datasets

1. Run an ARTIQ master session using the command `artiq_master --git` and connect to it with a dashboard (try `artiq_master --help` and `artiq_dashboard --help`).
2. Write an experiment which writes "Hello World" to the log and run it from the dashboard.
3. Write an experiment which writes messages at each of the five log levels - control their visibility with the log filter.
4. a. Write a dataset from an ARTIQ experiment with an array of numbers (maybe a sin curve, or your favourite function).
   b. Plot it in an applet (not possible on WSL).
   c. Add some Arguments to let the user control your data output
5. Quit ARTIQ and use your python interface of choice (or Jupyter notebook if in doubt) to access the dataset you made and plot it in matplotlib. You will need to use the python package `HDF5` which is already installed for you.

1.3 Devices

ARTIQ is designed to interface with physical devices. The main device that ARTIQ interfaces with is the Sinara core (we will lots of time on this later), but the ARTIQ platform handles all devices the same way.

In this module, we will explore this interface by making our own "device" which does nothing other than print text to the screen. We will add this "virtual device" to the ARTIQ device_db and use it from Experiments.

1. Create a file in the root of your repository (not in the "repository" folder but one level higher) called e.g. "mydevice.py"
2. Within this file, define a class which has some methods. Following the usual python conventions, you would call the class e.g. "MyDevice". Make sure that your class:
   a. …has a method "set_number" which accepts a number
   b. …has a method "add_to_number" which adds to the number which was previously set
   c. …has a method "get_number"
   d. …accepts an optional parameter "extra_offset" in its constructor, with default value = 0. This should be added to all the nubmers returned by `get_number`. (Hint: ARTIQ will pass two positional arguments to your contructor which you do not need for this example. You can ignore them by using a function definition like `def __init__(self, *args, get_number=0):`)
   e. Test these methods by making a copy of your class outside of ARTIQ, e.g. in `ipython`.
3. Using your device

a. Open the `device_db.py` file and read the comments.

b. Add a "local" device to the `device_db` which tells ARTIQ the recipe to create a copy of your object. (You'll need to tell ARTIQ to "Scan device database" to read the changed `device_db`)

c. Make a new ARTIQ experiment which uses `self.get_device` or `self.setattr_device` to request a copy of your device.

d. Call some of your device's methods and see what happens

4. *(extra credit)* Repackage your device as a controller. To do this, you will need to:

a. Make your device file executable by adding an `if __name__=="__main__":` block

b. In this block, launch a `sipyco simple_server_loop` on an unused port

c. Change the definition in your `device_db` to launch a controller. You'll need to ignore the `#!/usr/bin/env python3` advice from the ARTIQ manual - use `python <your file name>` instead for the command

d. Try running consecutive experiments that access the device from ARTIQ - notice that it now remembers values from run to run.