Developing for CASA

(or in, alongside, in spite of)

or, Becoming a CASA Taskmaster

or, How to get your Python code in and working.

Steven T. Myers

National Radio Astronomy Observatory

Socorro, NM

CASA Environment

- casapy
  - IPython for interactive environment
  - Python accessible for scripting (e.g. execfile)
  - can import Python libraries (standard or custom)
- toolkit
  - functional interface to C++ code (libraries, d.o.)
  - contains atomic data access and processing commands
  - user contributions require full build access
- tasks
  - Python wrapper around toolkit and pythoncode
  - a (minimal) parameter setting interface
  - accessible to user-supplied functionality

The ALMA Development Aspect

- CASA = Python + Toolkit + Applications
  - You can develop in CASA at C++ level
    - “hard” but clearly possible (become a CASA developer)
  - If it’s a Python (2.7x) module/script CASA can use it
  - If you can run it in casapy you can use the toolkit
    - write a CASA task or function (or simple script)
  - If you have an app with command interface you can call it from CASA if it works on standard data formats
    - MS, casa images, FITS images, some flavors of uvfits, text, …
  - We should strive to conform to minimal common interface
    - e.g. Numpy arrays, matplotlib, use of other standard facilities
  - Tweaking CASA
    - possible to define other interfaces in CASA (functional? GUIs)

Target Rich Environment

- Possible targets for CASA-related development
  - operations on image (cube) data
    - extract pixels, manipulate, report results, possibly return to cube
    - examples: source/line fitting/extraction, filtering, statistics, transforms
    - also physical modelling (e.g. from spectral cube)
  - image visualization
    - interactive exploration, hardcopy, cross-matching, identification
    - possibly with built-in image operations
  - data-space operations
    - uv modelfitting, imaging, intereference mitigation, data visualization
  - modelling
    - simulation-to-image, simulation-to-data
  - threshold for study/proposal: significant scope and FTEs

What do I do?

- Write a function in Python
  - learn Python (e.g. python.org)
  - write <function>.py
  - bring into casapy
    - execfile('<function>.py') OR
    - import <function>
  - call function in casapy
    - <function>.<method>(<args>)
  - good for simple functionality
    - bypasses task parameter interface
    - see any Python reference on how to do this

What else can I do?

- Write a casapy Task
  - learn Python (e.g. python.org)
  - read (parts of) CASA User Reference & Cookbook
    - e.g. Appendix G, includes example
  - get existing task as template
    - currently need code/xmlcasa directory tree
  - put Python code into task_<task>.py
  - put params and help text into <task>.xml
  - use “buildmytasks task” from unix (outside casa)
    - compiles to .pyc and puts into mytasks.py
  - go into casapy and execfile(‘mytasks.py’)
    - to update task, need to restart casapy
  - future: an importmytask inside casapy

Task Interface

- standard tasking interface
  - use parameters set as global Python variables
    - set <param> = <value> (e.g. vis = ‘ngc5921.demo.ms’)
  - parameter manipulation commands
    - using inp , default , saveinputs , tget
  - execute
    - <taskname> or go (e.g. clean())
  - return values
    - some tasks return Python dictionaries, e.g. myval=imval()

Task Parameter Interface

- example task parameters with inp :

IPy:Jupiter

```
CASA <1>: default('clean')

CASA <2>: inp('clean')
#  clean :: Deconvolve an image with selected algorithm
vis                 =          ''         #  name of input visibility file
imagename           =          ''         #  Pre-name of output images
field               =          ''         #  Field Name
spw                 =          ''         #  Spectral windows:channels: '' is all
selectdata          =       False         #  Other data selection parameters
mode                =       'mfs'         #  Type of selection (mfs, channel, velocity, frequency)
niter               =         500         #  Maximum number of iterations
gain                =         0.1         #  Loop gain for cleaning
threshold           =    '0.0mJy'         #  Flux level to stop cleaning.  Must include units
psfmode             =     'clark'         #  method of PSF calculation to use during minor cycles
imagermode          =          ''         #   Use csclean or mosaic.  If '', use psfmode
multiscale          =          []         #  set deconvolution scales (pixels), default: multiscale=[] (standard CLEAN)
interactive         =       False         #  use interactive clean (with GUI viewer)
mask                =          []         #  cleanbox(es), mask image(s), and/or region(s)  used in cleaning
imsize              = [256, 256]          #  x and y image size in pixels, symmetric for single value
cell                = ['1.0arcsec', '1.0arcsec'] #  x and y cell size, default unit arcsec
phasecenter         =          ''         #  Image phase center: position or field index
restfreq            =          ''         #  rest frequency to assign to image (see help)
stokes              =         'I'         #  Stokes params to image (eg I,IV, QU,IQUV)
weighting           =   'natural'         #  Weighting to apply to visibilities
uvtaper             =       False         #  Apply additional uv tapering of  visibilities.
modelimage          =          ''         #  Name of model image(s) to initialize cleaning
restoringbeam       =        ['']         #  Output Gaussian restoring beam for CLEAN image
pbcor               =       False         #  Output primary beam-corrected image
minpb               =         0.1         #  Minimum PB level to use
async               =       False         #  If true the taskname must be started using clean(...)

CASA <3>:
```

Expandable Parameters

- boldface parameters are expandable
  - one level deep: parameter->sub-parameter

IPy:Jupiter

```
CASA <3>: tget('clean')
Restored parameters from file clean.last

CASA <4>: inp()
#  clean :: Deconvolve an image with selected algorithm
vis                 = 'ngc5921.usecase.ms.contsub' #  name of input visibility file
imagename           = 'ngc5921.usecase.clean' #  Pre-name of output images
field               =         '0'          #  Field Name
spw                 =          ''          #  Spectral windows:channels: '' is all
selectdata          =       False          #  Other data selection parameters
mode                =   'channel'          #  Type of selection (mfs, channel, velocity, frequency)
     nchan          =          46          #  Number of channels (planes) in output image
     start          =           5          #  first input channel to use
     width          =           1          #  Number of input channels to average

niter               =        6000          #  Maximum number of iterations
gain                =         0.1          #  Loop gain for cleaning
threshold           =         8.0          #  Flux level to stop cleaning.  Must include units
psfmode             =     'clark'          #  method of PSF calculation to use during minor cycles
imagermode          =          ''          #   Use csclean or mosaic.  If '', use psfmode
multiscale          =          []          #  set deconvolution scales (pixels), default: multiscale=[] (standard CLEAN)
interactive         =       False          #  use interactive clean (with GUI viewer)
mask                = [108, 108, 148, 148] #  cleanbox(es), mask image(s), and/or region(s)  used in cleaning
imsize              = [256, 256]           #  x and y image size in pixels, symmetric for single value
cell                = [15.0, 15.0]         #  x and y cell size, default unit arcsec
phasecenter         =          ''          #  Image phase center: position or field index
restfreq            =          ''          #  rest frequency to assign to image (see help)
stokes              =         'I'          #  Stokes params to image (eg I,IV, QU,IQUV)
weighting           =    'briggs'          #  Weighting to apply to visibilities
     robust         =         0.5          #  Briggs robustness parameter
     npixels        =           0          #  number of pixels to determine uv-cell size 0=> field of view

uvtaper             =       False          #  Apply additional uv tapering of  visibilities.
modelimage          =          ''          #  Name of model image(s) to initialize cleaning
```

Task Parameter Checking

- sanity checks of parameters in inp :
  - parameter types defined in <task>.xml

IPy:Jupiter

```
CASA <5>: psfmode='hogwarts'

CASA <6>: inp()
#  clean :: Deconvolve an image with selected
vis                 = 'ngc5921.usecase.ms.con                    ty file
imagename           = 'ngc5921.usecase.clean'
field               =        '0'        #  Fi
spw                 =         ''        #  Sp                   all
selectdata          =      False        #  Ot
mode                =   'channel'       #  Ty                   velocity, frequency)
     nchan          =         46        #  Number of channels (planes) in output image
     start          =          5        #  First input channel to use
     width          =          1        #  Number of input channels to average

niter               =       6000        #  Maximum number of iterations
gain                =        0.1        #  Loop gain for cleaning
threshold           =        8.0        #  Flux level to stop cleaning.  Must include units
psfmode             =  'hogwarts'       #  method of PSF calculation to use during minor cycles
imagermode          =         ''        #   Use csclean or mosaic.  If '', use psfmode
multiscale          =         []        #  set deconvolution scales (pixels), default: multiscale=[] (standard CLEAN)
interactive         =      False        #  use interactive clean (with GUI viewer)
mask                = [108, 108, 148, 148] #  cleanbox(es), mask image(s), and/or region(s)  used in cleaning
imsize              = [256, 256]        #  x and y image size in pixels, symmetric for single value
cell                = [15.0, 15.0]      #  x and y cell size, default unit arcsec
phasecenter         =         ''        #  Image phase center: position or field index
restfreq            =         ''        #  rest frequency to assign to image (see help)
stokes              =        'I'        #  Stokes params to image (eg I,IV, QU,IQUV)
weighting           =   'briggs'        #  Weighting to apply to visibilities
     robust         =        0.5        #  Briggs robustness parameter
     npixels        =          0        #  number of pixels to determine uv-cell size 0=> field of view

uvtaper             =      False        #  Apply additional uv tapering of  visibilities.
modelimage          =         ''        #  Name of model image(s) to initialize cleaning
restoringbeam       =       ['']        #  Output Gaussian restoring beam for CLEAN image
```

erroneous values in red

Tools in CASA

- CASA Toolkit underneath tasks
  - core AIPS++ code (mostly in C++)
- tools are functions
  - call from casapy as <tool>.<method>()
  - methods either set state or do something
  - can return objects or records (dictionaries)
  - default tool objects are pre-constructed
    - e.g. imager (im) , calibrater (cb), ms (ms) , etc. (see toolhelp)
- Historical Context
  - aips++ had only toolkit, in transition to CASA we were told by UGs to concentrate on Tasks…

CASA Tool List

- list of default tools from toolhelp :

```
IPy:Jupiter
CASA <8>: toolhelp()

Available tools:

 at : Juan Pardo ATM library
 cb : Calibration utilities
 cp : Cal solution plotting utilities
 fg : Flagging/Flag management utilities
 ia : Image analysis utilities
 im : Imaging utilities
 me : Measures utilities
 ms : MeasurementSet (MS) utilties
 mp : MS plotting (data (amp/phase) versus other quantities)
 tb : Table utilities (selection, extraction, etc)
 tp : Table plotting utilities
 qa : Quanta utilities
 sm : Simulation utilities
 vp : Voltage pattern/primary beam utilties
 ---
 pl : pylab functions (e.g., pl.title, etc)
 ---
```

- tools = set (state) + apply (process) methods
- tools described in the CASA Toolkit Reference Manual:
  - http://casa.nrao.edu/docs/CasaRef/CasaRef.html

How does this work in practice?

- There have been contributed tasks, e.g.
  - importevla (wrapping asdm2ms)
  - flagcmd (wrapping table and flagger)
  - boxit (autoboxing, wrapping images)
  - autoclean (using autoboxing and imager)
- Cool. How do I distribute/get stuff like this?
  - “insiders” : get CASA team to check into code base
  - “outsiders” : post somewhere (CASA Science Forum)
    - https://science.nrao.edu/forums/
  - “associates” : get put onto casaguides
    - http://casaguides.nrao.edu
  - future: better mechanism?

Discussion Points

- Future developments
  - Better support for Python programming (import)
  - Better support for C++ programming (plugins?)
  - Application (e.g. viewer) control (Qt, Dbus, blahblahblah)
  - Integrating interfaces (GUIs plus param setting)
  - Refactoring interfaces (meta-tasks? functional lang.?)
- User Support
  - RTFM. I spent time writing documentation. Why?
    - OK, how can we do this better? Or reduce the need?
  - Enabling transferral of knowledge/scripts
    - Forums? Wiki? Other?
  - Import of general Astro toolkit (e.g. astropy)
  - Import of CASA into other astro (e.g. LSST)
- Other? Programmatics? Consortia? Management?