drive-casa

Release 0.7.5

June 06, 2016

Contents

Version 0.7.5

Welcome to drive-casa's documentation. If you're new here, I recommend you start with the *introduction*, or you could jump straight to the *example*.

Contents:

CHAPTER 1

Introduction to drive-casa

A Python package for scripting the NRAO CASA pipeline routines (casapy).

drive-casa provides an interface to allow dynamic interaction with CASA from a *separate* Python process, allowing utilization of CASA routines alongside other Python packages which may not easily be installed into the casapy environment.

For example, one can spawn an instance of casapy, send it some data reduction commands to run (while saving the logs for future reference), do some external analysis on the results, and then run some more casapy routines. All from within a standard Python script, and preferably from a virtualenv. This is particularly useful when you want to embed use of CASA within a larger pipeline which uses external Python libraries alongside CASA functionality.

drive-casa can be used to run plain-text casapy scripts directly; alternatively the package includes a set of convenience routines which try to adhere to a consistent style and make it easy to chain together successive CASA reduction commands to generate a casapy command-script programmatically; e.g.

importUVFITS -> Perform Clean on resulting MeasurementSet

is implemented like so:

```
ms = drivecasa.commands.import_uvfits(script, uvfits_path)
dirty_maps = drivecasa.commands.clean(script, ms, niter=0, threshold_in_jy=1,
                                      other_clean_args=clean_args)
```

1.1 Rationale

Newcomers to CASA should note that it is trivial to run simple Python scripts within the casapy environment, or even to launch casapy into a script directly from the command line, e.g.:

```
casapy --nologger -c hello_world.py
```

While this mostly works fine from a command line or within a shell script, things start to get messy if you want to run CASA functions alongside routines from external Python libraries.

casapy uses its own bundled-and-modified copy of the Python interpreter[*], so a first thought might be to try and install external libraries into the CASA environment directly, and then run everything via the casapy interpreter. Thanks to recent efforts, this is now possible. However it still breaks the virtualenv workflow, and requires that your external Python modules are compatible with the CASA-bundled version of Python.

Alternatively one can try to 'break-out' the casapy modules from the CASA environment, but this also requires binary compatibility and some monkeying around with embedded paths as detailed in this post from Peter Williams.

At a pinch, you might be tempted to try dumping CASA command scripts to file and then spawning a casapy instance via subprocess. **Don't.** This was how drive-casa got started, and I quickly ran into issues with casapy filling the stdin / stdout pipe buffers and causing the whole process to freeze up.

Which leads us to the drive-casa approach - emulate terminal interaction with casapy via use of pexpect. drive-casa can be installed along with any other Python packages in the usual Python package fashion, since we only interface with casapy indirectly via the command line. The downside is that data has to be written to file to transfer it between the standard Python script and the casapy environment, but it brings some added benefits:

Error handling CASA tasks do not, as far as I can tell, return useful values as standard (or even throw exceptions). Instead, since the over-riding assumption is that the package will be run in interactive mode, all information is written to stderr as part of the logging output, making it hard to programmatically verify if a task has completed sucessfully. drive-casa attempts to solve this by parsing the log output for 'SEVERE' warnings - the user may then choose to throw an exception when it is sensible to do so.

Logging / reproducibility If scripting the reduction of large amounts of data in batches, it is often useful to record logging information along with the data output, both for purposes of debugging and data provenance. As far as I can tell, CASA does not provide an interface to control or redirect the logging output once the program has been instantiated. drive-casa can work-around this issue by simply restarting CASA with a fresh logging location specified for each dataset.

1.2 Project status, licence and acknowledgement

drive-casa is BSD licensed. The package is now in use by a few people other than myself, and can reasonably be used 'in production'. Any bug-fixes or interface changes should be accompanied by a version increment, so you can be assured of stability by specifying the PyPI version. I'd be interested to hear if others find it useful, and welcome any bug reports or pull requests. Any major changes should be recorded in the change-log.

If you make use of drive-casa in work leading to a publication, I ask that you cite Staley and Anderson (2015) and the relevant ASCL entry.

1.3 Installation

Requirements:

- A working installation of casapy.
- pexpect (As listed in *requirements.txt*, installed automatically when using pip.)

drive-casa is *pip* installable, simply run:

```
pip install drive-casa
```

Warning: Multiprocessing bug with pexpect 3.3:
During 2015, the default version of pexpect available on PyPI was 3.3. If you wish to use drive-casa in a parallel-processing context, you should beware of this bug which means pexpect 3.3 is broken under multiprocessing. Fortunately, both the older pexpect 2.4 and the latest pexpect 4.0.1 seem to work fine.

1.4 Developer setup

Those wanting to modify the source will need a git checkout, followed by a git-submodule checkout to grab the test-data for the unittests. So a setup script might look like this:

```
git clone git@github.com:timstaley/drive-casa.git
cd drive-casa
git submodule init
git submodule update
pip install -r requirements # (grab pexpect)
cd tests
nosetests -sv
```

1.5 Documentation

Reference documentation can be found at http://drive-casa.readthedocs.org, or generated directly from the repository using Sphinx.

1.6 Usage

Creating an instance of the *drivecasa.interface.Casapy* class will start up casapy in the background, awaiting instruction. Class init arguments determine details such as where to find casapy, where to write the casapy logfile, etc. The *drivecasa.interface.Casapy.run_script()* and *drivecasa.interface.Casapy.run_script_from_file()* commands can then be used to send casapy a list of commands or a script to execute (through use of the casapy execfile function). Logging output from the commands executed is returned for inspection.

You are free to create the casapy scripts by any method you like, but a number of convenience functions are provided that aim to make this process simpler and more programmatic. These functions try to adhere to a consistent calling signature, as detailed under *drivecasa.commands*.

1.7 A Brief Example

Assuming you already have a uv-measurement dataset in uvFITS format, basic usage might go something like this:

```
from __future__ import print_function
import drivecasa
casa = drivecasa.Casapy()
script = []
uvfits_path = '/path/to/uvdata.fits'
vis = drivecasa.commands.import_uvfits(script, uvfits_path, out_dir='./')
clean_args = {
   "imsize": [512, 512],
   "cell": ['5.0arcsec'],
   "weighting": 'briggs',
      "robust": 0.5,
   }
dirty_maps = drivecasa.commands.clean(script, vis, niter=0, threshold_in_jy=1,
                                      other_clean_args=clean_args)
dirty_map_fits_image = drivecasa.commands.export_fits(script, dirty_maps.image)
```

```
print(script)
casa.run_script(script)
```

After which, there should be a dirty map converted to FITS format waiting for you.

1.8 See also

Note that drive-casa is designed as a fairly basic interface layer. If you're putting together a substantial pipeline, you will probably want to built up subroutines and data-structures around it, to keep your code manageable. For one such example, see chimenea, a pipeline for automated processing of multi-epoch radio observations.

CHAPTER 2

drivecasa API reference

Drive-casa is an interfacing package for scripting of CASA from a separate Python process (see *Introduction to drive-casa*).

The package includes several convenience routines that allow chaining of CASA commands, see `drivecasa.commands` module.

2.1 drivecasa.interface - Casapy interface class

class drivecasa.interface.**Casapy**(*casa_logfile=None*, *commands_logfile=None*, *casa_dir=None*, *working_dir='/tmp/drivecasa'*, *timeout=600*, *log2term=True*, *echo_to_stdout=False*)

Handles the interface with casapy.

Simply instantiate, then use member function 'run_script' to pass valid casapy commands (i.e. python function calls) to casapy.

Note: Imported into the root of the `drivecasa` package to provide convenient instantiation, e.g:

```
casa = drivecasa.Casapy()
casa.run_script(['tasklist'])
```

---

load_subroutines()

run_script(*script*, *raise_on_severe=True*, *timeout=-1*)

Run the commands listed in *script*.

Parameters

- **script** – A list of commands to execute. (One command per list element.)
- **raise_on_severe** – Raise a `RuntimeError` if SEVERE messages are encountered in the logging output. Set to `False` if you want to attempt to continue execution anyway (e.g. if you want to ignore errors caused by trying to re-import UVFITs data when the outputs are pre-existing from a previous run).
- **timeout** – If *-1* (the default, use the class default timeout). Otherwise, specifies timeout in seconds for this command. *None* implies no timeout (wait indefinitely).

Returns Tuple `(casa_out, errors)` Where `casa_out` is a line-by-line list containing the contents of the casapy terminal output, and `errors` is a line-by-line list of 'SEVERE' error messages.

run_script_from_file(*path_to_scriptfile*, *raise_on_severe=True*, *command_pre_logged=False*, *timeout=-1*)

Run the script at given path.

Parameters

- **path_to_scriptfile** – Can be relative or absolute, since we apply abspath conversion before passing to casapy.
- **raise_on_severe** – Raise a `RuntimeError` if SEVERE messages are encountered in the logging output. Set to `False` if you want to attempt to continue execution anyway (e.g. if you want to ignore errors caused by trying to re-import UVFITs data when the outputs are pre-existing from a previous run).
- **timeout** – If *-1* (the default, use the class default timeout). Otherwise, specifies timeout in seconds for this command. *None* implies no timeout (wait indefinitely).

Returns Tuple `(casa_out, errors)` Where `casa_out` is a line-by-line list containing the contents of the casapy terminal output, and `errors` is a line-by-line list of 'SEVERE' error messages.

2.2 drivecasa.casa_env - Shell environment configuration

Convenience routines for manipulating shell environments.

drivecasa.casa_env.**casapy_env**(*casa_topdir*)

Returns an environment dictionary configured for CASA execution.

Args:

•*casa_topdir*: should either contain the top-level directory containing CASA installation, or be set to None if casa is already available from the default environment.

Note: It's not a bad idea to always specify the casa dir anyway, so you don't have to rely on the environment paths being set up already.

2.3 drivecasa.commands - Convenience routines for building command lists

This subpackage provides convenience functions for composing casapy data-reduction scripts.

While the casapy scripts can be composed by hand, use of convenience functions helps to prevent syntax errors, and allows for various optional extras such as forcing overwriting of previous datasets, automatic derivation of output filenames, etc.

2.3.1 drivecasa.commands.reduction - Data reduction commands

Note:

All the data-reduction command composing functions have a common set of parameters:

- *script*: The list to which the requested commands should be appended.
- *out_dir*: The output directory to place output files in, using a derived filename.
- *out_path*: Overrides out_dir, specifies an output file / directory path exactly.
- *overwrite*: Deletes any pre-existing data at the output location - use with caution!

The composing functions return the paths to the files which should be created once the scripted command has been executed.

---

class drivecasa.commands.reduction.**CleanMaps**

A namedtuple for bunching together the paths to maps produced by clean.

Fields: `('image', 'model', 'residual', 'psf', 'mask')`

drivecasa.commands.reduction.**clean**(*script*, *vis_paths*, *niter*, *threshold_in_jy*, *mask=''*, *modelimage=''*, *other_clean_args=None*, *out_dir=None*, *out_path=None*, *overwrite=False*)

Perform clean process to produce an image/map.

If out_path is None, then the output basename is derived by appending a *.clean* or *.dirty* suffix to the input basename. The various outputs are then further suffixed by casa, e.g. *foo.clean.image*, *foo.clean.psf*, etc. Since multiple outputs are generated, this function returns a *CleanMaps* object detailing the expected paths.

NB Attempting to run with pre-existing outputs and `overwrite=False` *will not* throw an error, in contrast to most other routines. From the CASA cookbook, w.r.t. the outputs:

> "If an image with that name already exists, it will in general be overwritten. Beware using names of existing images however. If the clean is run using an imagename where <imagename>.residual and <imagename>.model already exist then clean will continue starting from these (effectively restarting from the end of the previous clean). Thus, if multiple runs of clean are run consecutively with the same imagename, then the cleaning is incremental (as in the difmap package)."

You can override this behaviour by specifying `overwrite=True`, in which case all pre-existing outputs will be deleted.

NB niter = 0 implies create a 'dirty' map, outputs will be named accordingly.

> **Warning:** This function can accept a list of multiple input visibilities. This functionality is not extensively tested and should be considered experimental - the CASA cookbook is vague on how parameters should be passed in this use-case.

Returns

expected_map_paths – namedtuple,

listing paths for resulting maps.

Return type *CleanMaps*

drivecasa.commands.reduction.**concat**(*script*, *vis_paths*, *out_basename=None*, *out_dir=None*, *out_path=None*, *overwrite=False*)

Concatenates multiple visibilities into one.

By default, output basename is derived by concatenating the basenames of the input visibilities, with the prefix *concat_*. However, this can result in something very long and unwieldy. Alternatively you may specify the exact out_path, or just the out_basename.

Returns Path to concatenated ms.

drivecasa.commands.reduction.**export_fits**(*script*, *image_path*, *out_dir=None*, *out_path=None*, *overwrite=False*)

Convert an image ms to FITS format.

Returns Path to resulting FITS file.

drivecasa.commands.reduction.**import_uvfits**(*script*, *uvfits_path*, *out_dir=None*, *out_path=None*, *overwrite=False*)

Import UVFITS and convert to .ms format.

If out_path is `None`, a sensible output .ms directory path will be derived by taking the FITS basename, switching the extension to .ms, and locating as a subdirectory of `out_dir`, e.g. if `uvfits_path = '/my/data/obs1.fits', out_dir = '/tmp/junkdata'` then the output data will be located at */tmp/junkdata/obs1.ms*.

Parameters

- **script** – List to which the relevant casapy command line will be appended.
- **uvfits_path** – path to input data file.
- **out_dir** – Directory in which to place output file. `None` signifies to place output .ms in same directory as the original FITS file.
- **out_path** – Provides an override to the automatic output naming system. If this is not `None` then the `out_dir` arg is ignored and the specified path used instead.
- **overwrite** – Delete any pre-existing data at the output path (danger!).

Returns Path to newly converted ms.

drivecasa.commands.reduction.**mstransform**(*script*, *vis_path*, *out_path*, *other_transform_args=None*, *overwrite=False*)

Useful for pre-imaging steps of interferometric data reduction.

Guide: http://www.eso.org/~scastro/ALMA/casa/MST/MSTransformDocs/MSTransformDocs.html

Returns out_path

2.3.2 drivecasa.commands.simulation - simulation commands

Provides convenience functions for composing casapy simulation scripts.

drivecasa.commands.simulation.**close_sim**(*script*)

Flush simulated data to disk and close simulator tool (*sm.close()*)

cf https://casa.nrao.edu/docs/CasaRef/simulator.close.html

drivecasa.commands.simulation.**corrupt**(*script*)

Apply pre-configured simulated noise via *sm.corrupt*

cf https://casa.nrao.edu/docs/CasaRef/simulator.corrupt.html

drivecasa.commands.simulation.**format_astropy_skycoord_as_casa_direction**(*skycoord*)

Parameters **skycoord** (*astropy.coordinates.SkyCoord*) – Sky position

Returns (str): casa *me.direction* instantiation expression.

drivecasa.commands.simulation.**format_astropy_time_as_casa_epoch**(*time*)

Parameters **time** (*astropy.time.Time*) – Reference time

Returns (str): casa *me.epoch* instantiation expression (uses UTC conversion from astropy Time).

drivecasa.commands.simulation.**make_componentlist**(*script*, *source_list*, *out_path*, *overwrite=True*)

Build a componentlist and save it to disk.

Runs *cl.done()* to clear any previous entries, the *cl.addcomponent* for each source in the list, and finally *cl.rename*, *cl.close*.

cf https://casa.nrao.edu/docs/CasaRef/componentlist-Tool.html

Typically used when simulating observations.

Parameters

- **script** (*list*) – List of strings to append commands to.
- **source_list** – List of (position, flux, frequency) tuples. Positions should be `astropy.coordinates.SkyCoord` instances, while flux and frequency should be quantities supplied using the `astropy.units` functionality.
- **out_path** (*str*) – Path to save the component list at
- **overwrite** (*bool*) – Delete any pre-existing component list at out_path.

drivecasa.commands.simulation.**observe**(*script*, *stop_delay*, *start_delay=<Quantity 0.0 s>*)

Simulate an empty-field observation's UVW data with *sm.observe*

cf https://casa.nrao.edu/docs/CasaRef/simulator.observe.html

Parameters

- **stop_delay** (*astropy.units.Quantity*) – Time-span. Stop observing this long after the reference time defined by *settimes()*.
- **start_delay** (*astropy.units.Quantity*) – Time-span. Start observing this long after the reference time defined by *settimes()*. (Defaults to 0, so the observation starts immediately at the reference time).

drivecasa.commands.simulation.**predict**(*script*, *component_list_path*)

Use *sm.predict* to add synthetic source-visibilities to a MeasurementSet.

cf https://casa.nrao.edu/docs/CasaRef/simulator.predict.html

drivecasa.commands.simulation.**set_simplenoise**(*script*, *noise_std_dev*)

Use *sm.setnoise* to assign a simple fixed-sigma noise to visibilities.

cf https://casa.nrao.edu/docs/CasaRef/simulator.setnoise.html

NB should be followed by a call to *corrupt* to actually apply the noise addition.

drivecasa.commands.simulation.**setauto**(*script*, *autocorr_weight=0.0*)

Set autocorrelation weight with *sm.setauto*.

cf https://casa.nrao.edu/docs/CasaRef/simulator.setauto.html

Parameters

- **script** (*list*) – casapy script-list
- **autocorr_weight** (*float*) – Weight to assign autocorrelations

drivecasa.commands.simulation.**setconfig**(*script*, *telescope_name*, *antennalist_path*)

Configure the telescope parameters with *sm.setconfig*

cf https://casa.nrao.edu/docs/CasaRef/simulator.setconfig.html

Parameters

- **script** (*list*) – casapy script-list

- **telescope_name** (*str*) – e.g. 'VLA'
- **antennalist_path** (*str*) – antenna-list config file

drivecasa.commands.simulation.**setfeed**(*script, mode='perfect X Y', pol=['']*)
Set feed polarisation with *sm.setfeed*

cf https://casa.nrao.edu/docs/CasaRef/simulator.setfeed.html

Parameters

- **script** (*list*) – casapy script-list
- **mode** (*str*) – choice between 'perfect R L' and 'perfect X Y'
- **pol** (*str*) – Polarization (undocumented).

drivecasa.commands.simulation.**setfield**(*script*, *pointing_centre*)
Set pointing centre of simulated field of view with *sm.setfield*.

cf https://casa.nrao.edu/docs/CasaRef/simulator.setfield.html

Parameters

- **script** (*list*) – casapy script-list
- **pointing_centre** (*astropy.coordinates.SkyCoord*) – Field pointing centre

drivecasa.commands.simulation.**setlimits**(*script*, *shadow_limit=0.001*, *elevation_limit=<Quantity 15.0 deg>*)
Set shadowing / elevation limits before simulated data are flagged.

Runs *sm.setlimits* cf https://casa.nrao.edu/docs/CasaRef/simulator.setlimits.html

Parameters

- **script** (*list*) – casapy script-list
- **shadow_limit** (*float*) – Maximum fraction of geometrically shadowed area before flagging occurs
- **elevation_limit** (*astropy.units.Quantity*) – Minimum elevation angle before flagging occurs

drivecasa.commands.simulation.**setpb**(*script*, *telescope_name*, *primary_beam_hwhm*, *frequency*)
Configure Gaussian primary beam parameters for a measurement simulation.

Runs *vp.setpbgauss* cf https://casa.nrao.edu/docs/CasaRef/vpmanager.setpbgauss.html

Parameters

- **script** (*list*) – casapy script-list
- **telescope_name** (*str*) – e.g. 'VLA'
- **primary_beam_hwhm** (*astropy.units.Quantity*) – HWHM radius, i.e. angular radius to point of half-maximum in primary beam.
- **frequency** (*astropy.units.Quantity*) – Reference frequency for primary beam.

drivecasa.commands.simulation.**setspwindow**(*script*, *freq_start*, *freq_resolution*, *freq_delta*, *n_channels*, *stokes='XX XY YX YY'*)
Define a spectral window with *sm.setspwindow*.

cf https://casa.nrao.edu/docs/CasaRef/simulator.setspwindow.html

Parameters

- **script** (*list*) – casapy script-list
- **freq_start** (*astropy.units.Quantity*) – Starting frequency for spectral window.
- **freq_resolution** (*astropy.units.Quantity*) – Frequency width of each channel.
- **freq_delta** (*astropy.units.Quantity*) – Frequency increment per channel.
- **n_channels** (*int*) – Number of channels
- **stokes** (*str*) – Stokes types to simulate

drivecasa.commands.simulation.**settimes**(*script*, *integration_time*, *reference_time*)

Set integration time, reference time with *sm.settimes*

cf https://casa.nrao.edu/docs/CasaRef/simulator.settimes.html

The 'reference time' defines an epoch, start and stop are defined relative to that epoch.

Parameters

- **integration_time** (*astropy.units.Quantity*) – Time-span of each integration.
- **reference_time** (*astropy.time.Time*) – Reference epoch.

2.4 drivecasa.utils - Miscellaneous subroutines

drivecasa.utils.**byteify**(*input*)

Co-erce unicode to 'bytestring'

(or string containing unicode, or dict containing unicode) Useful when e.g. importing filenames from JSON (CASA sometimes breaks if passed Unicode strings.)

cf http://stackoverflow.com/a/13105359/725650

drivecasa.utils.**derive_out_path**(*in_paths*, *out_dir*, *out_extension=''*, *strip_in_extension=True*, *out_prefix=None*)

Derives an 'output' path given some 'input' paths and an output directory.

In the simple case that only a single path is supplied, this is simply the pathname resulting from replacing extension suffix and moving dir, e.g.

`input_dir/basename.in` -> `output_dir/basename.out`

If the out_dir is specified as 'None' then it is assumed that the new file should be located in the same directory as the first input path.

In the case that multiple input paths are supplied, their basenames are concatenated, e.g.

`in_dir/base1.in + in_dir/base2.in` -> `out_dir/base1_base2.out`

If the resulting output path is identical to any input path, this raises an exception.

NB the extension should be supplied including the '.' prefix.

drivecasa.utils.**ensure_dir**(*dirname*)

Ensure directory exists.

Roughly equivalent to *mkdir -p*

drivecasa.utils.**get_box_mask_string**(*centre_pix_posns*, *width*)
Get a mask string representing box apertures about (x,y) tuples

drivecasa.utils.**get_circular_mask_string**(*centre_ra_dec_posns*, *aperture_radius='1arcmin'*)
Get a mask string representing circular apertures about (x,y) tuples

drivecasa.utils.**listify**(*x*)
Ensure x is a (non-string) iterable; if not, enclose in a list.

Returns x or [x], accordingly.

drivecasa.utils.**save_script**(*script*, *filename*)
Save a list of casa commands as a text file

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

d