MWA ASVO: Command Line Client (manta-ray-client)

Overview

The "manta-ray-client" is the name given to a set of Python APIs and a command line client which can be used to submit data jobs, monitor the progress of the data jobs and download the resulting data product.

The source code and documentation reside on a public repository on github: MWATelescope/manta-ray-client.

mwa_client is a helper script which provides the following functions:

- Submit MWA ASVO jobs in bulk
- Monitor the status of your jobs
- Download your completed jobs

There are two types of MWA ASVO jobs:

- Conversion: Average, convert and download a visibility data set (and optionally apply calibration solutions).
- Download: Package and download a raw visibility data set. (This is recommended for advanced users, as the raw visibility files are in an MWA-specific format and require conversion and calibration).

Prerequisites

Python

Supported Python versions:

- Python 3.8
- Python 3.7
- Python 3.6
- Python 2.7 works, however see note below:

⚠ Python2.x is now end of life, so we recommend making the switch to Python versions at or above Python 3.6 ASAP. At time of writing, manta-ray-client worked in Python2.7. Support for EOL versions of Python will be on a best effort basis where it is not a burden to do so, but will not go on indefinitely.

Existing MWA ASVO Account and API Key Set

You must have an activated account on the MWA ASVO website to use the mwa_client.

Set your API key as an environment variables in Linux (usually in your .bashrc file). You can get your API key from Your Profile page on the MWA ASVO website. Alternatively you can set it in your current shell session as follows:

```
~$ export MWA_ASVO_API_KEY=your_api_key
```

Install Method #1 (Native)

This method will install the manta-ray-client as a native python application in a new python virtual environment.

```
# Clone the git repository
~$ git clone https://github.com/ICRAR/manta-ray-client.git

# Create a virtual environment
~$ python3 -m venv env

# Activate the virtual environment
~$ source env/bin/activate

# Install manta-ray-client and requirements
(env)~$ cd manta-ray-client

(env)~/manta-ray-client$ pip3 install -r requirements.txt

(env)~/manta-ray-client$ python3 setup.py install
```

Running mwa_client natively

```
# Activate the virtual environment
~$ source env/bin/activate

# Run mwa_client (bring up command line help)
(env)~$ mwa_client --help
```

Install Method #2 (Docker)

If you prefer, you can also run the manta-ray-client as a Docker container instead of installing it locally. This assumes you have docker installed on your machine. If not please see the Get Docker (external link) page for instructions.

```
# Clone the git repository
~$ git clone https://github.com/mwatelescope/manta-ray-client.git

# Build the docker image
~$ cd manta-ray-client

~/manta-ray-client$ docker build --tag manta-ray-client:latest .
```

Running the docker image

Once the image is built, you can run the mwa_client directly. The below command will:

- Create and launch and instances of the image (called a container),
- Map '/your/host/data/path/' which should be a directory on your machine, to the container's /data directory
- Remove the container once it has finished the command
- Map your machine's MWA_ASVO_API_KEY environment variable into the container so it has your MWA ASVO API key
- Then 'mwa_client -w all -d /data' will run the mwa_client and download all 'Completed' jobs to the container's /data directory (which we mapped to '/your/host/data/path/' on your machine)

```
~$ docker run --name my_mwa_client --entrypoint="" --volume=/your/host/data/path/:/data --rm=true -e
MWA_ASVO_API_KEY manta-ray-client:latest mwa_client -w all -d /data
```

Usage

Command Line Options

Whichever way (above) you run the mwa_client, the following guide will describe it's usage and options.

```
mwa_client -c csvfile -d destdir          Submit jobs in the csv file, monitor them, then download the files,
then exit
mwa_client -c csvfile -s                  Submit jobs in the csv file, then exit
mwa_client -d destdir -w JOBID            Download the job id (assuming it is ready to download), then exit
mwa_client -d destdir -w all              Download any ready to download jobs, then exit
mwa_client -d destdir -w all -e error_file Download any ready to download jobs, then exit, writing any errors
to error_file
mwa_client -l                             List all of your jobs and their status, then exit

optional arguments:
  -h, --help            show this help message and exit
  -s, --submit-only     submit job(s) from csv file then exit (-d is ignored)
  -l, --list-only       List the user's active job(s) and exit immediately
                        (-s, -c & -d are ignored)
  -w DOWNLOAD_JOB_ID, --download-only DOWNLOAD_JOB_ID
                        Download the job id (-w DOWNLOAD_JOB_ID), if it is ready;
                        or all downloadable jobs (-w all | -w 0), then exit (-s, -c & -l are ignored)
  -c FILE, --csv FILE   csv job file
  -d DIR, --dir DIR     download directory
  -e ERRFILE, --error-file ERRFILE, --errfile ERRFILE
                        Write errors in json format to an error file
  -v, --verbose         verbose output
```

CSV File Format

Each row is a single job definition and each CSV element must be a key=value pair. Whitespace (blank rows) and comments (lines beginning with #) are allowed. Please see the included example.csv for several full working examples.

Conversion Job Options

- `obs_id: <integer>`
  - Observation ID
- `job_type: c`
  - Always 'c' for conversion jobs.
- `timeres: <decimal>`
  - Time resolution: average N seconds of time steps together before writing output.
- `freqres: <integer>`
  - Average N kHz bandwidth of fine channels together before writing output.
- `edgewidth: <integer>`
  - Flag the given width (in kHz) of edge channels of each coarse channel.
  - Defaults to 80 kHz.
  - Set to 0 kHz to disable edge flagging.
- `conversion: <ms || uvfits>`
  - Output format.
  - `ms`: CASA measurement set.
  - `uvfits`: uvfits output.

Optional options

- To enable an option, set value to true e.g. `norfi=true`
- If you omit an option it is equivalent to false. e.g. not specifying norfi is equivalent to `norfi=false`.
- Recommended defaults:
  - `allowmissing: true` Do not abort when not all GPU box (visibility) files are available.
  - `flagdcchannels: true` Flag the centre/DC channel of each coarse channel.

RFI options:

If omitted, the below options default to false.

- `norfi: true` Do not perform RFI detection.
- `noprecomputedflags: true` Do not use observatory generated precomputed flags. The combination of (or lack of) the above RFI options provides the following capabilities:
- (Default- i.e. neither option specified) Use precomputed flags if they exist, if not perform RFI detection.
- `noprecomputedflags: true` Ignore precomputed flags if they exist and perform RFI flagging instead.
- `norfi: true, noprecomputedflags: true` Do not flag any RFI even if there are precomputed flag files available.

Pointing options:

If none of the 3 options below are set, the observation's phase centre is assumed to be used.

- `usepcentre: true` Centre on the observation's pointing centre.
- `phasecentrera: <ra formatted as: 00h00m00.0s>` ICRS (J2000.0). Centre on a custom phase centre with this right ascension (must include `phasecentredec`).
- `phasecentredec: <dec formatted as: +00d00m00.0s>` ICRS (J2000.0). Centre on a custom phase centre with this declination (must include `phasecentrera`).

Expiry option:

If omitted, defaults to 7 days.

- `expiry_days: <integer 1-7>` Once job is completed, how long should the download stay on the MWA ASVO server before being removed. Set this lower if you want to process more jobs.

Other options:

If the below options are omitted, they default to false.

- `calibrate: true` Apply a calibration solution to the dataset, if found. If not found, the job will fail- in this case you can resubmit the job without this option for uncalibrated raw visibilities. See: Data Access/MWA ASVO Calibration Option on the MWA Telescope Wiki for more information.
- `nostats: true` Disable collecting statistics.
- `nogeom: true` Disable geometric corrections.
- `noantennapruning: true` Do not remove the flagged antennae.
- `noflagautos: true` Do not flag auto-correlations.
- `nosbgains: true` Do not correct for the digital gains.
- `noflagmissings: true` Do not flag missing gpu box files (only makes sense with `allowmissing`).
- `sbpassband: true` Apply unity passband (i.e. do not apply any passband corrections)

Example line in csv file

```
obs_id=1110103576, job_type=c, timeres=8, freqres=40, edgewidth=80, conversion=ms, calibrate=true,
allowmissing=true, flagdcchannels=true
```

Download Job Options

- `obs_id: <integer>`
  - Observation ID
- `job_type: d`
  - Always 'd' for download jobs.
- `download_type: <vis_meta || vis>`
  - `vis_meta`: download visibility metadata only (metafits and RFI flags).
  - `vis`: download raw visibility data sets and metadata (raw visibility files, metafits and RFI flags).

Example lines in csv file

```
obs_id=1110103576, job_type=d, download_type=vis
obs_id=1110105120, job_type=d, download_type=vis_meta
```

Understanding and using the error file output

You can get a machine readable error file in JSON format by specifying "-e" | "--error-file" | "--errfile" on the command line. This might be useful if you are trying to automate the download and processing of many observations and you don't want to try and parse the human readable standard output.

An example of the format is below, with two jobs with errors:

```
[
    {
        "obs_id": "1216295963",
        "job_id": 28979,
        "result": "Error: an error message"
    },
    {
        "obs_id": "1216298341",
        "job_id": 28980,
        "result": "Error: some error message"
    }
]
```

Since this is JSON, in python you could simply use the below code to iterate through any errors by deserialising the JSON string:

```
import json

# Open the error file mwa_client produced when using -e
with open("error.txt", "r") as f:
    # Read the JSON from the file into a string
    json_string = f.read()

    # Deserialise the JSON into a python list of objects
    result_list = json.loads(json_string)

    # Iterate through all of the errors
    for r in result_list:
        print("Job:{0} ObsId:{1} Result:{2}", r['job_id'], r['obs_id'], r['result'])
```