Beyond CASA Tasks: The Toolkit, Python, & Scripting

Steven T. Myers
(NRAO-Socorro)
What is the CASA Toolkit?

• The CASA Toolkit is bound into Python as function methods. The CASA Tasks are built upon this toolkit, providing a simpler interface and workflow to carry out common data reduction activities.

• Why would I use a tool rather than a task?
  – some functionality not (yet?) available in a task
  – do something different than a task does
  – you only want to do a small part of what a task does
    • often significant gain in efficiency
  – direct access to data: MS, tables, text files
  – use of shell commands, access to shell variables, etc.
How do I find and use the CASA tools?

• Get a list of tools in CASA: `toolhelp`
• Help for a tool method:
  – in CASA: `<tool>..<method>?` (sometimes `<tool>?` is useful)
  – example: `im.makeimage?`
  – Also, tab completion works to give list of methods: `<tool>..<TAB>`
• Online documentation of toolkit:
    [http://casa.nrao.edu/docs/CasaRef/CasaRef.html](http://casa.nrao.edu/docs/CasaRef/CasaRef.html)
    (Warning: many of the examples and some of the documentation not fully up-to-date and reflect earlier versions of CASA. You may have to experiment a bit or ask CASA staff how to use these in some cases.)
• Other documentation: examples and advice for toolkit usage are sprinkled throughout the CASA User Reference & Cookbook:
  [http://casa.nrao.edu/docs/cookbook/index.html](http://casa.nrao.edu/docs/cookbook/index.html)
toolhelp

• in CASA 4.5.2
example <tool> <TAB> completion

CASA <62>: im.
im._class__
im._del__
im._delattr__
im._dict__
im._doc__
im._format__
im._getattr__
im._getattribute__
im._hash__
im._init__
im._module__
im._new__
im._reduce__
im._reduce_ex__
im._repr__
im._setattr__
im._sizeof__
im._str__
im._subclasshook__
im._swig_destroy__
im._swig_getmethods__
im._swig_setmethods__
im._weakref__
im._bmaj_fitpsf
im._bmin_fitpsf
im._bpa_fitpsf
im._cell_advise
im._facets_advise
im._phasecenter_advise
im._pixels_advise
im._pointsource_apparentsens
im._pointsource_sensitivity
im._relative_apparentsens
im._relative_sensitivity
im._senrec_sensitivity
im._sumweights_sensitivity
im.advice
im.advisechansel
im.apparentsens
im.approximatepsf
im.boxmask
im.calcuvw
im.clean
im.clipimage
im.clipvis
im.close
im.defineimage
im.done
im.drawmask
im.exprmask
im.feather
im.filter
im.fitpsf
im.fixvis
im.ft
im.getweightgrid
im.linearmosaic
im.make
im.makeimage
im.makemodelfromsd
im.mapextent
im.mask
im.mem
im.nnls
im.open
im.pb
im.plotsummary
im.plotuv
im.plotvis
im.plotweights
im.predictcomp
im.regionmask
im.regiontoimagemask
im.residual
im.restore
im.selectvis
im.sensitivity
im.setbeam
im.setjy
im.setmfccontrol
im.setoptions
im.setscales
im.setsdoptions
im.setsmallscalebias
im.settaylorterms
im.setvp
im.setweightgrid
im.smooth
im.ssosflux
im.stop
im.summary
im.this
im.updateresidual
im.uvrange
im.weight
example `<tool>..<method>`?

`CASA <63>: im.makeimage?`

Type: `instancemethod`

Base Class: `<type 'instancemethod'>`

String Form: `<bound method imager.makeimage of <__casac__.imager.imager; proxy of <Swig Obj ect of type 'casac::imager*' at 0x2c3eab0>>`

Namespace: `Interactive`

File: `/home/casa/packages/RHEL6/release/casa-release-4.5.2-e16/lib/python2.7/__casac __/imager.py`

Definition: `im.makeimage(self, *args, **kwargs)`

Docstring:

```
makeimage(self, type = string("observed"), image = string("""), compleximage = string("""),
         verbose = True, async = False) -> bool
```

Summary

Calculate images by gridding, etc.

Description

This tool function actually does gridding (and Fourier inversion if needed) of visibility data to make an image. It allows calculation of various types of image:

egin{description}
\item[observed] Make the dirty image from the DATA column (\{ default\})
\item[model] Make the dirty image from the MODEL\_DATA column
\item[corrected] Make the dirty image from the CORRECTED\_DATA column
\item[residual] Make the dirty image from the difference of the CORRECTED\_DATA and MODEL\_DATA columns
\item[psf] Make the point spread function
\item[singledish] Make a single dish image
\item[coverage] Make a single dish coverage image
\item[holography] Make a complex holography image
\item[pb] Make the primary beam as defined by setpb
d{description}

Note the full \{ t imager\} equation is not used and so, for example, the primary beam correction is not performed. Use restore to get a residual image using the full \{ t imager\} equation where primary beam correction is performed.

A position shift can be applied when specifying the image parameters with defineimage. If a shift is specified then the uuv coordinates are reprojected prior to gridding, and a phase
Key CASA tools

- The most used tools (and some example methods) include:
  - Imager (im) : synthesis imaging
    - `im.makeimage`, `im.make`, `im.clean`, `im.setdata`, `im.defineimage`
  - Images (ia) : analysis of and access to images
    - `ia.statistics`, `ia.coordsys`, `ia.shape`
  - Measurement Set (ms) and Tables (tb) : access to MS & table data
    - `ms.getdata`, `ms.putdata`, `tb.getcol`, `tb.putcol`
  - Quanta (qa) and Measures (me) : manipulation of values
    - `qa.quantity`, `me.direction`
  - Pylab (pl) : access to Pylab, numpy, matplotlib, etc.
    - `pl.plot`, `pl.array`, `pl.median`
  - utilities like CASA log (casalog), Python modules (os, time) etc.
    - `casalog.post`, `casalog.version`, `os.system`, `time.time`
General Tool Advice

• use of tools often involves an ordered use of methods
  – `.open` (see below) to indicate what dataset or file to access
    • e.g. `im.open('my.ms') , ia.open('myimage.im')`
  – methods to set parameters, define things, before doing other things
    • e.g. `im.selectvis , im.setoptions , im.defineimage`
• some methods have return “values” which can be
  – Python tuple
  – CASA (Python) record
  – Numpy array
  – tool method
• again, the tool documentation is of variable quantity and quality 😞
  – may be out of date (pre-Python in some cases) or incomplete
  – you may need to experiment or ask for help

DON’T PANIC!
Tool Example: `imager`

- **Goal**: make a dirty image
  - **Why?**: running clean task with niter=0 is inefficient as it does a bunch of stuff expecting you to want to later clean the image!!! This gives a many times speedup (especially on small images)

```
sandrock<180>% cat makemyimage.py
im.open(msname)
im.defineimage(cellx=cell, celly=cell, nx=imsizex, ny=imsizex, stokes='I')
im.selectvis(field=field, time=timerange)
im.makeimage(type=usedatacolumn,image=outim+'.image')
im.close()
sandrock<181>%
```
Tool Example: imager

- Goal: make a primary beam image
  - Why? you sometimes want to use the primary beam without doing a large clean to make it.

```
sandrock<181>% cat makemypb.py
im.open(visname)
im.selectvis(field=ifld)
im.defineimage(cellx=mycell, celly=mycell, nx=myfldimsize, ny=myfldimsize, stokes='I', phasecenter=ifld)
im.setvp(dovp=True, usedefaultvp=True, dosquint=False)
im.makeimage(type='pb', image=pbimg)
im.close()
sandrock<182>%
```
Tool Example: imager & images

- Goal: make a mask image for clean from a file with regions
  - Why? if you have a set of regions (clean boxes, source locations) some of which may be outside the image clean will make, you need to make an image mask with those only within the image.

```
sandrock<188>$ cat makemaskimage.py
#
# make a blank image mask based on image
maskimage = fldimname+'.'mask'
in.open(visname)
in.selectvis(field=fld,spw=[ispw])
in.defineimage(cellx=mycell, celly=mycell, nx=myfldimsize, ny=myfldimsize, stokes='I',phasecenter=fld,spw=[ispw])
in.make(maskimage)
#
# Extract coordsys and shape of image
ia.open(maskimage)
ncoordsys=ia.coordsys()
inshape=ia.shape()
ia.close()
#
# Extract regions from text file containing regions
myreg=rg.fromtextfile(filename=myregionfilename,csys=imcoordsys.torecord(),shape=inshape)
#
# Convert to image mask
in.regionstomagemask(maskimage,region=myreg)
in.close()
#
# Can now clean using this mask, e.g.
# clean(vis=visname,imagename=fldimname,field=fld,
#      imsize=[myfldimsize],cell=[mycell],stokes='I',
#      mask=maskimage,
# ...     
sandrock<189>$
```
CASA Python Scripting – Why?

• a durable and executable record of your processing
  – should contain annotation!
• transportable and distributable
  – can send to your colleagues or post online
• efficient for long reduction sequences
  – datasets often too large to “archive” by the user
• reproducible (for a given version of CASA)
  – important for debugging and error finding
• build up a custom library of useful functions and tasks
  – importable, reusable, tradeable
• session logs: casapy-<…>.log and ipython-<…>.log not complete
Python

• Many online (and book) sources for information
  – Easy for users to “build their own” scripts/tasks

• Public Documentation:
  – Python:
    http://python.org/doc (e.g. Tutorial for novices)
  – IPython:
    http://ipython.org
  – matplotlib:
    http://matplotlib.sourceforge.net/

• CASA-specific Documentation:
  – Casaguides:
    http://casaguides.nrao.edu (Getting Started in CASA)
  – CASA User Reference and Cookbook: (Appendix B)
    http://casa.nrao.edu/docs/UserMan/UserMan.html

Also: astropy
http://www.astropy.org
Python Basics

• Setting variables:
  – Assignment `<parameter> = <value>`
  – Testing `<parameter> == <value>` (or `>`, `<`, `>=`, `<=`)
  – Tasks use a “standard” set of global variables
    • Watch out for mis-spellings (e.g. correlation vs. corellation), you just create a new variable

• Lists:
  – Assignment: `antlist = ['ea04', 'ea05', 'ea13']`
  – Append: `antlist.append('ea28')`
  – 0-based indices: `antlist[0]` (returns value ‘ea04’)
Strings split operator (and tuples)

```python
# break string into key=val sets
keyvlist = cmdstr.split()
if keyvlist.__len__()>0:
    for keyv in keyvlist:
        (xkey,val) = keyv.split('=')
```

File creation and access

```python
logfile=open(outfile,'w')
```

Output: print

```python
print '  Cleaning MFS continuum image SPW '+spw+' '+instokes
print '  Field %s  P/I = %10.4f  RLPD = %8.3f deg' % (infield,mflx,rlpd)
print >>logfile,'  Field %s  P/I = %10.4f  RLPD = %8.3f deg' % (infield,mflx,rlpd)
```
Python Basics – Ranges, Loops

• Range function
  – Assignment `antlist = range(4,8)`
  • Equivalent to `antlist = [4, 5, 6, 7]`

• Blocks, Loops, and Indentation:
  – Indentation matters, sets apart levels of blocks/loops
  – Conditional blocks: `if-elif-else`
  – Loops: `for`, `while`
    • `for i in range(5)`
    • `for ant in antlist`
    • `while <boolean>`
Example Script – Conditional, Assignment

• Conditional blocks: if – elif – else

if obsconfig=='C':
    # C-config FOV/beam = 128
    myimsize = 400
if obsband=='L':
    # L-band beam 30' FOV at 1.5 GHz
    # C-config resolution: 1.5 GHz = 14"
    mycell = '4.0arcsec'
elif obsband=='S':
    # S-band beam 15' FOV at 3 GHz
    # C-config resolution: 3 GHz = 7"
    mycell = '2.0arcsec'
else:
    print 'ERROR: unknown band '+obsband
Example Script - Loop

• Loops (with some string construction thrown in)

    for field in myfieldlist:
        splitfile = prefix + '.field' + field + '.split.ms'
        outputvis = splitfile
        saveinputs(taskname,splitfile+'.saved')
        print ' Splitting field '+field+' to '+splitfile
        split()
Python Dictionaries

• A nested associative (hashed) list of \{ <key> : <value> \}

```python
polname = 'J1331+3030'
polsrc = {}
polsrc[polname] = {}
polsrc[polname][\'0\'] = \{ 'I' : 14.61, 'F' : 0.094, 'X' : 66.0 \}
polsrc[polname][\'1\'] = \{ 'I' : 13.09, 'F' : 0.094, 'X' : 66.0 \}
```

• Access

```python
polsrc[polname][\'0\'][\'I\']
```

Dictionaries can be saved to files using pickle module:

```python
import pickle
...
```

You can also use text files containing dictionary, e.g.

```python
polsrc = \{\'0\': \{\'I\': 14.61,...\},...\}
```
Python / IPython Basics

- Toolkit return values (and some math)
  
  ```python
  if instokes.count('QU')>0:
    qval = imval(mfsimage,stokes='Q')
    uval = imval(mfsimage,stokes='U')
    qflx = qval['data'][0]  # access element of numpy array
    uflx = uval['data'][0]  # access element of numpy array
    rlpd = atan2(uflx,qflx)*180.0/pi
    pflx = sqrt(qflx*qflx + uflx*uflx)
  ```

- Exception handling: try, except  (catch stuff that fails)
  
  ```python
  try:
    gaincal()
  except:
    print 'ERROR: aborting script'
  raise
  ```
**Useful Python modules**

- **pickle** – read/write python dictionaries
- **os**
  - `os.system()` # execute shell commands
  - `os.access()` # test existence of file and directories etc.
  - `os.getenv()` # get value of environment variables
- **time**
  - `time.time()` # return time, use for benchmarking
- **datetime**
  - `datetime.today()` # todays date
  - `datetime.isoformat()` # turn date into string
- **xml.dom** – read from xml file (others xml modules exist also)
Example Function - getfieldcone

- See: [http://www.aoc.nrao.edu/~smyers/casa/scripts/4.5.2/getfieldcone.py](http://www.aoc.nrao.edu/~smyers/casa/scripts/4.5.2/getfieldcone.py)

- This example function returns a list of field IDs within a distance (radius) of an input sky location.

- Notes about functions
  - arguments are passed by reference (more or less)
  - if a variable, changes inside function will not be passed back
  - if a list or dictionary, changes will be made to original array
  - normally you return things in return variables
Example Function - getfieldcone

See: http://www.aoc.nrao.edu/~smyers/casa/scripts/4.5.2/getfieldcone.py

CASA Toolkit - Data Reduction Workshop Mar 2016

```
sandrock(146)% casa

=======================================================================
The start-up time of CASA may vary depending on whether the shared libraries are cached or not.
=======================================================================

CASA Version 4.5.2-REL (r36115)
  Compiled on: Wed 2016/02/10 13:31:05 UTC

For help use the following commands:
tasklist    - Task list organized by category
taskhelp    - One line summary of available tasks
help taskname - Full help for task
toolhelp    - One line summary of available tools
help par.parametername - Full help for parameter name

Activating auto-logging. Current session state plus future input saved.
Filename    : ipython-20160314-183715.log
Mode        : backup
Output logging : False
Raw input log : False
Timestamping  : False
State        : active
*** Loading ATNF ASAP Package...
*** ... ASAP (4.3.0a rev#34723) import complete ***

CASA <2>: execfile('getfieldcone.py')

CASA <3>: myms = 'TSKY0001_M31_1_sb30647879_57149_calibrated_target.ms'

CASA <4>: mydist = '0.25deg'

CASA <5>: mycen = 'J2000 00:41:41.249 41.03.33.26'

CASA <6>: myflds = getfieldcone(myms,distance=mydist,center=mycen)
  Found 1280 fields
  Cataloged 1280 fields
  Using center direction J2000 00:41:41.249 41.03.33.26
  Looking for fields with maximum separation 0.25 deg
  Found 14 fields within 0.25 degrees

CASA <7>: myflds
  Out[7]: [676, 677, 678, 679, 760, 761, 762, 763, 836, 837, 838, 839, 921, 922]

CASA <8>:
```
Example Function -

```python
def getfieldcone(msfile=None, distance='0deg', center=' '):
    # Return a list of fields with distance of center
    # Example:
    # mycenter = 'J2000 10:00:23.6 2.12.21'
    # myflds = getfieldcone(mys, distance='10deg', center=mycenter)
    #
    fieldlist = []
    try:
        qdist = qa.toangle(distance)
        qdeg = qa.convert(qdist, 'deg')
        maxdeg = qdeg['value']
    except:
        print 'ERROR: cannot parse distance ', distance
        return
    try:
        tb.open(msfile+'/FIELD')
    except:
        print 'ERROR: could not open '+msfile+'/FIELD'
        return
    field_dirs=tb.getcol('PHASE_DIR')
    field_names=tb.getcol('NAME')
    fdir = me.direction('J2000', ral[0], decl[0])
    for i in range(nf):
        fra = field dirs[0,0,i]
        fdd = field dirs[1,0,i]
        rapos = qa.quantity(fra,'rad')
        decpos = qa.quantity(fdd,'rad')
        ral = qa.angle(rapos, form=['tim'], prec=9)
        decl = qa.angle(decpos, prec=10)
        if fdir in field_names[i] and 'name'=fn:
            if flookup.has_key(fn):
                flookup[fn].append(i)
            else:
                flookup[fn]=i
            print 'Cataloged '+str(nf)+ ' fields'
```
Example Function - `getfieldcone`

- See: [http://www.aoc.nrao.edu/~smyers/casa/scripts/4.5.2/getfieldcone.py](http://www.aoc.nrao.edu/~smyers/casa/scripts/4.5.2/getfieldcone.py)
Example Function - getfieldcone

- See: http://www.aoc.nrao.edu/~smyers/casa/scripts/4.5.2/getfieldcone.py

```python
# Construct offset separations
print 'Looking for fields with maximum separation '+str(maxdeg)+' deg'
for i in range(nf):
    dd = ddirs[i]['dir']
    sep = me.separation(cdir,dd)
    sepdeg = qa.convert(sep,'deg')
    offs = sepdeg['value']
    ddirs[i]['offset']=offs
    if offs<=maxdeg:
        fieldlist.append(i)
print 'Found '+str(len(fieldlist))+' fields within '+str(maxdeg)+' degrees'
# return fieldlist
(END)
```
Build your own CASA task

• How?
  – 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
Building Tasks - Documentation

- **Documentation:**
  - CASA User Reference and Cookbook:
    [http://casa.nrao.edu/ref_cookbook.html](http://casa.nrao.edu/ref_cookbook.html) (Appendix J)
  - Casaguides:
    [http://casaguides.nrao.edu](http://casaguides.nrao.edu) (Writing a CASA Task)

CASA Documentation Summary

• Homepage: http://casa.nrao.edu → Using CASA

• CASA Reference Manual & Cookbook:
  ☑ http://casa.nrao.edu/docs/UserMan/UserMan.html

• CASA Task Reference (same as inline help):
  ☑ http://casa.nrao.edu/docs/TaskRef/TaskRef.html

• CASA Toolkit Manual:
  ☑ http://casa.nrao.edu/docs/casaref/CasaRef.html

• CASAguides Wiki:
  ☑ http://casaguides.nrao.edu

• Python:
  ☑ http://python.org/doc (e.g., see Tutorial for novices)

• IPython:
  ☑ http://ipython.org

• matplotlib:
  ☑ http://matplotlib.sourceforge.net/

Also: astropy
http://www.astropy.org