Advanced CASA

Joshua Marvil

8th VLA Data Reduction Workshop

Introduction

This presentation will provide an in-depth look at how CASA works behind the scenes

By achieving a better understanding of CASA, it is my hope that you will be able to:

- use interactive CASA more efficiently
- access advanced CASA functionality
- directly read and write CASA data products
- integrate CASA with your own Python code

Introduction

The following code examples are intended for use with CASA version 6.1.2-7

I will describe each example and show the example's output when appropriate

I suggest that you try out these examples after the talk and do not try to follow along interactively

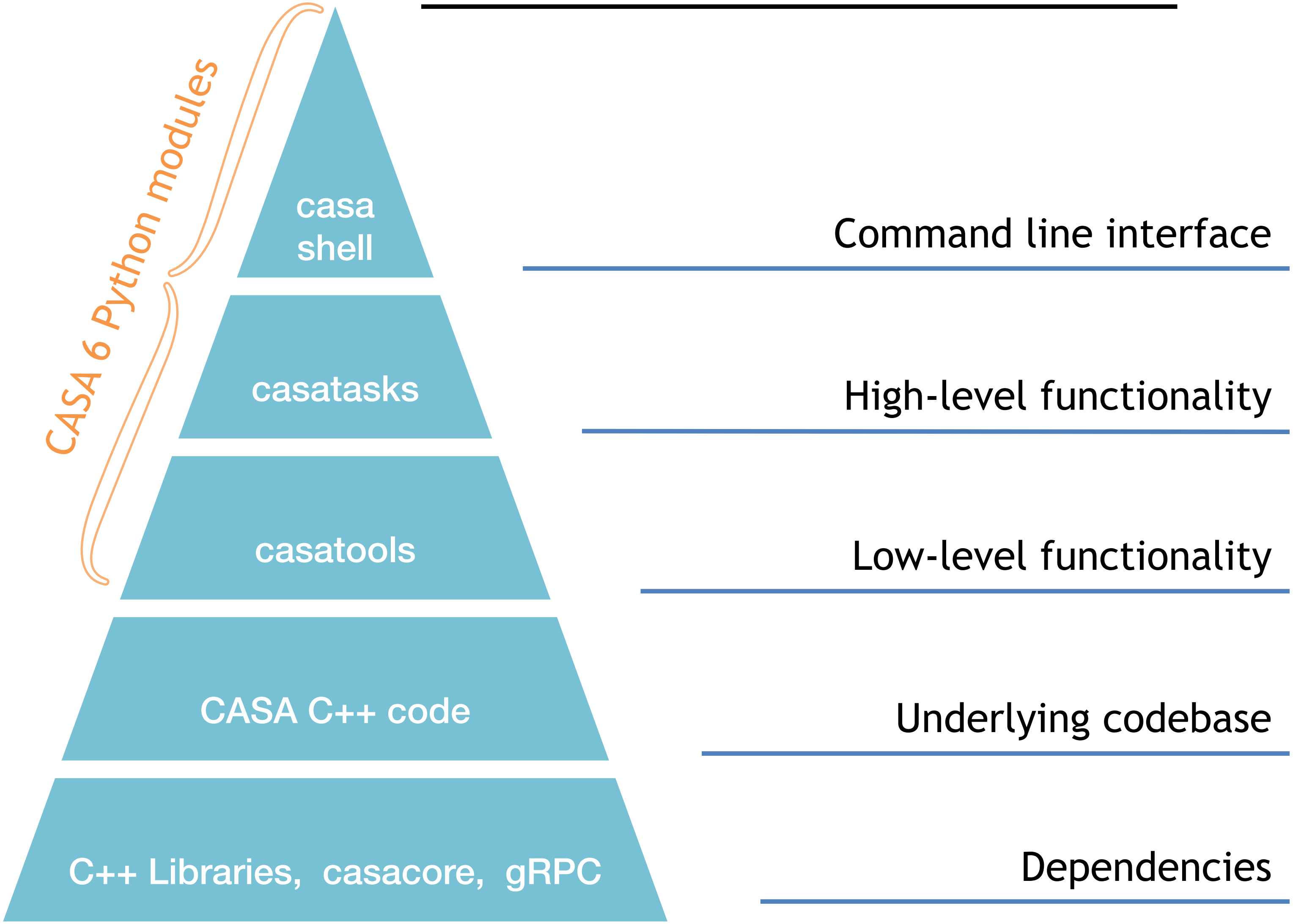
Outline

Advanced CASA overview

The CASA toolkit

CASA scripts and custom tasks

How CASA is Structured



CASA 6 Python modules

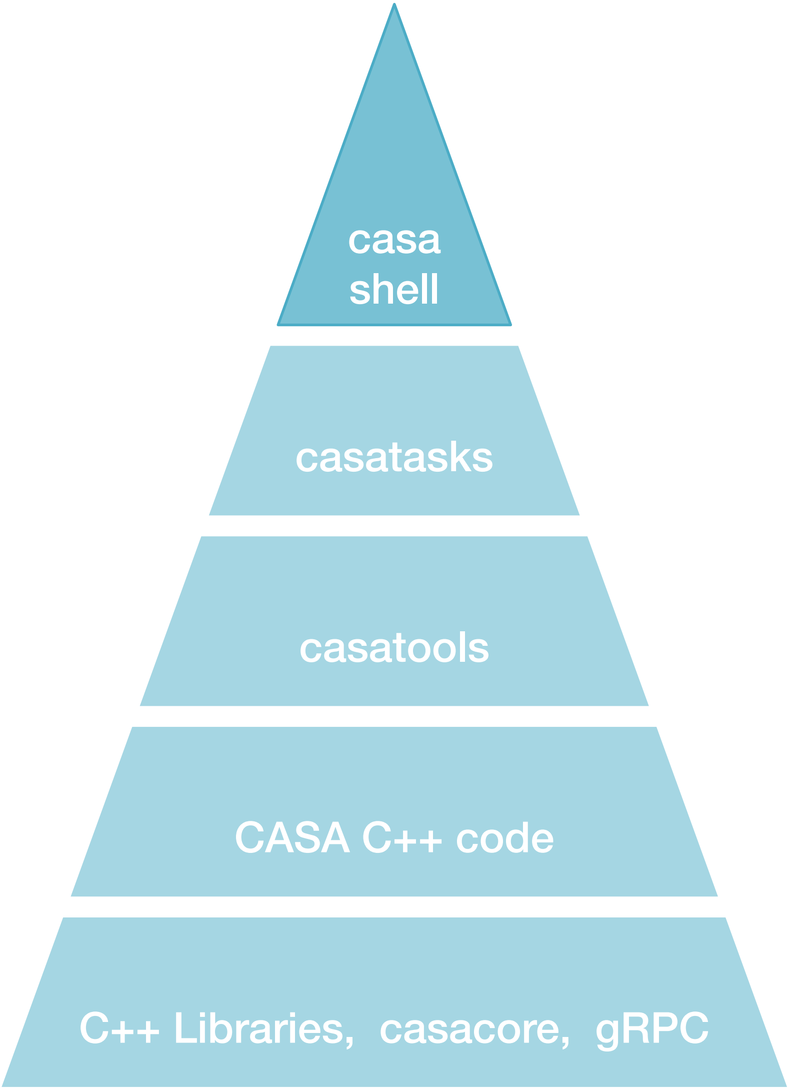
casa shell — Command line interface

casatasks — High-level functionality

casatools — Low-level functionality

CASA C++ code — Underlying codebase

C++ Libraries, casacore, gRPC — Dependencies



casashell

casashell is what starts when you run 'casa'

CASA's interactive shell is a customized Ipython environment

Additional python functions create the task interface

casashell: the Command Line Interface (CLI)

Interactive Shell

- Python interpreter
- standard library
- Python modules
- iPython extensions
- CASA modules
- startup scripts

Task system

- default
- inp
- go
- tget
- tput
- tasklist
- taskhelp

Python (3.6)

Python objects, (e.g., int, float, bool, str, list, tuple, set, dict), and their methods

Other programming elements, e.g., operators, expressions, statements, syntax

Built-in functions, e.g., help, open, print, format, eval, exec, type, input

Python standard library

All standard modules are available in CASA

E.g., os, sys, re, glob, shutil, pickle, time, urllib

Full list of Python 3.6 standard library:
docs.python.org/3.6/library

```
# In CASA

import os

os.path.isdir( '3C75.ms' )
```

Additional Python modules

Several additional Python modules are available in CASA

Details are platform and version specific, but some version of the following are typically available:

numpy, scipy, matplotlib, dateutil, pytz, pyfits

```
# In CASA

import matplotlib.pyplot as plt

import numpy as np

plt.plot( np.random.randn(42) )
```

IPython magic commands

IPython has many enhancements over the standard interpreter

Designed for interactive use— may not work in scripts

Magic commands, preceded by %, are one such enhancement

Others include completion <tab>, introspection ?, shell access !

```
# In CASA

%lsmagic
```

IPython magic commands

Notable magic commands:

automagic: % not required -- on by default
autocall: () not required -- on by default
pprint: pretty printing -- on by default
cpaste: paste an entire cell (block of commands)
history: view command line history
run: execute a script (similar to Python 2's execfile)

Example of autocall in CASA:

```
CASA <1>: help tclean
---------> help(tclean)
```

IPython system shell access

Shell commands can be executed with !

Output can be returned as a Python variable

If your desired system command is not found, check the path that CASA is using (e.g., %env PATH or os.environ['PATH'])

```
# In CASA

!du -hs *

hostname = !echo $HOSTNAME

print( hostname )
```

The IPython alias command

%alias is an IPython magic command that defines an alias to a system shell command

An alias is treated like a new magic command

Aliases have lower precedence than magic functions and normal Python variables

```
# In CASA

alias

alias du du -hs

du *
```

Further customization

CASA can be further customized by creating a file ***startup.py*** in the $HOME/.casa directory

You can put commands here that will run every time CASA starts up, e.g., import modules, set variables, modify the PATH

You can also customize many aspects of IPython using ***startup.py***, e.g., the color scheme, magic commands, aliases, behavior of extensions

CASA Tasking System

Tasking system

- default
- inp
- go
- tget
- tput
- tasklist
- taskhelp

Python functions that facilitate setting up and running tasks

Task parameters are variables in the global namespace

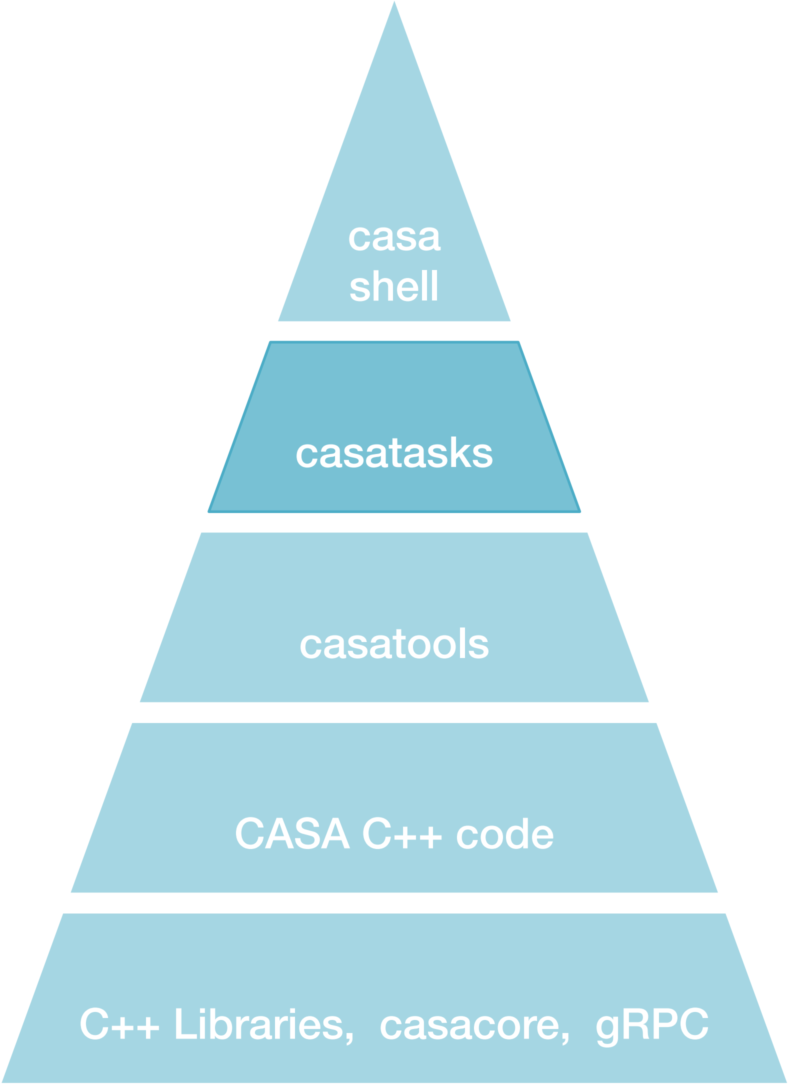
Together with %autocall, they give CASA a look and feel that is more user friendly and less 'Pythonic'

CASA Tasking System

```
# In CASA

go?
```

example output

```
Signature: go(obj=None)
Docstring: <no docstring>
File:      /home/casa/packages/pipeline/casa-6.1.2-7-
pipeline-2020.1.0.36/lib/py/lib/python3.6/site-packages/
casashell/private/init_subparam.py
Type:      function
```



CASA Tasks

Task interface (XML)

- parameters
- description
- defaults
- expansions
- validation

Task script (Python)

- CASA toolkit
- logging
- history
- exceptions
- return values

CASA task XML

Each CASA task has an associated XML file

These are used by casashell when running the task interactively

Files are named '***taskname*.xml**' and are located in a folder inside the CASA distribution

```
# In CASA

xml_path = casatasks.__path__[0]+'/__XML__'

print( xml_path )

ls $xml_path
```

CASA task 'scripts'

Each CASA task executes a Python script

It can be informative to read some of these scripts

Scripts are named '**task_*taskname*.py**' and are located in a folder inside the CASA distribution

```
# In CASA

task_path = casatasks.__path__[0]+'/private'

print( task_path )

ls $task_path
```

Initializing CASA tasks

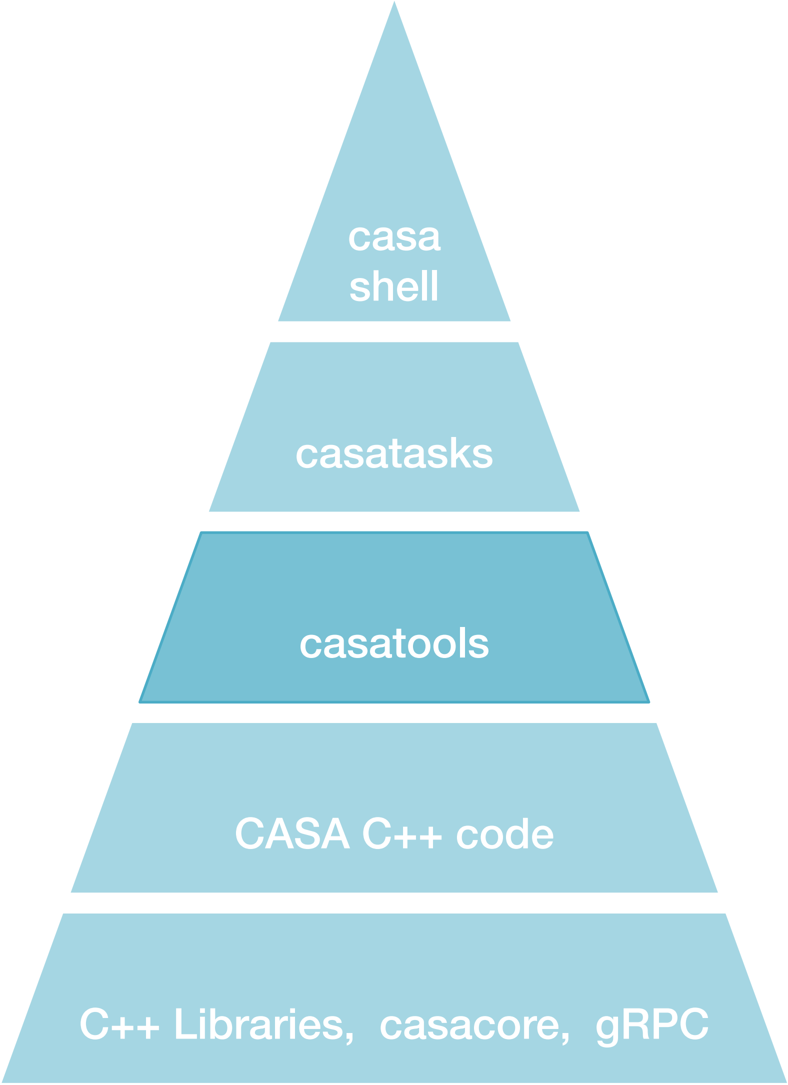
The full CASA distribution will initialize tasks during startup

If using the casatasks python module, you need to import them

```
# In Python

from casatasks import tclean

tclean( vis=...
```

or

```
import casatasks

casatasks.tclean( vis=...
```



CASA Tools

Most tools are Python interfaces to the C++ code using Simplified Wrapper and Interface Generator (SWIG)

Tool methods are more atomic than tasks; many tasks consist of calls to several tool methods

Tasks are meant to capture common use cases and be simple to use

Tools offer additional flexibility and functionality over tasks

Comparison: task vs. tool

imstat task

```
# In CASA

image = '3C75_initial.image.tt0'

imstat( image )
```

equivalent toolkit method (inside ***task_imstat.py***)

```
# In CASA

ia.open( image )

ia.statistics( robust=True )

ia.close()
```

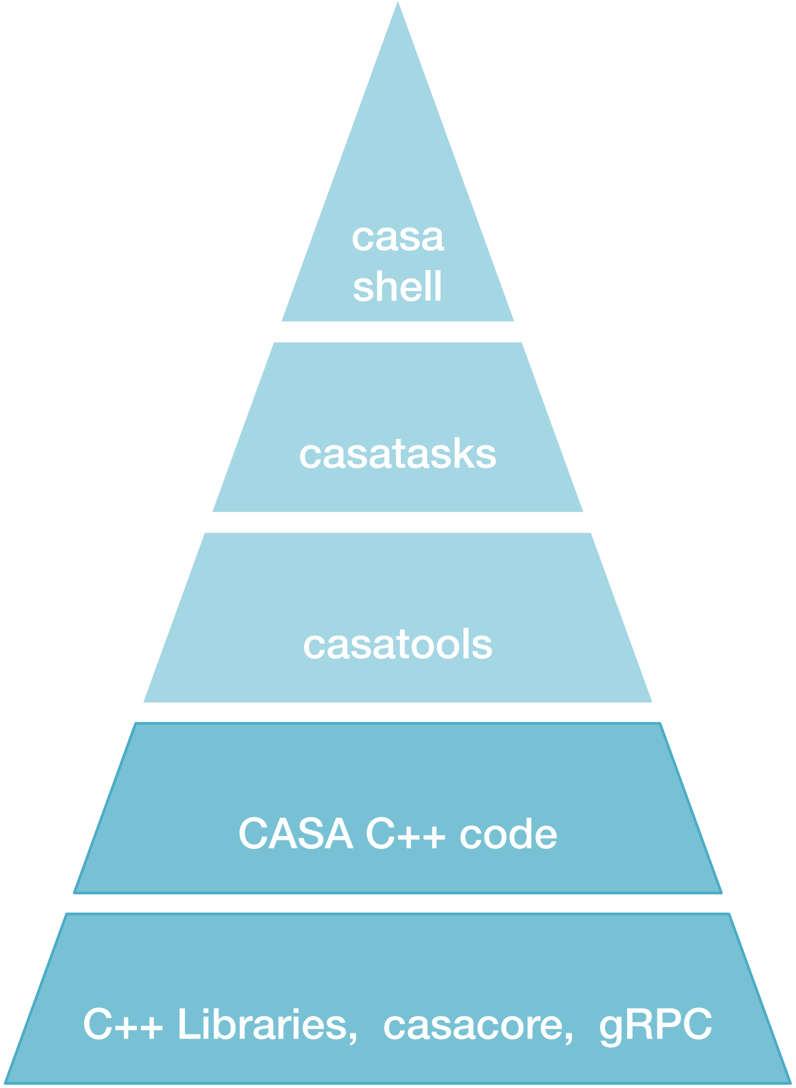
Initializing CASA tasks

The full CASA distribution initializes these tools during startup

If using the casatools python module, you need to import and instantiate them

```
# In Python

from casatools import image

ia = image()

ia.open( imagename=...
```

use toolhelp or see table here:
casa.nrao.edu/casadocs/casa-6.1.0/usingcasa/obtaining-and-installing



casa shell

casatasks

casatools

CASA C++ code

C++ Libraries, casacore, gRPC

CASA C++

Most of CASA is written in C++ for performance reasons

CASA is built on top of the casacore libraries (an independent project)

Other CASA C++ dependencies include: gsl, blas, lapack, atlas, cfitsio, wcslib, fftw

gRPC is used to facilitate communication with GUIs

Outline

Advanced CASA overview

The CASA toolkit

CASA scripts and custom tasks

CASA Tools (partial list)

af : Agent flagger
at : Atmospheric library
cb : Calibration
cl : Component list
cs : Coordinate system
ia : Image analysis
im : Imaging
me : Measures
ms : MeasurementSet
msmd : MS metadata

mt : MS transformer
qa : Quanta
po : Imagepol
rg : Region manipulation
sdms : Single-Dish MS
sl : Spectral line catalog
sm : Simulation
tb : Table
vp : Voltage pattern

(also PySynthesis : tclean)

quanta tool

Values with units, unit conversion, string formatting

```
# In CASA

angle = qa.quantity( '1rad' )

print( angle )
```

example output

```
{'unit': 'rad', 'value': 1.0}
```

quanta tool

Values with units, unit conversion, string formatting

```
# In CASA

qa.convert( angle, 'arcsec' )

qa.time( angle, prec=10 )
```

example output

```
{'unit': 'arcsec', 'value': 206264.80624709636}

['03:49:10.9871']
```

measures tool

Reference frames, directions, conversions, measurements

```
# In CASA

dir1 = me.direction('J2000','06:18:00','+41.00.00')

dir1
```

example output (using pprint)

```
{'m0': {'value': 1.6493361431346414, 'unit': 'rad'},
 'm1': {'value': 0.7155849933176751, 'unit': 'rad'},
 'refer': 'J2000',
 'type': 'direction'}
```

measures tool

Reference frames, directions, conversions, measurements

```
# In CASA

me.shift( dir1, '3arcmin', pa='45deg' )
```

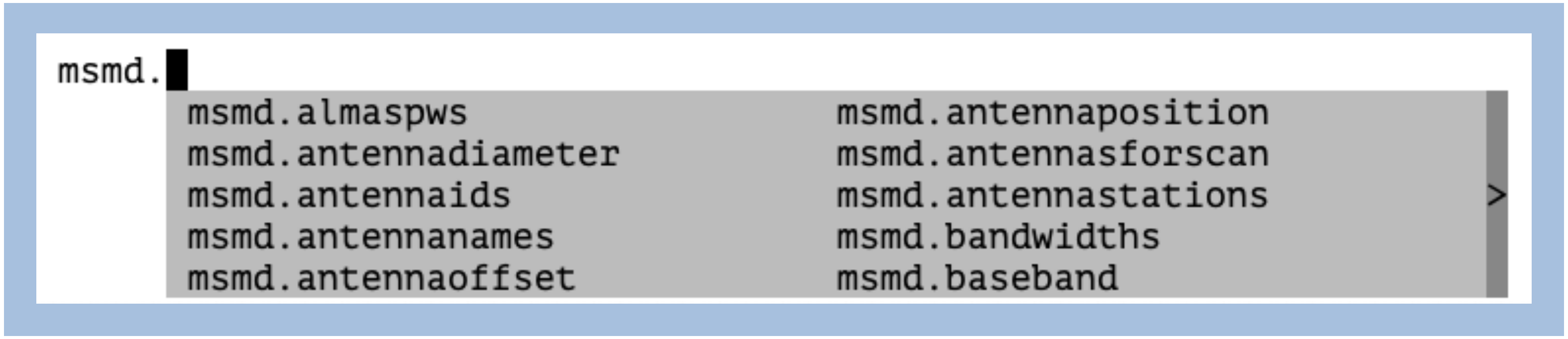
```
# In CASA

me.doframe( me.observatory('VLA') )

me.doframe( me.epoch('utc','today') )

me.measure( dir1, 'azel' )
```

MS metadata tool

Helper functions to retrieve measurement set properties

```
# In CASA

vis = '3C75.ms'

msmd.open( vis )

msmd.<TAB>    (browse functions with TAB or ↑ ↓, select with RETURN)
```

example output



```
msmd.
      msmd.almaspws                msmd.antennaposition
      msmd.antennadiameter         msmd.antennasforscan
      msmd.antennaids              msmd.antennastations        >
      msmd.antennanames            msmd.bandwidths
      msmd.antennaoffset           msmd.baseband
```

MS metadata tool

Helper functions to retrieve measurement set properties

```
# In CASA

fields = msmd.fieldnames()

print( fields )

msmd.close()
```

example output

```
['3C75']
```

measurement set tool

Measurement set access and manipulation

```
# In CASA

ms.open( vis )

results = ms.getdata( ['time','uvw'] )

ms.close()
```

Note: this will try to retrieve data from all rows of the MS

Additional selection and / or iteration may be required for large data sets to avoid memory issues

measurement set tool

Measurement set access and manipulation

```
# In CASA

type(results)

results.keys()

results['uvw'].shape
```

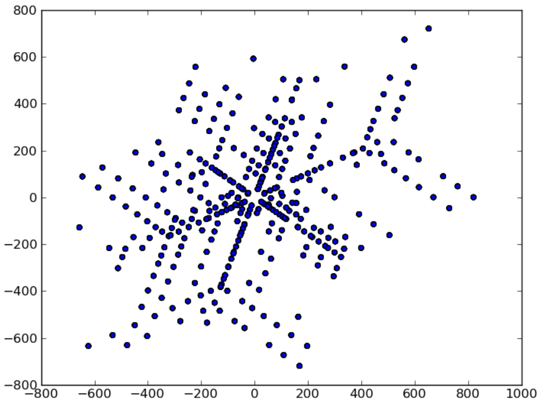
example output

```
dict

['uvw', 'time']

(3, 2808)
```

measurement set tool

Measurement set access and manipulation

```
# In CASA

u,v = results['uvw'][:2]

plt.scatter( u, v )

plt.scatter( -u, -v )
```



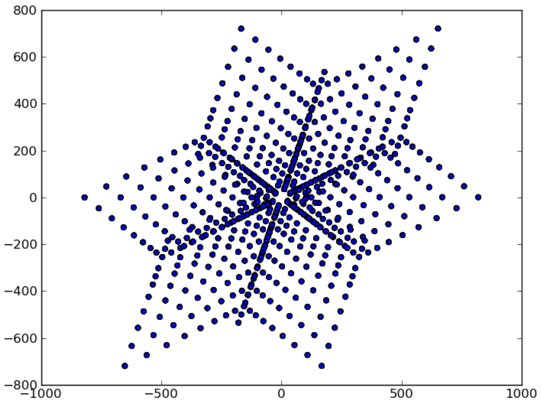


image analysis tool

Image inspection and manipulation

```
# In CASA

ia.open( image )

image_data = ia.getchunk()

ia.close()

image_data.shape

np.max( image_data )
```

table tool

Read, write and filter CASA tables. Works on measurement sets, ancillary tables of the MS, calibration tables, images, component lists, etc.

```
# In CASA

tb.open( vis+'/ANTENNA' )

tb.colnames()

tb.getcol( 'NAME' )

stb = tb.query( "NAME == 'ea12'" )

stb.getcol( 'POSITION' )

stb.done()

tb.close()
```

PySynthesisImager

Tool-like library that interfaces with C++ imaging routines.

See 'task_tclean.py' for a complete example.

```
from imagerhelpers.imager_base import PySynthesisImager

imager = PySynthesisImager( params )

imager.initializeImagers()
imager.initializeNormalizers()
imager.setWeighting()
imager.initializeDeconvolvers()
imager.initializeIterationControl()

imager.makePSF()
imager.makePB()
```

PySynthesisImager

Tool-like library that interfaces with C++ imaging routines.

See 'task_tclean.py' for a complete example.

```
imager.runMajorCycle()

while ( not imager.hasConverged() ):

    imager.runMinorCycle()
    imager.runMajorCycle()
    imager.updateMask()


imager.predictModel()
imager.restoreImages()
imager.pbcorImages()
```

Outline

Advanced CASA overview

The CASA toolkit

CASA scripts and custom tasks

Getting started with CASA scripts

A CASA script is just a file that contains a sequence of commands, e.g., tasks, tools, general Python

Name your script almost anything you want, e.g. myScript.py

Run your script in CASA (several options):

```
CASA>>  execfile( 'myScript.py' )

CASA>>  run -i 'myScript.py'

CASA>>  exec(open('myfile.py').read())
```

Run your script from the terminal:

```
bash$  casa -c myScript.py
```

Why write a script?

Reproducible - an executable record of your processing

Efficient - launch a sequence of several commands

Inspectable - easy to edit, expand, debug

Shareable - distribute to colleagues / helpdesk / etc.

Transportable - run on different resources

Generalizable - compatible with other similar datasets

Tips for CASA scripts

It is generally advised to run CASA tasks as functions

Use Python fundamentals where appropriate,
e.g., variables, loops, conditionals, exception handling

Learn from the examples in the documentation,
e.g., casadocs, casaguides, the toolkit reference manual

Avoid Ipython magic commands, e.g., autocall

Use a persistent session when running remotely,
e.g., vnc, screen, nohup

Work with a Python-aware text editor
-block indent, block comment, syntax highlighting

Example CASA scripts

Example Script: G55.7 CASAguides tutorial

```
setjy(vis='G55.7+3.4_10s.ms', field='0542*',
      spw='2~3,5~6', modimage='3C147_L.im')


gaincal(vis='G55.6+3.4_10s.ms', field='0542*',
        spw='2~3,5~6', caltable='G55.6+3.4_10s.G0',
        solint='int', calmode='p')
```

Example CASA scripts

Example Script: G55.7 CASAguides tutorial

```
vis = 'G55.7+3.4_10s.ms'
field = '0542*'
spw = '2~3,5~6'


setjy(vis=vis, field=field, spw=spw,
      modimage= '3C147_L.im')

gaincal(vis=vis, field=field, spw=spw,
        caltable=vis.replace('.ms','.G0'),
        solint='int', calmode='p')
```

Example CASA scripts

For loops:

```
spws = ['2','3','4','5']

for spw in spws:
    tclean( spw=spw, imagename='image_spw'+spw ... )
```

Flow control:

```
do_polcal = True

if do_polcal:
    polcal( ... )
```

3rd Party tasks and tools

There are many CASA tasks and tools written by members of the user community that can be imported into CASA

These are available from various locations, e.g., github, observatory websites, personal websites

Many of these are listed on the following CASAguide page:

casaguides.nrao.edu/index.php/Contributed_CASA_Tasks_and_Scripts

Writing your own CASA tasks

You can turn your own code into a CASA task in 3 easy steps:

- Create your own ***task_taskname.py***
- Create a xml file ***taskname.xml***
- Run !***buildmytasks*** to create **taskname.py**

Then you can import and use your new task!

More info here:

casadocs.readthedocs.io/en/latest/
api/casashell/buildmytasks.html

Documentation

Python: docs.python.org/3.6/

iPython: ipython.readthedocs.io/en/7.15.0/

CASA Guides: casaguides.nrao.edu

CASA Docs: casa.nrao.edu/casadocs

CASA Toolkit: casa.nrao.edu/docs/CasaRef/CasaRef.html
or casadocs.readthedocs.io/en/latest/api/casatools.html

Measurement Set: casa.nrao.edu/Memos/229.html

Table Query Language: casa.nrao.edu/aips2_docs/notes/199

www.nrao.edu
science.nrao.edu
public.nrao.edu

The National Radio Astronomy Observatory is a facility of the National Science Foundation
operated under cooperative agreement by Associated Universities, Inc.