CASA Toolkit Reference Manual

CASA Group, eds

October 4, 2016
The CASA Reference Manual contains the documentation on the tool-based functionality within the system. There are five broad packages:

- **General** - modules that are of general use for astronomical processing
  - **Images** - create, manipulate and analyze images
  - **Coordsys** - functionality for manipulating coordinate systems
  - **Componentlist** - functionality for manipulating components
  - **MeasurementSet** - functionality for manipulating MeasurementSets (CASA data structures)
  - **Measures** - functionality for handling quantities with a specified reference frame
  - **Quanta** - functionality for handling quantities (values with units)

- **Synthesis** - modules needed for processing synthesis data
  - **Calibrator** - synthesis calibration facilities
  - **AgentFlagger** - flagging framework for manual or automatic flagging
  - **Imager** - synthesis and single dish imaging including deconvolution
  - **Simulator** - facilities for simulation of telescope data
  - **vpmanager** - facilities for specifying voltage patterns and primary beams

- **Utility** - non-astronomy specific functionality
  - **Table** - functionality for manipulating tables in CASA

- **Third Party** - modules that interface to 3rd party packages
  - **Atmosphere** - Interface to Juan R. Padro’s Atmospheric Transmission Model (ATM) library.

- **Single-dish** - functionality for processing single-dish data
  - **ASAP** - single-dish data analysis package
## Contents

### 1 Package General 32

#### 1.1 images - Module 32

##### 1.1.1 image - Tool 35

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>image.newimage - Function</td>
<td>49</td>
</tr>
<tr>
<td>image.newimagefromfile - Function</td>
<td>50</td>
</tr>
<tr>
<td>image.imagecalc - Function</td>
<td>52</td>
</tr>
<tr>
<td>image.collapse - Function</td>
<td>56</td>
</tr>
<tr>
<td>image.decimate - Function</td>
<td>60</td>
</tr>
<tr>
<td>image.imageconcat - Function</td>
<td>63</td>
</tr>
<tr>
<td>image.fromarray - Function</td>
<td>67</td>
</tr>
<tr>
<td>image.fromascii - Function</td>
<td>70</td>
</tr>
<tr>
<td>image.fromfits - Function</td>
<td>73</td>
</tr>
<tr>
<td>image.fromimage - Function</td>
<td>76</td>
</tr>
<tr>
<td>image.fromshape - Function</td>
<td>79</td>
</tr>
<tr>
<td>image.maketestimage - Function</td>
<td>82</td>
</tr>
<tr>
<td>image.adddegaxes - Function</td>
<td>84</td>
</tr>
<tr>
<td>image.addnoise - Function</td>
<td>87</td>
</tr>
<tr>
<td>image.convolve - Function</td>
<td>90</td>
</tr>
<tr>
<td>image.boundingbox - Function</td>
<td>93</td>
</tr>
<tr>
<td>image.boxcar - Function</td>
<td>96</td>
</tr>
<tr>
<td>image.brightnessunit - Function</td>
<td>100</td>
</tr>
<tr>
<td>image.calc - Function</td>
<td>101</td>
</tr>
<tr>
<td>image.calcmask - Function</td>
<td>103</td>
</tr>
<tr>
<td>image.close - Function</td>
<td>106</td>
</tr>
<tr>
<td>image.continuumsub - Function</td>
<td>107</td>
</tr>
<tr>
<td>image.convertflux - Function</td>
<td>110</td>
</tr>
<tr>
<td>image.convolve2d - Function</td>
<td>112</td>
</tr>
<tr>
<td>image.coordsys - Function</td>
<td>116</td>
</tr>
<tr>
<td>image.coordmeasures - Function</td>
<td>118</td>
</tr>
<tr>
<td>image.decompose - Function</td>
<td>121</td>
</tr>
<tr>
<td>image.deconvolvecomponentlist - Function</td>
<td>125</td>
</tr>
<tr>
<td>image.deconvolvefrombeam - Function</td>
<td>127</td>
</tr>
<tr>
<td>image.beamforconvolvedsize - Function</td>
<td>129</td>
</tr>
<tr>
<td>image.commonbeam - Function</td>
<td>131</td>
</tr>
</tbody>
</table>
image.remove - Function ........................................ 132
image.removefile - Function .................................. 134
image.done - Function ......................................... 135
image.fft - Function ........................................... 137
image.findsources - Function ................................. 140
image.fitprofile - Function .................................. 143
image.fitcomponents - Function .............................. 155
image.fromrecord - Function .................................. 163
image.getchunk - Function ................................... 165
image.getregion - Function ................................... 168
image.getprofile - Function ................................... 171
image.getslice - Function ..................................... 175
image.hanning - Function .................................... 178
image.haslock - Function ..................................... 182
image.histograms - Function ................................ 184
image.history - Function ..................................... 187
image.insert - Function ....................................... 189
image.isopen - Function ....................................... 191
image.ispersistent - Function ............................... 193
image.lock - Function .......................................... 195
image.makecomplex - Function ................................ 197
image.maskhandler - Function ................................ 199
image.miscinfo - Function .................................... 202
image.modify - Function ....................................... 204
image.maxfit - Function ....................................... 206
image.moments - Function ..................................... 208
image.name - Function ......................................... 216
image.open - Function ......................................... 218
image.pad - Function .......................................... 220
image.crop - Function ......................................... 223
image.pixelvalue - Function .................................. 226
image.putchunk - Function .................................... 228
image.putregion - Function ................................... 232
image.rebin - Function ........................................ 235
image.regrid - Function ....................................... 238
image.transpose - Function ................................... 246
image.rotate - Function ....................................... 248
image.rotatebeam - Function .................................. 252
image.rename - Function ....................................... 253
image.replacemaskedpixels - Function ...................... 255
image.beamarea - Function ................................... 260
image.restoringbeam - Function .............................. 261
image.sepconvolve - Function ................................ 263
image.set - Function .......................................... 267
image.setbrightnessunit - Function ......................... 270
image.setcoordsys - Function ................................. 271
regionmanager.wbox - Function ................................................. 385
regionmanager.wpolygon - Function ........................................ 390
1.1.3 coordsys - Tool .............................................................. 393
coordsys.newcoordsys - Function ............................................. 401
coordsys.addcoordinate - Function ......................................... 404
coordsys.axesmap - Function ................................................. 406
coordsys.axiscoordinatetypes - Function ................................. 407
coordsys.conversiontype - Function ........................................ 408
coordsys.convert - Function .................................................. 410
coordsys.convertdirection - Function ...................................... 413
coordsys.convertmany - Function ........................................... 415
coordsys.coordinatestypes - Function ...................................... 417
coordsys.copy - Function ..................................................... 418
coordsys.done - Function ..................................................... 420
coordsys.epoch - Function .................................................... 421
coordsys.findaxis - Function ................................................. 422
coordsys.findaxisbyname - Function ...................................... 424
coordsys.findcoordinate - Function ........................................ 426
coordsys.frequencytofrequency - Function .............................. 428
coordsys.frequencytovelocity - Function ................................. 430
coordsys.fromrecord - Function ............................................. 432
coordsys.increment - Function .............................................. 434
coordsys.lineartransform - Function ...................................... 436
coordsys.names - Function .................................................... 437
coordsys.naxes - Function ..................................................... 439
coordsys.ncoordinates - Function .......................................... 441
coordsys.observer - Function ............................................... 442
coordsys.projection - Function ............................................. 443
coordsys.referencecode - Function ........................................ 445
coordsys.referencepixel - Function ........................................ 447
coordsys.referencevalue - Function ....................................... 449
coordsys.reorder - Function ................................................. 451
coordsys.transpose - Function .............................................. 452
coordsys.replace - Function ................................................ 453
coordsys.restfrequency - Function ........................................ 455
coordsys.setconversiontype - Function ................................. 457
coordsys.getconversiontype - Function .................................. 460
coordsys.setdirection - Function .......................................... 461
coordsys.setepoch - Function ............................................... 464
coordsys.setincrement - Function ........................................ 466
coordsys.setlineartransform - Function ................................. 469
coodsys.setnames - Function ................................................. 471
coordsys.setobserver - Function .......................................... 473
coordsys.setprojection - Function ....................................... 474
coordsys.setreferencecode - Function .................................. 476
coordsys.setreferencelocation - Function ............................. 479
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ms.uvsub</td>
<td>813</td>
</tr>
<tr>
<td>ms.addephemeris</td>
<td>815</td>
</tr>
<tr>
<td>ms.ngetdata</td>
<td>817</td>
</tr>
<tr>
<td>ms.niterinit</td>
<td>818</td>
</tr>
<tr>
<td>ms.niterorigin</td>
<td>819</td>
</tr>
<tr>
<td>ms.niternext</td>
<td>820</td>
</tr>
<tr>
<td>ms.niterend</td>
<td>821</td>
</tr>
<tr>
<td>1.3.2 msmetadata</td>
<td>823</td>
</tr>
<tr>
<td>msmetadata.almaspws</td>
<td>826</td>
</tr>
<tr>
<td>msmetadata.antennadiameter</td>
<td>828</td>
</tr>
<tr>
<td>msmetadata.antennanids</td>
<td>829</td>
</tr>
<tr>
<td>msmetadata.antennanames</td>
<td>831</td>
</tr>
<tr>
<td>msmetadata.antennanoffset</td>
<td>832</td>
</tr>
<tr>
<td>msmetadata.antennaposition</td>
<td>833</td>
</tr>
<tr>
<td>msmetadata.antennastations</td>
<td>834</td>
</tr>
<tr>
<td>msmetadata.antennasforscan</td>
<td>835</td>
</tr>
<tr>
<td>msmetadata.bandwidths</td>
<td>836</td>
</tr>
<tr>
<td>msmetadata.baseband</td>
<td>837</td>
</tr>
<tr>
<td>msmetadata.baselines</td>
<td>838</td>
</tr>
<tr>
<td>msmetadata.chanavgspws</td>
<td>839</td>
</tr>
<tr>
<td>msmetadata.chanfwbws</td>
<td>840</td>
</tr>
<tr>
<td>msmetadata.chanfreqs</td>
<td>842</td>
</tr>
<tr>
<td>msmetadata.channres</td>
<td>843</td>
</tr>
<tr>
<td>msmetadata.chanwidths</td>
<td>844</td>
</tr>
<tr>
<td>msmetadata.close</td>
<td>845</td>
</tr>
<tr>
<td>msmetadata.corrpdsforpol</td>
<td>846</td>
</tr>
<tr>
<td>msmetadata.corrtypesforpol</td>
<td>847</td>
</tr>
<tr>
<td>msmetadata.datadescids</td>
<td>848</td>
</tr>
<tr>
<td>msmetadata.done</td>
<td>849</td>
</tr>
<tr>
<td>msmetadata.effexposuretime</td>
<td>850</td>
</tr>
<tr>
<td>msmetadata.exposuretime</td>
<td>852</td>
</tr>
<tr>
<td>msmetadata.fdmspws</td>
<td>854</td>
</tr>
<tr>
<td>msmetadata.fieldnames</td>
<td>855</td>
</tr>
<tr>
<td>msmetadata.fieldsforintent</td>
<td>856</td>
</tr>
<tr>
<td>msmetadata.fieldsforname</td>
<td>857</td>
</tr>
<tr>
<td>msmetadata.fieldsforscan</td>
<td>858</td>
</tr>
<tr>
<td>msmetadata.fieldsforscans</td>
<td>860</td>
</tr>
<tr>
<td>msmetadata.fieldsforsource</td>
<td>862</td>
</tr>
<tr>
<td>msmetadata.fieldsforsources</td>
<td>863</td>
</tr>
<tr>
<td>msmetadata.fieldsforswp</td>
<td>864</td>
</tr>
<tr>
<td>msmetadata.fieldsfortimes</td>
<td>865</td>
</tr>
<tr>
<td>msmetadata.intents</td>
<td>866</td>
</tr>
<tr>
<td>msmetadata.intentsforfield</td>
<td>867</td>
</tr>
<tr>
<td>msmetadata.intentsforscan</td>
<td>868</td>
</tr>
<tr>
<td>msmetadata.intentsforswp</td>
<td>869</td>
</tr>
<tr>
<td>msmetadata.meanfreq</td>
<td>870</td>
</tr>
</tbody>
</table>
measures.dirshow - Function ........................................ 994
measures.show - Function ........................................... 995
measures.epoch - Function .......................................... 996
measures.direction - Function ...................................... 998
measures.getvalue - Function ...................................... 1000
measures.gettype - Function ...................................... 1001
measures.getref - Function ......................................... 1002
measures.getoffset - Function ...................................... 1003
measures.cometname - Function ................................... 1004
measures.comettype - Function .................................... 1005
measures.cometdist - Function ..................................... 1006
measures.cometangdiam - Function .............................. 1007
measures.comettopo - Function .................................... 1008
measures.framecomet - Function .................................. 1009
measures.position - Function ....................................... 1011
measures.observatory - Function .................................. 1013
measures.obslist - Function ....................................... 1015
measures.linelist - Function ....................................... 1016
measures.spectralline - Function ............................... 1017
measures.sourcelist - Function ................................... 1018
measures.source - Function ........................................ 1019
measures.frequency - Function .................................... 1020
measures.doppler - Function ....................................... 1022
measures.radialvelocity - Function ............................ 1024
measures.shift - Function .......................................... 1026
measures.uvw - Function ........................................... 1027
measures.tourvw - Function ....................................... 1030
measures.expand - Function ........................................ 1033
measures.earthmagnetic - Function ............................. 1035
measures.baseline - Function ...................................... 1037
measures.asbaseline - Function .................................. 1040
measures.listcodes - Function .................................... 1042
measures.measure - Function ....................................... 1044
measures.doframe - Function ....................................... 1047
measures.framenow - Function ..................................... 1049
measures.showframe - Function .................................... 1050
measures.toradialvelocity - Function .......................... 1051
measures.tofrequency - Function ............................... 1053
measures.todoppler - Function .................................... 1055
measures.torestfrequency - Function ......................... 1057
measures.rise - Function ............................................ 1059
measures.riseset - Function ........................................ 1060
measures.posangle - Function ..................................... 1062
measures.separation - Function ................................... 1064
measures.addxvalue - Function ................................... 1066
measures.type - Function ............................................ 1068
2 Package Synthesis

2.1 calibrater - Module

2.1.1 calibrater - Tool

2.1.1.1 calibrater.calibrater - Function

2.1.1.2 calibrater.open - Function

2.1.1.3 calibrater.selectvis - Function

2.1.1.4 calibrater.setmodel - Function

2.1.1.5 calibrater.setptmodel - Function

2.1.1.6 calibrater.setapply - Function

2.1.1.7 calibrater.setcallib - Function

2.1.1.8 calibrater.validatecallib - Function

2.1.1.9 calibrater.setsolve - Function

2.1.1.10 calibrater.setsolvegainspline - Function

2.1.1.11 calibrater.setsolvebandpoly - Function

2.1.1.12 calibrater.state - Function

2.1.1.13 calibrater.reset - Function

2.1.1.14 calibrater.initcalset - Function

2.1.1.15 calibrater.delmod - Function

2.1.1.16 calibrater.solve - Function

2.1.1.17 calibrater.correct - Function

2.1.1.18 calibrater.corrupt - Function

2.1.1.19 calibrater.initweights - Function

2.1.1.20 calibrater.fluxscale - Function

2.1.1.21 calibrater.accumulate - Function

2.1.1.22 calibrater.activityrec - Function

2.1.1.23 calibrater.specifycal - Function

2.1.1.24 calibrater.smooth - Function

2.1.1.25 calibrater.listcal - Function

2.1.1.26 calibrater.posangcal - Function

1.6 spectralline - Module
2.1.2 calanalysis - Tool ........................................... 1227
  calanalysis.calanalysis - Function ................................ 1234
  calanalysis.open - Function ....................................... 1235
  calanalysis.close - Function ..................................... 1236
  calanalysis.calname - Function .................................. 1237
  calanalysis.msname - Function .................................. 1238
  calanalysis.viscal - Function ................................... 1239
  calanalysis.polbasis - Function ................................ 1240
  calanalysis.numfield - Function ................................ 1242
  calanalysis.field - Function .................................... 1243
  calanalysis.numantenna - Function .............................. 1244
  calanalysis.numantenna1 - Function ......................... 1245
  calanalysis.numantenna2 - Function ......................... 1246
  calanalysis.antenna - Function ................................. 1247
  calanalysis.antenna1 - Function ............................... 1248
  calanalysis.antenna2 - Function ............................... 1249
  calanalysis.numfeed - Function ................................. 1250
  calanalysis.feed - Function .................................... 1251
  calanalysis.numtime - Function ................................ 1252
  calanalysis.time - Function .................................... 1253
  calanalysis.numspw - Function ................................ 1254
  calanalysis.spw - Function ..................................... 1255
  calanalysis.numchannel - Function ......................... 1256
  calanalysis.freq - Function .................................... 1257
  calanalysis.get - Function ..................................... 1258
  calanalysis.fit - Function ...................................... 1261

2.2 agentflagger - Module ........................................ 1263
  2.2.1 agentflagger - Tool ....................................... 1264
    agentflagger.agentflagger - Function ....................... 1268
    agentflagger.done - Function ................................ 1269
    agentflagger.open - Function ................................ 1270
    agentflagger.selectdata - Function ....................... 1271
    agentflagger.parseagentparameters - Function ............ 1274
    agentflagger.minit - Function ................................ 1276
    agentflagger.run - Function .................................. 1277
    agentflagger.getflagversionlist - Function ............... 1278
    agentflagger.printflagselction - Function .............. 1279
    agentflagger.saveflagversion - Function ................. 1280
agentflagger.restoreflagversion - Function ............................................ 1281
agentflagger.deleteflagversion - Function ............................................. 1282
agentflagger.parsecliptmanualparameters - Function .............................. 1283
agentflagger.parseclipparameters - Function ........................................ 1286
agentflagger.parsequackparameters - Function ...................................... 1289
agentflagger.parseselevationparameters - Function ............................... 1292
agentflagger.parseextendparameters - Function ................................... 1295
agentflagger.parsesummaryparameters - Function ................................ 1298

2.3 imager - Module ............................................................................... 1303
  2.3.1 imager - Tool .............................................................................. 1314
    imager.imager - Function ................................................................ 1316
    imager.advise - Function ................................................................ 1318
    imager.advisechansel - Function ...................................................... 1320
    imager.approximatepsf - Function .................................................... 1323
    imager.boxmask - Function ................................................................ 1325
    imager.calcuvw - Function ................................................................ 1327
    imager.clean - Function .................................................................. 1329
    imager.clipimage - Function ............................................................. 1332
    imager.clipvis - Function .................................................................. 1335
    imager.close - Function .................................................................. 1337
    imager.defineimage - Function .......................................................... 1338
    imager.drawmask - Function .............................................................. 1345
    imager.exprmask - Function .............................................................. 1347
    imager.feather - Function ................................................................. 1349
    imager.filter - Function .................................................................. 1352
    imager.fitpsf - Function .................................................................. 1354
    imager.fixvis - Function .................................................................. 1356
    imager.ft - Function .......................................................................... 1358
    imager.getweightgrid - Function ....................................................... 1360
    imager.linearmosaic - Function ......................................................... 1362
    imager.make - Function ..................................................................... 1364
    imager.predictcomp - Function .......................................................... 1366
    imager.makeimage - Function ............................................................. 1368
    imager.makemodelfromsd - Function .................................................... 1371
    imager.mask - Function ..................................................................... 1373
    imager.mem - Function ...................................................................... 1375
    imager.mls - Function ........................................................................ 1379
    imager.open - Function ..................................................................... 1382
    imager.pb - Function ......................................................................... 1383
    imager.plotsummary - Function .......................................................... 1386
    imager.plotuv - Function .................................................................... 1387
    imager.plotvis - Function ................................................................... 1388
    imager.plotweights - Function ........................................................... 1389
    imager.regionmask - Function ............................................................ 1390
imager.regiontoimagemask - Function . . . . . . . . . . . 1393
imager.residual - Function . . . . . . . . . . . . . . . . . 1395
imager.restore - Function . . . . . . . . . . . . . . . . . . 1396
imager.updateresidual - Function . . . . . . . . . . . . . 1398
imager.sensitivity - Function . . . . . . . . . . . . . . . . 1400
imager.apparentsens - Function . . . . . . . . . . . . . 1402
imager.setbeam - Function . . . . . . . . . . . . . . . . . . 1405
imager.selectvis - Function . . . . . . . . . . . . . . . . . 1406
imager.setjy - Function . . . . . . . . . . . . . . . . . . . 1410
imager.ssoflux - Function . . . . . . . . . . . . . . . . . . 1413
imager.setncontrol - Function . . . . . . . . . . . . . . . . 1414
imager.setoptions - Function . . . . . . . . . . . . . . . . 1418
imager.setscales - Function . . . . . . . . . . . . . . . . . . 1423
imager.setsmallscalebias - Function . . . . . . . . . . . . 1425
imager.settaylorterms - Function . . . . . . . . . . . . . 1426
imager.setsdoptions - Function . . . . . . . . . . . . . . . . 1427
imager.setvp - Function . . . . . . . . . . . . . . . . . . . 1430
imager.setweightgrid - Function . . . . . . . . . . . . . . 1432
imager.smooth - Function . . . . . . . . . . . . . . . . . . . 1434
imager.stop - Function . . . . . . . . . . . . . . . . . . . . 1436
imager.summary - Function . . . . . . . . . . . . . . . . . . 1437
imager.uvrange - Function . . . . . . . . . . . . . . . . . . . 1438
imager.weight - Function . . . . . . . . . . . . . . . . . . . 1439
imager.mapextent - Function . . . . . . . . . . . . . . . . . 1442

2.3.2 vpmanager - Tool . . . . . . . . . . . . . . . . . . . . . . 1443
vpmanager.vpmanager - Function . . . . . . . . . . . . . 1446
vpmanager.saveastable - Function . . . . . . . . . . . . . 1447
vpmanager.loadfromtable - Function . . . . . . . . . . . 1448
vpmanager.summarizevps - Function . . . . . . . . . . . . 1449
vpmanager.setcannedpb - Function . . . . . . . . . . . . 1450
vpmanager.setpbairy - Function . . . . . . . . . . . . . . 1452
vpmanager.setpbcospoly - Function . . . . . . . . . . . . 1455
vpmanager.setpbgauss - Function . . . . . . . . . . . . . 1458
vpmanager.setpbminpoly - Function . . . . . . . . . . . . 1461
vpmanager.setpbnumeric - Function . . . . . . . . . . . . 1464
vpmanager.setpimage - Function . . . . . . . . . . . . . . 1467
vpmanager.setpbpoly - Function . . . . . . . . . . . . . . 1469
vpmanager.setpbtables - Function . . . . . . . . . . . . . 1472
vpmanager.reset - Function . . . . . . . . . . . . . . . . . . 1473
vpmanager.setuserdefault - Function . . . . . . . . . . . 1474
vpmanager.getuserdefault - Function . . . . . . . . . . . 1475
vpmanager.getanttypes - Function . . . . . . . . . . . . . 1476
vpmanager.numvps - Function . . . . . . . . . . . . . . . . . 1477
vpmanager.getvp - Function . . . . . . . . . . . . . . . . . . 1478
vpmanager.getvps - Function . . . . . . . . . . . . . . . . . 1479
vpmanager.createantresp - Function . . . . . . . . . . . . 1481
3.1.2 deconvolver - Tool

- deconvolver.deconvolver - Function
- deconvolver.open - Function
- deconvolver.reopen - Function
- deconvolver.close - Function
- deconvolver.done - Function
- deconvolver.summary - Function
- deconvolver.boxmask - Function
- deconvolver.regionmask - Function
- deconvolver.clipimage - Function
- deconvolver.clarkclean - Function
- deconvolver.fullclarkclean - Function
- deconvolver.dirtyname - Function
- deconvolver.psfname - Function
- deconvolver.make - Function
- deconvolver.convolve - Function
- deconvolver.makegaussian - Function
- deconvolver.state - Function
- deconvolver.updatestate - Function
- deconvolver.clean - Function
- deconvolver.naclean - Function
- deconvolver.setscales - Function
- deconvolver.ft - Function
- deconvolver.restore - Function
- deconvolver.residual - Function
- deconvolver.smooth - Function
- deconvolver.mem - Function
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>table.rownumbers - Function</td>
<td>1682</td>
</tr>
<tr>
<td>table.setmaxcachesize - Function</td>
<td>1684</td>
</tr>
<tr>
<td>table.isscalarcol - Function</td>
<td>1685</td>
</tr>
<tr>
<td>table.isvarcol - Function</td>
<td>1686</td>
</tr>
<tr>
<td>table.coldatatype - Function</td>
<td>1687</td>
</tr>
<tr>
<td>table.colarraytype - Function</td>
<td>1688</td>
</tr>
<tr>
<td>table.ncols - Function</td>
<td>1689</td>
</tr>
<tr>
<td>table.nrows - Function</td>
<td>1690</td>
</tr>
<tr>
<td>table.addrows - Function</td>
<td>1691</td>
</tr>
<tr>
<td>table.removerows - Function</td>
<td>1692</td>
</tr>
<tr>
<td>table.addcols - Function</td>
<td>1693</td>
</tr>
<tr>
<td>table.renamecol - Function</td>
<td>1695</td>
</tr>
<tr>
<td>table.removecols - Function</td>
<td>1697</td>
</tr>
<tr>
<td>table.iscelldefined - Function</td>
<td>1698</td>
</tr>
<tr>
<td>table.getcell - Function</td>
<td>1699</td>
</tr>
<tr>
<td>table.getcellslice - Function</td>
<td>1700</td>
</tr>
<tr>
<td>table.getcol - Function</td>
<td>1702</td>
</tr>
<tr>
<td>table.getvarcol - Function</td>
<td>1704</td>
</tr>
<tr>
<td>table.getcolslice - Function</td>
<td>1706</td>
</tr>
<tr>
<td>table.putcell - Function</td>
<td>1708</td>
</tr>
<tr>
<td>table.putcellslice - Function</td>
<td>1709</td>
</tr>
<tr>
<td>table.putcol - Function</td>
<td>1710</td>
</tr>
<tr>
<td>table.putvarcol - Function</td>
<td>1711</td>
</tr>
<tr>
<td>table.putcolslice - Function</td>
<td>1713</td>
</tr>
<tr>
<td>table.getcolshapestring - Function</td>
<td>1715</td>
</tr>
<tr>
<td>table.getkeyword - Function</td>
<td>1717</td>
</tr>
<tr>
<td>table.getkeywords - Function</td>
<td>1719</td>
</tr>
<tr>
<td>table.getcolkeyword - Function</td>
<td>1721</td>
</tr>
<tr>
<td>table.getcolkeywords - Function</td>
<td>1722</td>
</tr>
<tr>
<td>table.putkeyword - Function</td>
<td>1723</td>
</tr>
<tr>
<td>table.putkeywords - Function</td>
<td>1725</td>
</tr>
<tr>
<td>table.putcolkeyword - Function</td>
<td>1727</td>
</tr>
<tr>
<td>table.putcolkeywords - Function</td>
<td>1729</td>
</tr>
<tr>
<td>table.removekeyword - Function</td>
<td>1731</td>
</tr>
<tr>
<td>table.removecolkeyword - Function</td>
<td>1732</td>
</tr>
<tr>
<td>table.getdminfo - Function</td>
<td>1733</td>
</tr>
<tr>
<td>table.keywordnames - Function</td>
<td>1734</td>
</tr>
<tr>
<td>table.fieldnames - Function</td>
<td>1735</td>
</tr>
<tr>
<td>table.colkeywordnames - Function</td>
<td>1736</td>
</tr>
<tr>
<td>table.colfieldnames - Function</td>
<td>1737</td>
</tr>
<tr>
<td>table.getdesc - Function</td>
<td>1738</td>
</tr>
<tr>
<td>table.getcoldesc - Function</td>
<td>1739</td>
</tr>
<tr>
<td>table.ok - Function</td>
<td>1740</td>
</tr>
<tr>
<td>table.clearlocks - Function</td>
<td>1741</td>
</tr>
<tr>
<td>table.listlocks - Function</td>
<td>1742</td>
</tr>
<tr>
<td>table.statistics - Function</td>
<td>1743</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>table.showcache</td>
<td>1745</td>
</tr>
<tr>
<td>table.testincrstman</td>
<td>1746</td>
</tr>
<tr>
<td>tableplot</td>
<td>1747</td>
</tr>
<tr>
<td>tableplot.open</td>
<td>1748</td>
</tr>
<tr>
<td>tableplot.setgui</td>
<td>1749</td>
</tr>
<tr>
<td>tableplot.savefig</td>
<td>1750</td>
</tr>
<tr>
<td>tableplot.selectdata</td>
<td>1752</td>
</tr>
<tr>
<td>tableplot.plotdata</td>
<td>1753</td>
</tr>
<tr>
<td>tableplot.iterplotstart</td>
<td>1759</td>
</tr>
<tr>
<td>tableplot.replot</td>
<td>1761</td>
</tr>
<tr>
<td>tableplot.iterplotnext</td>
<td>1762</td>
</tr>
<tr>
<td>tableplot.iterplotstop</td>
<td>1764</td>
</tr>
<tr>
<td>tableplot.markregions</td>
<td>1766</td>
</tr>
<tr>
<td>tableplot.flagdata</td>
<td>1768</td>
</tr>
<tr>
<td>tableplot.unflagdata</td>
<td>1769</td>
</tr>
<tr>
<td>tableplot.locatedata</td>
<td>1770</td>
</tr>
<tr>
<td>tableplot.clearflags</td>
<td>1771</td>
</tr>
<tr>
<td>tableplot.saveflagversion</td>
<td>1772</td>
</tr>
<tr>
<td>tableplot.restoreflagversion</td>
<td>1773</td>
</tr>
<tr>
<td>tableplot.deleteflagversion</td>
<td>1774</td>
</tr>
<tr>
<td>tableplot.getflagversionlist</td>
<td>1775</td>
</tr>
<tr>
<td>tableplot.clearplot</td>
<td>1776</td>
</tr>
<tr>
<td>tableplot.done</td>
<td>1777</td>
</tr>
</tbody>
</table>

3.2.2 tableplot - Tool

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>atmosphere.initAtmosphere</td>
<td>1778</td>
</tr>
<tr>
<td>atmosphere.getAtmVersion</td>
<td>1779</td>
</tr>
<tr>
<td>atmosphere.listAtmosphereTypes</td>
<td>1783</td>
</tr>
<tr>
<td>atmosphere.initAtmProfile</td>
<td>1784</td>
</tr>
<tr>
<td>atmosphere.updateAtmProfile</td>
<td>1787</td>
</tr>
<tr>
<td>atmosphere.getBasicAtmParms</td>
<td>1789</td>
</tr>
<tr>
<td>atmosphere.getNumLayers</td>
<td>1792</td>
</tr>
<tr>
<td>atmosphere.getGroundWH2O</td>
<td>1793</td>
</tr>
<tr>
<td>atmosphere.getProfile</td>
<td>1794</td>
</tr>
<tr>
<td>atmosphere.initSpectralWindow</td>
<td>1797</td>
</tr>
<tr>
<td>atmosphere.addSpectralWindow</td>
<td>1799</td>
</tr>
<tr>
<td>atmosphere.getNumSpectralWindows</td>
<td>1801</td>
</tr>
<tr>
<td>atmosphere.getNumChan</td>
<td>1802</td>
</tr>
<tr>
<td>atmosphere.getRefChan</td>
<td>1803</td>
</tr>
<tr>
<td>atmosphere.getRefFreq</td>
<td>1804</td>
</tr>
<tr>
<td>atmosphere.getChanSep</td>
<td>1805</td>
</tr>
<tr>
<td>atmosphere.getChanFreq</td>
<td>1806</td>
</tr>
<tr>
<td>atmosphere.getSpectralWindow</td>
<td>1808</td>
</tr>
<tr>
<td>atmosphere.getChanNum</td>
<td>1809</td>
</tr>
</tbody>
</table>
atmosphere.getBandwidth - Function . . . . . . . . . . . 1811
atmosphere.getMinFreq - Function . . . . . . . . . . . . 1812
atmosphere.getMaxFreq - Function . . . . . . . . . . . . 1813
atmosphere.getDryOpacity - Function . . . . . . . . . . . 1814
atmosphere.getDryContOpacity - Function . . . . . . . . . 1815
atmosphere.getO2LinesOpacity - Function . . . . . . . . . 1816
atmosphere.getO3LinesOpacity - Function . . . . . . . . . 1817
atmosphere.getCOLinesOpacity - Function . . . . . . . . . 1818
atmosphere.getN2OLinesOpacity - Function . . . . . . . . . 1819
atmosphere.getWetOpacity - Function . . . . . . . . . . . . 1820
atmosphere.getH2OLinesOpacity - Function . . . . . . . . . 1821
atmosphere.getH2OContOpacity - Function . . . . . . . . . 1822
atmosphere.getDryOpacitySpec - Function . . . . . . . . . . 1823
atmosphere.getWetOpacitySpec - Function . . . . . . . . . . 1824
atmosphere.getDispersivePhaseDelay - Function . . . . . . 1826
atmosphere.getDispersiveWetPhaseDelay - Function . . . . . 1828
atmosphere.getNonDispersiveWetPhaseDelay - Function . . . 1830
atmosphere.getNonDispersiveDryPhaseDelay - Function . . . 1832
atmosphere.getNonDispersivePhaseDelay - Function . . . . . 1834
atmosphere.getDispersivePathLength - Function . . . . . . 1836
atmosphere.getDispersiveWetPathLength - Function . . . . . 1838
atmosphere.getNonDispersiveWetPathLength - Function . . . 1840
atmosphere.getNonDispersiveDryPathLength - Function . . . 1842
atmosphere.getO2LinesPathLength - Function . . . . . . . . . 1844
atmosphere.getO3LinesPathLength - Function . . . . . . . . . 1845
atmosphere.getCOLinesPathLength - Function . . . . . . . . . 1846
atmosphere.getN2OLinesPathLength - Function . . . . . . . . . 1847
atmosphere.getNonDispersivePathLength - Function . . . . . 1848
atmosphere.getAbsH2OLines - Function . . . . . . . . . . . . 1850
atmosphere.getAbsH2OCont - Function . . . . . . . . . . . . 1851
atmosphere.getAbsO2Lines - Function . . . . . . . . . . . . 1852
atmosphere.getAbsO3Lines - Function . . . . . . . . . . . . 1853
atmosphere.getAbsCOLines - Function . . . . . . . . . . . . 1854
atmosphere.getAbsN2OLines - Function . . . . . . . . . . . . 1855
atmosphere.getAbsTotalDry - Function . . . . . . . . . . . . 1856
atmosphere.getAbsTotalWet - Function . . . . . . . . . . . . 1857
atmosphere.setUserWH2O - Function . . . . . . . . . . . . . 1858
atmosphere.getUserWH2O - Function . . . . . . . . . . . . . 1859
atmosphere.setSkyBackgroundTemperature - Function . . . . . 1860
atmosphere.getSkyBackgroundTemperature - Function . . . . . 1861
atmosphere.getAverageTebbSky - Function . . . . . . . . . . 1862
atmosphere.getTebbSky - Function . . . . . . . . . . . . . 1863
atmosphere.getTebbSkySpec - Function . . . . . . . . . . . . 1868
5 Package SingleDish

5.1 sd - Module

5.1.1 sd - Tool

sd.almaCal - Function

sd.apexCal - Function  

sd.average_time - Function

sd.caifs - Function  

sd.calibrate - Function

sd.calmad - Function

sd.caips - Function

sd.commands - Function

sd.dosigRef - Function

sd.dototalPower - Function

sd.get_revision - Function

sd.is_asap_cl - Function

sd.is_casapy - Function

sd.list_files - Function

sd.list_rparameters - Function

sd.list_scans - Function

sd.mask_and - Function

sd.mask_or - Function

sd.mask_not - Function

sd.merge - Function

sd.quotient - Function

sd.rc - Function

sd.skydip - Function

sd.splitant - Function

sd.unique - Function

sd.welcome - Function

5.1.2 sd.scantable - Tool

sd.scantable.add - Function

sd.scantable.auto_cspine_baseline - Function

sd.scantable.auto_poly_baseline - Function

sd.scantable.auto_quotient - Function

sd.scantable.auto_sinusoid_baseline - Function

sd.scantable.average_beam - Function

sd.scantable.average_pol - Function

sd.scantable.average_time - Function

sd.scantable.bin - Function

sd.scantable.chan2data - Function

sd.scantable.clip - Function

sd.scantable.convert_flux - Function
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd.scantable.convert_pol</td>
<td>1945</td>
</tr>
<tr>
<td>sd.scantable.copy</td>
<td>1946</td>
</tr>
<tr>
<td>sd.scantable.create_mask</td>
<td>1947</td>
</tr>
<tr>
<td>sd.scantable.cspline_baseliene</td>
<td>1949</td>
</tr>
<tr>
<td>sd.scantable.drop_scan</td>
<td>1952</td>
</tr>
<tr>
<td>sd.scantable.fft</td>
<td>1953</td>
</tr>
<tr>
<td>sd.scantable.flag</td>
<td>1954</td>
</tr>
<tr>
<td>sd.scantable.flag_nans</td>
<td>1955</td>
</tr>
<tr>
<td>sd.scantable.flag_row</td>
<td>1956</td>
</tr>
<tr>
<td>sd.scantable.freq_align</td>
<td>1957</td>
</tr>
<tr>
<td>sd.scantable.freq_switch</td>
<td>1958</td>
</tr>
<tr>
<td>sd.scantable.gain_el</td>
<td>1959</td>
</tr>
<tr>
<td>sd.scantable.get_abcsiss</td>
<td>1961</td>
</tr>
<tr>
<td>sd.scantable.get_antennaname</td>
<td>1962</td>
</tr>
<tr>
<td>sd.scantable.get_azimuth</td>
<td>1963</td>
</tr>
<tr>
<td>sd.scantable.get_column_names</td>
<td>1964</td>
</tr>
<tr>
<td>sd.scantable.get_coordinate</td>
<td>1965</td>
</tr>
<tr>
<td>sd.scantable.get_direction</td>
<td>1966</td>
</tr>
<tr>
<td>sd.scantable.get_directionval</td>
<td>1967</td>
</tr>
<tr>
<td>sd.scantable.get_elevation</td>
<td>1968</td>
</tr>
<tr>
<td>sd.scantable.get_fit</td>
<td>1969</td>
</tr>
<tr>
<td>sd.scantable.get_fluxunit</td>
<td>1970</td>
</tr>
<tr>
<td>sd.scantable.get_inter_time</td>
<td>1971</td>
</tr>
<tr>
<td>sd.scantable.get_mask</td>
<td>1972</td>
</tr>
<tr>
<td>sd.scantable.get_mask_indices</td>
<td>1973</td>
</tr>
<tr>
<td>sd.scantable.get_masklist</td>
<td>1974</td>
</tr>
<tr>
<td>sd.scantable.get_parangle</td>
<td>1975</td>
</tr>
<tr>
<td>sd.scantable.get_reffreqs</td>
<td>1976</td>
</tr>
<tr>
<td>sd.scantable.get_rms</td>
<td>1977</td>
</tr>
<tr>
<td>sd.scantable.get_row</td>
<td>1978</td>
</tr>
<tr>
<td>sd.scantable.get_row_selector</td>
<td>1979</td>
</tr>
<tr>
<td>sd.scantable.get_scan</td>
<td>1980</td>
</tr>
<tr>
<td>sd.scantable.get_selection</td>
<td>1981</td>
</tr>
<tr>
<td>sd.scantable.get_sourcename</td>
<td>1982</td>
</tr>
<tr>
<td>sd.scantable.get_spectrum</td>
<td>1983</td>
</tr>
<tr>
<td>sd.scantable.get_time</td>
<td>1984</td>
</tr>
<tr>
<td>sd.scantable.get_tsys</td>
<td>1985</td>
</tr>
<tr>
<td>sd.scantable.get_unit</td>
<td>1986</td>
</tr>
<tr>
<td>sd.scantable.get_weather</td>
<td>1987</td>
</tr>
<tr>
<td>sd.scantable.getbeam</td>
<td>1988</td>
</tr>
<tr>
<td>sd.scantable.getbeamnos</td>
<td>1989</td>
</tr>
<tr>
<td>sd.scantable.getcycle</td>
<td>1990</td>
</tr>
<tr>
<td>sd.scantable.gettif</td>
<td>1991</td>
</tr>
<tr>
<td>sd.scantable.gettfnos</td>
<td>1992</td>
</tr>
<tr>
<td>sd.scantable.getmolnos</td>
<td>1993</td>
</tr>
<tr>
<td>sd.scantable.getpol</td>
<td>1994</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>sd.selector.get_name</td>
<td>2050</td>
</tr>
<tr>
<td>sd.selector.get_order</td>
<td>2051</td>
</tr>
<tr>
<td>sd.selector.get_pols</td>
<td>2052</td>
</tr>
<tr>
<td>sd.selector.get_poltyes</td>
<td>2053</td>
</tr>
<tr>
<td>sd.selector.get_query</td>
<td>2054</td>
</tr>
<tr>
<td>sd.selector.get_rows</td>
<td>2055</td>
</tr>
<tr>
<td>sd.selector.get_scans</td>
<td>2056</td>
</tr>
<tr>
<td>sd.selector.get_types</td>
<td>2057</td>
</tr>
<tr>
<td>sd.selector.is_empty</td>
<td>2058</td>
</tr>
<tr>
<td>sd.selector.reset</td>
<td>2059</td>
</tr>
<tr>
<td>sd.selector.set_beams</td>
<td>2060</td>
</tr>
<tr>
<td>sd.selector.set_cycles</td>
<td>2061</td>
</tr>
<tr>
<td>sd.selector.set_ifs</td>
<td>2062</td>
</tr>
<tr>
<td>sd.selector.set_name</td>
<td>2063</td>
</tr>
<tr>
<td>sd.selector.set_order</td>
<td>2064</td>
</tr>
<tr>
<td>sd.selector.set_polarisations</td>
<td>2065</td>
</tr>
<tr>
<td>sd.selector.set_polarizations</td>
<td>2066</td>
</tr>
<tr>
<td>sd.selector.set_pols</td>
<td>2067</td>
</tr>
<tr>
<td>sd.selector.set_query</td>
<td>2068</td>
</tr>
<tr>
<td>sd.selector.set_rows</td>
<td>2069</td>
</tr>
<tr>
<td>sd.selector.set_scans</td>
<td>2070</td>
</tr>
<tr>
<td>sd.selector.set_tsys</td>
<td>2071</td>
</tr>
<tr>
<td>sd.selector.set_types</td>
<td>2072</td>
</tr>
<tr>
<td>sd.fitter.auto_fit</td>
<td>2073</td>
</tr>
<tr>
<td>sd.fitter.commit</td>
<td>2074</td>
</tr>
<tr>
<td>sd.fitter.fit</td>
<td>2075</td>
</tr>
<tr>
<td>sd.fitter.get_area</td>
<td>2076</td>
</tr>
<tr>
<td>sd.fitter.get_chi2</td>
<td>2077</td>
</tr>
<tr>
<td>sd.fitter.get_errors</td>
<td>2078</td>
</tr>
<tr>
<td>sd.fitter.get_estimate</td>
<td>2079</td>
</tr>
<tr>
<td>sd.fitter.get_fit</td>
<td>2080</td>
</tr>
<tr>
<td>sd.fitter.get_parameters</td>
<td>2081</td>
</tr>
<tr>
<td>sd.fitter.get_residual</td>
<td>2082</td>
</tr>
<tr>
<td>sd.fitter.plot</td>
<td>2083</td>
</tr>
<tr>
<td>sd.fitter.set_data</td>
<td>2084</td>
</tr>
<tr>
<td>sd.fitter.set_function</td>
<td>2085</td>
</tr>
<tr>
<td>sd.fitter.set_gauss_parameters</td>
<td>2086</td>
</tr>
<tr>
<td>sd.fitter.set_lorentz_parameters</td>
<td>2087</td>
</tr>
<tr>
<td>sd.fitter.set_parameters</td>
<td>2088</td>
</tr>
<tr>
<td>sd.fitter.set_sinusoid_parameters</td>
<td>2089</td>
</tr>
<tr>
<td>sd.fitter.set_scan</td>
<td>2090</td>
</tr>
<tr>
<td>sd.fitter.store_fit</td>
<td>2091</td>
</tr>
<tr>
<td>sd.linecatalog.get_frequency</td>
<td>2092</td>
</tr>
<tr>
<td>sd.linecatalog.get_name</td>
<td>2093</td>
</tr>
</tbody>
</table>
Chapter 1

Package General

The general package contains modules that are of general use for astronomical processing.

1.1 images - Module

Access and analysis of images

Description
This module contains functionality to access, create, and analyze CASA images. It offers both basic services and higher-level packaged tools. The available tools in this module are

- [Image] - create, manipulate and analyze images (default tool is ia). This tool provides a range of low to medium level services.
- [Regionmanager] - create and manipulate regions-of-interest (default tool is rg). **WARNING! Documentation describes Glish-based tool which has been only partially ported to CASA.**
- [Coordsys] - create and manipulate Coordinate Systems (default tool is cs).
- [Imagepol] - this offers specialized polarimetric analysis of images.

Images
We refer to a CASA image file when we are referring to the actual data stored on disk. The name that you give a CASA image file is actually the name of a directory containing a collection of CASA tables which together constitute the image file. But you only need to refer to the directory name and you can think of it as one logical file. Some images don’t have an associated disk file. These are called “virtual” images (e.g. an image that is temporary and
entirely in memory). Whenever we use the word “image”, we are just using it in a generic sense.

Images are manipulated with an Image tool.

**Pixel mask**

A pixel mask specifies which pixels are to be considered good (mask value $T$) or bad (mask value $F$). For example, you may have imported a FITS file which has blanked pixels in it. These will be converted into pixel mask elements whose values are bad ($F$). Or you may have made an error analysis of an image and computed via a statistical test that certain pixels should be masked out for future analysis.

An image file may contain zero, one, or more pixel masks. However, only one mask will be designated as the default mask and be applied to the data.

For more details, see the Image tool.

**Region-of-interest**

A region-of-interest or simply, region, designates which pixels of the image you are interested in for some (generally) astrophysical reason. This complements the pixel mask which specifies which pixels are good or bad (for statistical reasons). Regions-of-interest are generated and manipulated with the Regionmanager tool.

Briefly, a region-of-interest may be either a simple shape such as a multi-dimensional box, or a 2-D polygon, or some compound combination of regions-of-interest. For example, a 2-D polygon defined in the X and Y axes extended along the Z axis, or perhaps a union or intersection of regions.

See the Regionmanager documentation for more details on regions.

**Coordinates**

We will often refer to (absolute) pixel coordinates. Consider a 2-D image with shape [10,20]. Then our model is that the centre of the bottom-left corner pixel has pixel coordinate $[X,Y] = [0,0]$. The centre of the top-right corner pixel has pixel coordinate $[X,Y] = [9,19]$.

When a physical Coordinate System (e.g. an RA/DEC direction coordinate) is attached to an image, then we can convert pixel coordinates to a world (or physical) coordinate.

The Coordsys tool is available for manipulating Coordinate Systems. You can recover the Coordinate System from an image into a Coordsys tool via the Image function coordsys.

For more details, see the Image and Coordsys tools.

**Lattice Expression Language (LEL)**

This allows you to manipulate expressions involving images. For example, add this image to that image, or multiply the minimum value of that image by the square root of this image. The LEL syntax is quite rich and is described in detail in [note 223](#).

LEL is used in several of the Image tool functions.
Example  Here is a simple example to give you the flavour of using the image module. Suppose we have an image FITS disk file; we would like to convert it to a CASA image file (store the image in a CASA table), look at the header information and store it in a record for future use, work out some statistics and then close the image.

```python
""
print "\t----\t Module Ex 1 \t----"
pathname=os.environ.get("CASAPATH")
pathname=pathname.split()[0]
datapath=pathname+"/data/demo/Images/imagetestimage.fits"
ia.fromfits(outfile='testimage.im', infile=datapath, overwrite=T) # 1
hdr = ia.summary() # 2
ia.statistics() # 3
ia.close() # 4
print "Last example! Exiting..."
exit()
""
```

1. Convert (using `fromfits`) an image FITS disk file called `imagetestimage.fits` in the directory specified by the environment variable `$CASAPATH` to a CASA image file called `testimage.im`. The CASA image file is now associated with the default image tool `ia`.

2. This summarises (to the Logger) the basic header information in the image (name, masks, regions, brightness units, coordinates etc) and also stores it in a record named `hdr`.

3. This tool function evaluates basic statistics from the entire image.

4. This closes the Image tool, but does not delete its associated disk image file.
1.1.1 image - Tool

Operations on images
Requires: coordsys

Synopsis

Description

Summary
An Image tool provides access to CASA images. Currently only single precision floating point CASA images are supported by all methods in the Image tool and complex-valued images are supported by many, but not all, methods. Image tools also provide direct (native) access to FITS and Miriad images. You can also convert these foreign formats to CASA format (for optimum processing speed).
It is important to note that many methods return new image tools that are attached to an image that method has created. Even if one does not intend on using this returned tool, it is important to capture it and run done() on it or it will continue to use resources unnecessarily, eg

```python
new_image_tool = ia.collapse("my_collapsed.im")
# do things with new_image_tool and then run done() on it
new_image_tool.done()
```

Overview of Image tool functionality

- **Conversion** - There is functionality to interconvert between CASA images and FITS files. There is also native access to a FITS file:
  
  - fromfits - Convert a FITS image file to a CASA image
  - tofits - convert the image to a FITS file
  - image - native access to a FITS file
  
  - fromascii - Convert an ascii image file to a CASA image
  - fromarray - Convert an array into a CASA image
  - fromshape - Convert a shape into a CASA image

- **Analysis** -
- subimage (function) - collapse image along specified axis, computing aggregate function of pixels along that axis
- decompose - separate a complex image into individual components
- deconvolvecomponentlist - deconvolve a Componentlist from the restoring beam
- fft - FFT the image
- findsources - Find strong point sources in sky
- fitcomponents - Fit model components to an image.
- fitprofile - fit a 1-d profile with varying combinations of functional forms (see also the imageprofilefitter tool.
- histograms - compute histograms from the image
- insert - insert specified image into this image
- maxfit - Find maximum and do simple parabolic fit to sky
- modify - modify image by a model
- moments - compute moments from image
- regrid - regrid the image to the specified Coordinate System
- reorder - transpose the image (same as transpose())
- transpose - transpose the image (same as reorder())
- rotate - rotate the coordinate system and regrid the image to the rotated Coordinate System
- rebin - rebin an image by the specified binning factors
- statistics - compute statistics from the image
- twopointcorrelation - compute two point autocorrelation functions from the image
- subimage (function) - Create a (sub)image from a region of the image

- **Coordinates** - Manipulation of the coordinate system is handled through
  - coordmeasures - convert from pixel to world coordinate wrapped as Measures
  - coordsys - recover the Coordinate System into a Coordsys tool.
  - setcoordsys - set a new Coordinate System
  - topixel - convert from world coordinate to pixel coordinate
  - toworld - convert from pixel coordinate to world coordinate

The coordsys tool provides more extensive coordinate system manipulation.
• **Filtering** - Images may be filtered via
  
  – convolve - Convolve image with an array or by another image
  – convolve2d - Convolve image by a 2D kernel
  – sepconvolve - Separable convolution
  – hanning - Hanning convolution along one axis
  
  In the future filtering other than convolution will be provided

• **Masks** - Masks may be manipulated via
  
  – calcmask - Image mask calculator
  – maskhandler - handle masks (set, copy, delete, recover names)
  – replacemaskedpixels - replace the values of pixels which are masked bad
  – set - set pixel and/or mask values with a scalar in a **region-of-interest** of the image
  – summary - lists the mask names

• **Pixel access** - The pixel and mask values for an image may be accessed and calculated with via
  
  – imagecalc - Create image tool with image calculator
  – image - Create an image tool from a **CASA** image
  – calc - Image pixel calculator
  – calcmask - Image mask calculator
  – getchunk - get the pixel values from a regular region of the image into an array
  – getregion - get pixels and mask from a **region-of-interest** of the image
  – getslice - get a 1-D slice from the image
  – pixelvalue - get image value for specified pixel
  – putchunk - put pixels from an array into a regular region of the image
  – putregion - put pixels and mask into a **region-of-interest** of the image
  – set - set pixel and/or mask values with a scalar in a **region-of-interest** of the image

• **Inquiry** - Functions to report basic information about the image are
  
  – boundingbox - find bounding box of a **region-of-interest**.
- brightnessunit - Get image brightness unit
- haslock - does this image have a lock set
- history - recover/list history file
- ispersistent - is the image persistent (on disk)
- name - name of the image file this tool is attached to
- restoringbeam - Get restoring beam
- shape - the length of each axis in the image
- summary - summarize basic information about the image
- type - the type of this Image tool

*Utility* - There is wide range of utility services available through the functions

- adddegaxes - Add degenerate axes
- addnoise - Add noise to the image
- brightnessunit - Get image brightness unit
- close - Close the image tool (but don’t destroy it)
- convertflux - Convert flux density between peak and integral
- close - close this image tool
- haslock - does this image have a lock set
- history - recover/list history file
- imagefiles - Find the names of all image files in the given directory
- imagetools - Find the names of all global image tools
- isimage - Is this variable an Image tool
- isopen - Is this Image tool open?
- lock - acquire a lock on the image
- makecomplex - make a complex image from two real images
- miscinfo - recover miscellaneous information record
- open - open a new image file with this image tool
- rename - rename the image file associated with this Image tool
- restoringbeam - Get restoring beam
- remove - remove the image file associated with this Image tool
- setbrightnessunit - Set image brightness unit
- sethistory - set the history file
- setmiscinfo - set the miscellaneous information record
- setrestoringbeam - Set new restoring beam
- unlock - release lock on this image file

- **Reshaping** - Images can be reshaped via
  - fromimage - Create a (sub)image from a region of a CASA image
  - subimage - Create a (sub)image from a region of the image
  - insert - insert specified image into this image
  - imageconcat - Concatenate CASA images

**General**

We refer to a CASA image file when we are referring to the actual data stored on disk. The name that you give a CASA image file is actually the name of a directory containing a collection of CASA tables which together constitute the image file. But you only need to refer to the directory name and you can think of it as one logical file.

Whenever we use the word “image”, we are just using it in a generic sense. CASA images are manipulated with an Image tool associated with, or bound to, the actual image file. Note that some image tools don’t have a disk file associated with them. These are called “virtual” images and are discussed below.

When an image is stored on disk, it can, in principle, be stored in a variety of ways. For example, the image could be stored row by row; this is the way that most older generation packages store images. It makes for very fast row by row access, but very slow in other directions (e.g. extract all the profiles along the third axis of an image). A CASA image file is stored with what is called tiling. This means that small multi-dimensional chunks (a tile) are stored sequentially. It means that row by row access is a little slower, but access speed is essentially the same in all directions. This in turn means that you don’t need to (and can’t!) reorder images.

Here are some simple examples using image tools.

```
# print "\t----	 Intro Ex 1 \t----"
ia.maketestimage('zz', overwrite=true)# Make test image; writes disk file called 'zz'
print ia.summary() # Summarize (to logger)
print ia.statistics() # Evaluate statistics over entire image
box = rg.box([10,10], [50,50]) # Make a pixel box region with regionmanager
im2 = ia.subimage('zz2', box, overwrite=true) # Make a subimage called 'zz2'
print im2.statistics() # Evaluate statistics
print "CLEANING UP OLD zz2.amp/zz2.phase IF THEY EXIST. IGNORE WARNINGS!"
ia.removefile('zz2.amp')
ia.removefile('zz2.phase')
im2.fft(amp='zz2.amp', phase='zz2.phase') # FFT subimage and store amp and phase
```
Foreign Images

The Image tool also provides you with native access to some foreign image formats. Presently, these are FITS (Florar, Double, Short and Long are supported) and Miriad. This means that you don’t have to convert the file to native CASA format in order to access the image. For example:

```
print "\t----\t Intro Ex 2 \t----"
pathname=os.environ.get("CASAPATH") # Assumes environment variable is set
pathname=pathname.split()[0]
datapath1=pathname+"/data/demo/Images/imagetestimage.fits"
datapath2=pathname+"/data/demo/Images/test_image"
ia.open(datapath1) # Access FITS image
#ia.open('im.mir') # Access Miriad image (no image in repository)
ia.open(datapath2) # Access casa image
#
#ims = ia.newimagefromimage(infile=datapath1, region=rg.quarter())
# rg.quarter() not implemented yet so has grabbed entire image
ims = ia.newimagefromimage(infile=datapath1)
innerquarter=rg.box([0.25,0.25],[0.75,0.75],frac=true)
subim = ims.subimage(region=innerquarter)
print ia.name()
print ims.name()
print subim.name()
ims.done() # done on-the-fly image tool
subim.done() # done on-the-fly image tool
ia.close() # close (not done) default image analysis tool
#
```

Each of these Image tools has access to all the same toolfunctions.

Where ever you see an argument in an Image tool function which is an input image disk file, that disk file can be a CASA, FITS, or Miriad image file.

There are some performance penalties that you should be aware of. Firstly, because CASA images are tiled (see above) you get the same access speed regardless of how you access the image. FITS and Miriad images are not tiled. This means that the performance for these Image tools will be poorer for certain operations. For example, extracting a profile along the third axis of an image, or re-ordering an image with the display library.
Secondly, for FITS images, masked values are indicated via “magic value”. This means that the mask is worked out on the fly every time you access the image.

If you find performance is not good enough or you want a writable image, then use appropriate function (fromfits to convert to a native CASA image).

**Virtual Images**

We also have Image tools that are not associated one-to-one with disk files; these are called “virtual” images (see also the article in the AugustNewsLetter). For example, with the image calculator, imagecalc, one can create an expression which may contain many images. You can write the result of the expression out to a disk image file, but if you wish, you can also just maintain the expression, evaluating it each time it is needed - nothing is ever written out to disk in this case. There are other Image functions like this (the documentation for each one explains what it does). The rules are:

- If you specify the outfile argument, then the image is always written to the specified disk image file.

- If you leave the outfile argument unset, then if possible, a virtual image will be created. Sometimes this virtual image will be an expression as in the example above (i.e. it references other images) or a temporary image in memory, or a temporary image on disk. (the summary function will list for you the type of image you have). When you destroy that Image tool, the virtual image will be destroyed as well.

- If you leave outfile unset, and the function cannot make a virtual image, it will create a disk file for you with a name of its choice (usually input plus function name).

- You can always write a virtual image to disk with the subimage tool function.

**Coordinate Systems**

An image contains a Coordinate System. A Coordsys tool is used to manipulate the Coordinate System. An Image tool allows you to recover the Coordinate System into a Coordsys tool through the coordsys function. You can set a new Coordinate System with the setcoordsys function. You can do some direct coordinate conversion via the Image tool functions toworld, topixel, and coordmeasures. The actual work is done by a Coordsys tool, for which these Image tool functions are just wrappers.

**Lattice Expression Language (LEL)**

LEL allows you to manipulate expressions involving images. For example, add this image to that image, or multiply the minimum value of that image by the square root of this image. The LEL syntax is quite rich and is described in detail in note 223. LEL is accessed via the imagecalc and the calc tool functions. Here are some examples.
In this example the Image tool is associated with the non-virtual disk file zz. This image file name is used in an LEL expression.

Note that for image file names with special characters in them (like a dash for example), you should (double) escape those characters or put the file name in double quotes. E.g.

Region-of-interest
A region-of-interest or simply, region, designates which pixels of the image you are interested in for some (generally) astrophysical reason. This complements the pixel mask (see below) which specifies which pixels are good or bad (for statistical reasons). Regions-of-interest are generated and manipulated with the Regionmanager tool. Briefly, a region-of-interest may be either a simple shape such as a multi-dimensional box, or a 2-D polygon, or some compound combination of regions-of-interest. For example, a 2-D polygon defined in the X and Y axes extended along the Z axis, or perhaps a union or intersection of regions. See the Regionmanager documentation for more details on regions. Regions are always supplied to tool functions via the region argument.

Pixel mask
A pixel mask specifies which pixels are to be considered good (value T) or bad (value F). For example, you may have imported a FITS file which has blanked pixels in it. These will be converted into pixel mask elements whose values are bad (F). Or you may have made an error analysis of an image and computed via a statistical test that certain pixels should be masked out for future analysis.

If there is no pixel mask, all pixels are considered good (if you retrieve the pixel mask when there is none, you will get an all good mask). Pixels for which the pixel mask value is bad are not used in computations (e.g. in the calculation of statistics, moments or convolution).

The image may contain zero, one, or more pixel masks. However, only one mask will be designated as the default mask. This is the pixel mask that is actually applied to the data. You can also indicate that none of the pixel masks are the default, so that effectively an all good pixel mask is applied.

The function summary includes in its summary of the image the names of the masks (the first listed, if not in square brackets, is the default).

Pixel masks are handled with the function maskhandler. This allows you to find the names of pixel masks, delete them, copy them, nominate the default and so on. It is not used to change the value of pixel masks.

The functions with which you can change pixel mask values are putregion (put Boolean array), calcmask (put result of Boolean LEL expression), and set (put scalar Boolean).

The argument 'mask'

There is an argument, mask, which can be supplied to many functions. It is supplied with either a mask region-of-interest (generated via the function wmask) or a LEL Boolean expression string (the same string you would have supplied to the above Regionmanager function). Generally, one just supplies the expression string.

The LEL expression is simply used to generate a pixel mask which is then applied in addition to any default pixel mask in the image (a logical OR).

For example

```{}
***
#
print "\t----\t Intro Ex 5 \t----"
ia.maketestimage('zz', overwrite=true)
ia.statistics(mask='zz > 0') # Only evaluate for positive values
ia.calcmask (mask='(2*zz) > 0') # Create a new mask which is T (good)
      # when twice the image values are 
      # positive, else F

ia.close()
#
***
```

The mask expression must in general conform (shape and coordinates) with the image (i.e. that associated with the Image tool).
When `mask` is used with function `calcmask`, a persistent `pixel mask` is created and stored with the image. With all other functions, the `mask` argument operates as a transient (or On-The-Fly [OTF]) `pixel mask`. It can be very handy for analysing or displaying images with different masking criteria. Often I will refer to the “total input mask”. This is the combination (logical OR) of the default `pixel mask` (if any) and the OTF mask (if any).

In the following example we open a Rotation Measure image. We then evaluate statistics and display it where only those pixels whose error in the Rotation Measure (image file `rmerr`) is less than the specified value are shown; the others are masked. The nice thing is you can experiment with different `pixel masks` until you are satisfied, whereupon you might then make the `pixel mask` persistent with the `calcmask` function.

```
# print "\t----\t Intro Ex 6 \t----"
#myim = ia.newimagefromimage('rm')
#myim.statistics(mask='rmerr<10')
#myim.calcmask (mask='rmerr<20') # Make persistent mask
#
```

Finally, a subtlety that is worth explaining.

```
# print "\t----\t Intro Ex 7 \t----"
ia.maketestimage('zz', overwrite=true)
ia.statistics(mask='zz>0') # Mask of zz ignored
da.statistics(mask='mask(zz) && zz>0') # Mask of zz used
ia.close()
#
```

In the first example, any default mask associated with the image `zz` is ignored. Only the pixel values are looked at. In the second example, the mask of `zz` is also taken into account via the LEL `mask` function. That is, the transient output mask is T (good) only when the mask of `zz` is T and the expression `zz>0` is T.
A useful part of LEL to use with the mask argument is the indexin function. This enables the user to specify a mask based upon selected pixel coordinates or indices (specified 0-rel) rather than image values. For example

```
print "\t----\t Intro Ex 8 \t----"
ia.fromshape(shape=[20])
print ia.getregion(mask='indexin(0, [4:9, 14, 18:19])',getmask=true)
# [False False False False True True True True True True False False False False True False False False True True]
ia.close()
```

You can see the mask is good (T) for the specified indices along the specified axis. You can also pass in a premade variable for the specification if you like, viz.

```
print "\t----\t Intro Ex 9 \t----"
ia.fromshape(shape=[20])
axis = "0"
set = "[4:9, 14, 18:19]"
paint atgetregion(mask= \"indexin\"(\"+axis\",\"+set\"),getmask=true)
# [False False False False True True True True True True False False False False True False False False True True]
ia.close()
```

This capability is useful for fitting functions.

**Pixel masks and Regions**

Some comment about the combination of pixel masks and regions-of-interest is useful here. See the Regionmanager tool for basic information about regions-of-interest first. Regions are provided to Image tool functions via the standard region function argument.

Consider a simple polygonal region. This region-of-interest is defined by a bounding box, the polygonal vertices, and a mask called a region mask. The region mask specifies whether a pixel within the bounding box is inside or outside the polygon. For a simple box region-of-interest, there is obviously no need for a region mask.

Now imagine that you wish to recover the pixel mask of an image from a polygonal region-of-interest. The mask is returned to you in regular
Boolean array. Thus, the shape of the returned mask array reflects the bounding-box of the polygonal region. If the actual pixel mask that you apply is all good, then the retrieved mask would be good inside of the polygonal region and bad outside of it. If the actual pixel mask had some bad values in it as well, the retrieved mask would be bad outside of the polygonal region. Inside the polygonal region it would be bad if the pixel mask was bad. More simply put, the mask that you recover is just a logical “and” of the pixel mask and the region mask; if the pixel mask is T and the region mask is T then the retrieved mask is T (good), else it is F (bad). Finally, note that if you use the region and mask (the OTF mask) arguments together then they operate as follows. The shape of the Boolean expression provided by mask must be the same shape as the image to which it is being applied. The region is applied equally to the image and the mask expression. For example

```python
"""
#
print "\t----\t Intro Ex 10 \t----"
#rm1 = ia.newimagefromimage('rm')
#rm2 = ia.newimagefromimage('rmerr')
#rm1.shape()
#[128 128]
#rm2.shape()
#[128 128]
#r = rg.box([10,10], [50,50])
#rm1.statistics(region=r, mask='rmerr<10') # region applied to
# 'rmerr' and 'rm'
#
"""

Methods

- newimage: Construct a new image analysis tool using the specified image. (Also known as newimagefromfile.)
- newimagefromfile: Construct a new image analysis tool using the specified image. (Also known as newimage.)
- imagecalc: Perform mathematical calculations on an image or images.
- collapse: Collapse an image along a specified axis, computing a specified aggregate function of pixels along that axis.
- decimate: Remove planes from an image.
- imageconcat: Construct a CASA image by concatenating images.
- fromarray: Construct a CASA image from a numerical (integer or float) array.
- fromascii: This function converts a pre-existing ascii file into a CASA image.
- fromfits: Construct a CASA image by conversion from a FITS image file.
- fromimage: Construct a (sub)image from a region of a CASA image.
- fromshape: Construct an empty CASA image from a shape.
- maketestimage: Construct a CASA image from a test FITS file.
adddegaxes
  Add degenerate axes of the specified type to the image
addnoise
  Add noise to the image
convolve
  Convolvle image with an array or another image
boundingbox
  Get the bounding box of the specified region
boxcar
  Convolvle one axis of image with a boxcar kernel
brightnessunit
  Get the image brightness unit
calc
  Image calculator
calcmask
  Image mask calculator
close
  Close the image tool
continuumsub
  Image plane continuum subtraction
calculate
  Convert peak intensity to/from flux density for a 2D Gaussian.
convolve2d
  Convolvle image by a 2D kernel
coordsys
  Get the Coordinate System of the image
coordmeasures
  Convert from pixel to world coordinate wrapped as Measures
decompose
  Separate a complex image into individual components
decomconvolvecomponentlist
  Deconvolve a componentlist from the restoring beam
decovolvefrombeam
  Helper function to deconvolve the given source Gaussian from a beam Gaussian
getbeamforconvolvedsize
  Determine the size of the beam necessary to convolve with the given source to reach the convolved size
getcommonbeam
  Determine a beam to which all beams in an image can be convolved.
remove
  Delete the image file associated with this image tool
removefile
  Delete an unattached image file from disk. Note: use remove() if the image file is attached to the image tool
done
  Destroy this image tool
fft
  FFT the image
findsources
  Find point sources in the sky
fitprofile
  Fit gaussians and/or polynomials to a 1-dimensional profile.
fitcomponents
  Fit 2-dimensional models to an image.
fromrecord
  Generate an image from a record
getchunk
  Get the pixel values from a regular region of the image into an array
getregion
  Get pixels or mask from a region-of-interest of the image
getprofile
  Get values and mask for a one dimensional profile along a specified image axis by applying an aggregate function.
getslice
  Get 1-D slice from the image
hanning
  Convolve one axis of image with a Hanning kernel
haslock
  Does this image have any locks set?
histograms
  Compute histograms from the image
history
  Recover and/or list the history file
insert
  Insert specified image into this image
isopen
  Is this Image tool open?
ispersistent
  Is the image persistent?
lock
  Acquire a lock on the image
makecomplex
  Make a complex image
maskhandler
  Mask pixel masks
miscinfo
  Get the miscellaneous information record from an image
modify
  Modify image with a model
maxfit
  Find maximum and do parabolic fit in the sky
moments
  Compute moments from an image
name
  Name of the image file this tool is attached to
Open a new image file with this image tool
Pad the perimeter of the direction plane with a number of pixels of specified value
Crop masked pixels from the perimeter of an image.
Get value of image and mask at specified pixel coordinate
Put pixels from an array into a regular region of the image
Put pixels and mask into a region-of-interest of the image
Rebin an image by the specified integer factors
Regrid this image to the specified Coordinate System
Transpose the image.
Rotate the direction coordinate axes attached to the image and regrid the image to rotate the image’s beam(s) counterclockwise through the specified angle.
Rename the image file associated with this image tool
Replace the values of pixels which are masked bad
Get the beam area.
Get the restoring beam(s).
Separable convolution
Set pixel and/or mask values with a scalar in a region-of-interest of the image
Set the image brightness unit
Set new Coordinate System
Set the history for an image
Set the miscellaneous information record for an image
Length of each axis in the image
Set the restoring beam
Compute statistics from the image
Compute two point correlation function from the image
Create a (sub)image from a region of the image
Summarize basic information about the image
Convert the image to a FITS file
Convert the image to an ASCII file
Return a record containing the image associated with this tool
Return the type of this tool
Convert from world to pixel coordinate
Convert from pixel to world coordinate
Release any lock on the image
Construct a CASA image from an array
Construct a CASA image by conversion from a FITS image file
Construct an on-the-fly image tool from a region of a CASA image file
Construct an empty CASA image from a shape
Construct a primary beam corrected image from an image and a primary beam
Construct a position-velocity image between two points in the direction plane.
Construct an initialized multi-dimensional array.
Returns true of the shape, coordinate system, and axes order of the specified image.
image.newimage.html

**image.newimage - Function**

1.1.1 Construct a new image analysis tool using the specified image. (Also known as newimagefromfile.)

**Description**

This method is identical to ia.newimagefromfile(). The description of how it works is in the online help for that method.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>infile</td>
<td>Input image file name</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

image
Construct a new image analysis tool using the specified image. (Also known as newimage.)

Description

This method returns an image analysis tool associated with the specified image. Constructing a image analysis tool in addition to the default ia tool allows the user to operate on multiple images without having to close one before opening another. All ia.newimagefrom*() methods share this functionality.

The parameter infile may refer to a CASA image, a Miriad image, or a FITS image. FITS images of types Float, Double, Long, and Short are supported. When finished with the newly created tool, the user should close it to free up system resources (e.g., memory).

ia.newimage() is an alias for this method.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>infile</td>
<td>Input image file name</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

image

Example

# This is one way to copy a FITS image into an already extant CASA image
# of the same shape (ia.subimage() is more efficient, but this example is
# meant to demonstrate ia.newimagefromfile())

# note that the ia tool is not attached to an image after the first command,
# the fitsimage tool is
fitsimage = ia.newimagefromfile("myimage.fits")
# now attach the target CASA image to the ia tool
ia.open("myimage.im")
# copy pixel values
ia.putchunk(fitsimage.getchunk())
# copy the coordinate system
ia.setcoordsys(fitsimage.coordsys().torecord())
# copy other miscellaneous things
ia.setbrightnessunit(fitsimage.getbrightnessunit())
ia.setmiscinfo(fitsimage.miscinfo())
# be sure to call done() on both tools to free up memory
ia.done()
fitsimage.done()
image.imagecalc.html

**image.imagecalc - Function**

1.1.1 Perform mathematical calculations on an image or images.

**Description**

This method is used to evaluate a mathematical expression involving existing images. It fully supports both float and complex valued images. The syntax of the expression supplied via the pixels parameter (in what is called the Lattice Expression Language, or LEL) is explained in detail in [note 223](#). This is a rich mathematical language with allows all manner of mathematical operations to be applied to images.

Any image files embedded in the expression may be native CASA or FITS (but not yet Miriad) images.

If successful, this method always returns an image analysis tool that references the image resulting from the calculation. This returned tool should always be captured and closed as soon as the user is done with it to free up system resources (e.g., memory). The image analysis tool on which the method is called (e.g., the ia tool when one runs `ia.imagecalc()`) remains unaltered, e.g., it still refers to the same image it did prior to the `imagecalc()` call.

Values of the returned tool are evaluated "on demand". That is, only when a method is run on the returned tool are the necessary values computed. And in fact, the values have to be reevaluated for each operation (method call). This means that there is a small performance hit for using the returned tool rather than the image written to disk and that none of the images which were used in the expression should be deleted while the returned tool is in use because they must be accessed for calculating the expression each time an operation of the returned tool is performed. These limitations do not apply to the output image if one is specified with the `outfile` parameter; it is a genuine CASA image with numerical values. If `outfile` is blank, no output image is written (although the resulting image can still be accessed via the returned image analysis tool as described below).

Normally you should just write the image, close the returned tool, and open the results image with the default ia tool and operate on it. If you are interested in conserving disk space, you don’t need to keep the result of the calculation around for very long, and/or you are only going to do a small number of operations on the result image, should you set `outfile=""`. Note that when multiple image are used in the expression, there is no guarantee about which of those images will be used to create the metadata of the output image, unless `imagemd` is specified. If `imagemd` is specified, the following rules of metadata copying will be followed:
1. The pixel data type of the image specified by imagemd and the output image must be the same. 2. The metadata copied include the coordinate system (and so of course the dimensionality of the output image must correspond to the coordinate system to be copied), the image_info record (which contains things like the beam(s)), the misc_info record (should one exist in the image specified by imagemd), and the units. 3. If the output image is a spectral image, the brightness units are set to the empty string. 4. If the output image is a polarization angle image, the brightness unit is set to "deg" and the stokes coordinate is set to have a single plane of type of Pangle.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. If blank the resulting image is not written, but it can still be accessed via the returned image analysis tool. Allowed: string Default:</td>
</tr>
<tr>
<td>pixels</td>
<td>LEL expression. Must be specified. For example &quot;my-image1.im + myimage2.im&quot;. Allowed: string Default:</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file? Allowed: bool Default: false</td>
</tr>
<tr>
<td>imagemd</td>
<td>The image from which metadata should be copied. Default means no guarantee from which image is used. Allowed: string Default:</td>
</tr>
</tbody>
</table>

Returns

image

Example

```""" # Suppose aF and bF are images with single precision and we want # to determine the result of the following expression: # aF + min(float($\pi$, mean(bF))) 53```
In this case, the images aF and bF do not need to have the same shapes and coordinates, because only the mean(bF) results in the mean of all pixel values in bF. If aF has single precision pixel values, the resulting image will as well. This expression first computes the scalar value of the minimum of $\pi$ and the mean of the pixel values of bF. That scalar is then added to the value of each pixel in aF. In the code below, the result is written to image cF which can be accessed immediately via the returned image analysis tool captured in the variable myim. If the expression is masked, that mask will be copied to the new image.

```python
def create_images():
    ia.fromshape('aF', [10, 10], overwrite=True)
    ia.fromshape('bF', [10, 20, 30], overwrite=True)
    # close the ia tool to free up resources
    ia.done()
    # at each pixel in bF, take the minimum of that pixel value and pi and add
    # the resulting value to the corresponding pixel in aF
    # note that only the subset of pixels in bF that correspond to those in aF
    # are used; the resulting image has the same size as the smaller image, aF,
    # used in the input
    myim = ia.imagecalc(outfile='cF', pixels='aF + min(float(pi()), mean(bF))',
                        overwrite=True)
    # confirm the resulting image has the same size as aF, should be [10, 10]
    myim.shape()
    # close the myim tool to free up system resources
    myim.done()
```

Example

The following example shows the use of the two min() LEL functions. One takes a single argument and will return a scalar representing the minimum pixel value of that entire image. The other takes two arguments (either an image and a scalar or two images of conforming shapes) and returns an image for which the minimum has been calculated on a pixel by pixel basis for the input image(s).

```python
def example():
    # create an image to operate on
    ia.fromshape('aF', [10, 10], overwrite=True)
```
# give it interesting values
ia.addnoise()

# free up system resources
ia.done()

# do the calculation and write results to image cF
myim = ia.imagecalc('cF', 'min(aF, (min(aF)+max(aF))/2)', overwrite=true)

# do whatever stuff you want with myim and the close it to free
# up system resources
myim.done()

""

Example

""

# Here’s an example of a more complicated function. Currently
# ia.fromshape() only creates real-valued images so the real()
# function is not particularly exciting in this case but illustrates
# possibilities. Trigonometric functions such as sin() assume the
# pixel values are in radians.
ia.fromshape('aD', [10,10], overwrite=true)
ia.addnoise()

ia.fromshape('aF', [10,10], overwrite=true)
ia.addnoise()

ia.fromshape('bF', [10,10], overwrite=true)
ia.addnoise()

ia.fromshape('aC', [10,10], overwrite=true)
ia.addnoise()

ia.done()

myim = ia.imagecalc('eF', 'sin(aD)+(aF*2)+min(bF)+real(aC)', overwrite=true)

myim.done()

"""
image.collapse.html

image.collapse - Function

1.1.1 Collapse an image along a specified axis, computing a specified aggregate function of pixels along that axis.

Description

This method collapses an image along a specified axis or set of axes of length N pixels to a single pixel on each specified axis. Both float valued and complex valued images are supported. It computes a user-specified aggregate function for pixel values along the specified axes, and places those values in the single remaining plane of those axes in the output image. The method returns an image analysis tool containing the newly-created collapsed image. Valid choices of aggregate functions are: 'flux' (see below for constraints), 'max', 'mean', 'median', 'min', 'rms', 'stdev', 'sum' and 'variance'. Minimal unique matching is supported for the function parameter (e.g. function = 'r' will compute the rms of the pixel values, 'med' will compute the median, etc.). If one specifies function=’flux’, the following constraints must be true:

1. The image must have a direction coordinate, 2. The image must have at least one beam, 3. The specified axes must be exactly the direction coordinate axes, 4. Only one of the non-directional axes may be non-degenerate, 5. The image brightness unit must be conformant with x*yJy/beam, where x is an optional unit (such as km/s for moments images) and y is an optional SI prefix.

Axes may be specified as a single integer or an array of integers indicating the zero-based axes along which to collapse the image. Axes may also be specified as a single or array of strings which minimally and uniquely match (ignoring case) world axis names in the image (e.g. ‘dec’ for collapsing along the declination axis or ['ra', 'd'] for collapsing along both the right ascension and declination axes).

If outfile is not specified (or contains only whitespace characters), no image is written but the collapsed image is still accessible via the image analysis tool this method always returns (which references the collapsed image). If the returned object is not wanted, it should still be captured and destroyed via its done() method. If this is not done, there is no guarantee as to when the Python garbage collector will delete it. If the returned object is wanted, it should still be deleted as soon as possible for the same reasons, e.g.

collapsed_image = ia.collapse(...)

# do things (or not) with the collapsed_image and when finished working with the object, do
collapsed_image.done()
The reference pixel of the collapsed axis is set to 0 and its reference value is set to the mean of the the first and last values of that axis in the specified region of the input image. The reference value is the world coordinate value of the reference pixel. For instance, if an axis to be collapsed were to be the frequency axis, in the collapsed image, the reference value would be the mean value of the frequency range spanned, and would be stored in pixel 0.

If the input image has per plane beams, the beam at the origin of the subimage determined by the selected region is arbitrarily made the global beam of the output image. In general, the user should understand the pitfalls of collapsing images with multiple beams (i.e. that employing an aggregate function on pixels with varying beam sizes more often than not leads to ill-defined results). Convolution to a common beam is not performed automatically as part of the preprocessing before the actual rebinning occurs. In such cases, therefore, the user should probably first convolve the input image with a common restoring beam so that each plane has the same resolution, and/or use imsmooth to smooth the data to have the same beam.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>function</td>
<td>Aggregate function to apply. This can be set one of flux, max, mean, median, min, rms, stdev, sum, variance. Must be specified.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>axes</td>
<td>Zero-based axis number (specified as a list or integer) along which to collapse the specified image. Default value is 0.</td>
<td>any</td>
<td>variant 0</td>
</tr>
<tr>
<td>outfile</td>
<td>Output image file name. If left blank (the default), no image is written but a new image tool referencing the collapsed image is returned.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See ”help par.region” for details. Default is to use the full image.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>box</td>
<td>Rectangular region to select in direction plane. See ”help par.box” for details. Default is to use the entire direction plane.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>chans</td>
<td>Channels to use. See ”help par.chans” for details. Channels must be contiguous. Default is to use all channels.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>stokes</td>
<td>Stokes planes to use. See ”help par.stokes” for details. Planes specified must be contiguous. Default is to use all Stokes planes.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask for more details. Default setting is none.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file? Ignored if ”outfile” is left blank.</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>stretch</td>
<td>Stretch the mask if necessary and possible? See help par.stretch. Default value is False.</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>
Returns
image

Example


# myimage.im is a 512x512x128x4 (ra,dec,freq,stokes) image
ia.open("myimage.im")
# collapse a subimage of it along its spectral axis avoiding the 8 edge
# channels at each end of the band, computing the mean value of the pixels
# resulting image is 256x256x1x4 in size.
collapsed = ia.collapse(outfile="collapse_spec_mean.im", function="mean", axes=2, box="127,127,383,383", chans="8~119")
# manipulate collapsed
collapsed.done()


image.decimate.html

**image.decimate - Function**

1.1.1 Remove planes from an image.

**Description**

This application removes planes along the specified axis of an image. It supports both float valued and complex valued images. The factor parameter represents the factor by which to reduce the number of planes. The method parameter represents how to calculate the pixel values of the output image. A value of method="copy" means that every factorth plane of the selected region in the input image will be directly copied to the corresponding plane in the output image. So, if one wanted to copy every third spectral plane in the input image to the output image, one would specify factor=3 and method="copy". If the selected region along the specified axis had 11 planes, then there would be 4 output planes which would map to planes 0, 3, 6, and 9 of the specified region of input image. A value of method="mean" indicates that each of factor number of planes in the range starting at each factorth plane should be averaged to produce the corresponding output plane. So, if one specified factor=3 and method="mean" along an axis of the selected region of the input image which had 11 pixels, the corresponding axis in the output image would have three pixels and the pixel values for each of those output planes would corresponding to averaging along that axis planes 0-2, 3-5, and 6-8 of the selected region of the input image. Note that the remaining planes, 9 and 10, in the selected region of the input image would be ignored because the last interval must have exactly factor number of planes in order to be included in the output image. The coordinate system of the output image takes into account the decimation; that is, along the decimated axis, the increment of the output image is factor times that of the input image, and the reference pixel of the output image is located at pixel 1/factor times the reference pixel in the input image. This method returns an image analysis tool which references the output image. If this tool is not desired, one should capture it anyway and then close() it immediately to free up resources. Images with multiple beams are not supported; please convolve a multi-beam image to a single resolution before running this application.

**Arguments**

60
Inputs

outfile  Output image file name. If empty, a persistent image is not created.
  allowed:  string
  Default:
axis  Axis along which to remove planes.
  allowed:  int
  Default:  0
factor  Reduce number of planes by this factor.
  allowed:  int
  Default:  1
method  Method to use for calculating pixel values of output. Supported values are "copy" or "mean".
  allowed:  string
  Default:  copy
region  Region selection. See "help par.region" for details. Default is to use the full image.
  allowed:  any
  Default:  variant
mask  Mask to use. See help par.mask for more details. Default setting is none.
  allowed:  string
  Default:
overwrite  Overwrite (unprompted) pre-existing output file? Ignored if "outfile" is left blank.
  allowed:  bool
  Default:  false
stretch  Stretch the mask if necessary and possible? See help par.stretch. Default value is False.
  allowed:  bool
  Default:  false

Returns

image

Example

# Copy verbatim every 5th plane of axis 2 of the input image
ia.open("myim.im")
decimated = ia.decimate("decl.im", axis=2, factor=5, method="copy")

61
# do stuff with decimated and then close it
decimated.close()

# Decimate by averaging every 7 planes of the input image along axis 2
decimated = ia.decimate("dec2.im", axis=2, factor=7, method="mean")
# do stuff with decimated and then close it
decimated.close()
image.imageconcat.html

**image.imageconcat - Function**

1.1.1 Construct a CASA image by concatenating images

**Description**

This function is used to concatenate two or more input CASA images into one output image. For example, if you have two image cubes which are contiguous along one axis (say a spectral axis) and you would like to glue them together along this axis, then this function is the appropriate thing to use.

The axis parameter is used to specify which zero-based axis the images should be concatenated along. A negative value indicates that the spectral axis should be used. If a negative value is given but there is no spectral axis, an exception will be thrown. The zero-based order of the axes of an image can be determined from `ia.coordsys().names()`.

If successful, this method will return an image analysis tool referencing the concatenated image. Even if it is not wanted, the returned tool should be captured and closed as soon as the user is finished with it to free up system resources (e.g., memory).

If `outfile` is given, the image is written to the specified disk file. If `outfile` is unset, the on-the-fly Image tool created by the function actually references all of the input files. So if you deleted any of the input image disk files, it would render this tool useless. When you destroy this tool (with the done function) the reference connections are broken.

The input and output images must be of the same dimensionality. Therefore, if you wish to concatenate 2-D images into a 3-D image, the 2-D images must have a third axis (of length unity) so that the output image coordinates are known along the concatenation axis.

The input images are concatenated in the order in which they are listed unless the reorder parameter is set to True. If True, the images are reordered if necessary so that the world coordinate values along the selected axis monotonically increase or decrease. The direction of the increment is determined by the first listed image. If reorder=True, the world coordinate ranges of the images along the selected axis are not permitted to overlap, and the signs of the increments for this axis in all images must be the same. If reorder=False, the coordinate system of the first listed image is used as the coordinate system for the output image. If reorder=True, the coordinate system of the first image in the list of the reordered images is used as the coordinate system of the output image. Setting reorder=True can be especially useful if the infiles are specified using a wildcard character(s).
If relax=False, the input images are checked to see that they are contiguous along the concatenation axis and an error is generated if they are not. In addition, the coordinate descriptors (e.g. reference pixel, reference value etc) for the non-concatenation axes must be the same or an error will result. The input disk image files may be in native CASA, FITS, or Miriad formats. The contiguous criterion and coordinate descriptor equality criteria can be relaxed by setting relax=T whereupon only warnings will be issued. Dimension and shape must still be the same though. When the concatenation axis is not contiguous (but still monotonically increasing or decreasing) and relax=T, a tabular coordinate will be used to correctly describe the axis. But be aware that it means adjacent pixels are not regularly spaced. However, functions like toworld and topixel will correctly interconvert world and pixel coordinates.

In giving the input image names, the infiles argument can be a single string if you wild card it with standard shell symbols. For example, infiles='cena.???.*', where the "?" represents one character and "*" any number of characters. Otherwise, you must input a vector of strings such as infiles="cena1 cena2 cena3". An input such as infiles='files1,file2' will be interpreted as one string naming one file and you will get an error. The reason for this is that although the latter could be parsed to extract two file names by recognizing comma delimiters, it is not possible because an expression such as infiles='cena.{a,b}' (meaning files of name “cena.a” and “cena.b”) would confuse such parsing (you would get two files of name cena.{a} and {b}.

You can look at the coordinate system of the output image using the ia.summary() tool method to ensure it’s correct.

The argument tempclose is, by default, True. This means that all internal reference copies of the input images are kept closed until they are needed. Then they are opened temporarily and then closed again. This enables you to effectively concatenate as many images as you like without encountering any operating system open file number limits. However, it comes at some performance loss, because opening and closing all those files takes time. If you are concatenating a smallish number of files, you might use tempclose=F. This will leave all internal reference copies permanently open, but performance, if you don’t hit the file limit, will be better.

This method requires multiple images which are specified with the infiles parameter. Therefore calling ia.open() is not necessary, although calling imageconcat() using an already open image analysis tool will work and the state of that tool (eg the image it references) will not be changed.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
</tr>
<tr>
<td>allowed: string</td>
<td>Default:</td>
</tr>
<tr>
<td>infiles</td>
<td>List of input CASA image files to concatenate; wild cards accepted. Default is empty string.</td>
</tr>
<tr>
<td>allowed: any</td>
<td>Default: variant</td>
</tr>
<tr>
<td>axis</td>
<td>Concatenation pixel axis. Use ia.coordsys().names() to get a list of axes. A negative value means use the spectral axis if there is one, if not an exception is thrown.</td>
</tr>
<tr>
<td>allowed: int</td>
<td>Default: -1</td>
</tr>
<tr>
<td>relax</td>
<td>Relax constraints that axis coordinate descriptors match</td>
</tr>
<tr>
<td>allowed: bool</td>
<td>Default: false</td>
</tr>
<tr>
<td>tempclose</td>
<td>Keep all lattices closed until needed</td>
</tr>
<tr>
<td>allowed: bool</td>
<td>Default: true</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file?</td>
</tr>
<tr>
<td>allowed: bool</td>
<td>Default: false</td>
</tr>
<tr>
<td>reorder</td>
<td>Automatically reorder the images if necessary.</td>
</tr>
<tr>
<td>allowed: bool</td>
<td>Default: false</td>
</tr>
</tbody>
</table>

Returns
image

Example

```python
# Create three images to concatenate together.
ia.fromshape('im.1',[10,10,10],overwrite=T)
ia.fromshape('im.2',[10,10,10],overwrite=T)
ia.fromshape('im.3',[10,10,10],overwrite=T)
ia.done()
# now concatenate.
# The three images have the same shape along the axes not to be
```
# concatenated as they must. relax=T means that the contiguity
# constraint along the concatenated axis is not imposed (if it were
# the call would fail because the spectral axes of the input images
# are not contiguous).
bigim = ia.imageconcat(outfile='bigimage', infiles='im.1 im.2 im.3',
                        axis=2, relax=T, tempclose=F, overwrite=T)

# be sure to call done() on the return tool to free up system resources.
bigim.done()

Example

"""
# All images whose file names begin with \(\text{sff im.}\) that reside in
# the current directory are concatenated along the spectral axis if
# there is one. All image coordinate descriptors must match. If any
# input image does not have a spectral axis an error will
# result. Because an outfile is not specified, the returned image analysis
# tool captured in the variable named bigim just references the input images;
# this call does not create a persistent result image.
bigim = ia.imageconcat(infiles="im.*",relax=T)
bigim.done()
"""
Construct a CASA image from a numerical (integer or float) array

Description

This function converts a numerical (integer or float) numpy array of any size and dimensionality into a CASA image. It will create both float and complex valued images.

The image analysis tool on which this method is called will reference the created image; if this tool referenced another image before this call, that image will no longer be referenced by the tool after the creation of the new image. If you would rather have a new image analysis tool returned, keeping the one on which this method was called unaltered, use newimagefromarray() instead. If outfile is given, the image is written to disk, if not, the image tool on which this method was called will reference a temporary image (either in memory or on disk, depending on its size) that will be deleted when the tool is closed.

Float valued images are produced from real-valued arrays. Complex-valued images are produced from complex-valued arrays.

The coordinate system, provided as a coordsys tool converted to a record is optional. If you provide it, it must have the same number of dimensions as the pixels array (see also coordsys). Call the naxes() method on the coordinate system tool to see how many dimensions the coordinate system has. A coordinate system can be created from scratch using the coordinate system (cs) tool and methods therein, but often users prefer to use a coordinate system from an already existing image. This can be gotten using ia.coordsys() which returns a coordinate system tool. A torecord() call on that tool will result in a python dictionary describing the coordinate system which is the necessary format for the csys input parameter of ia.fromarray().

If csys is not specified, a default coordinate system is created. If linear=F (the default) the created coordinate system will have standard RA/DEC/Stokes/Spectral Coordinate axes depending upon the shape of the pixels array (Stokes axis must be no longer than 4 pixels and you may find the spectral axis preceding the Stokes axis if say, shape=[64,64,32,4]). Extra dimensions are given linear coordinates. If linear=T, then all the resulting coordinates are linear with the axes represent lengths. In this case each axis will have a value of 0.0 at its center pixel. The increment of each axis will be 1.0 km.

The method returns True if creation of the image was successful, False otherwise, so you can check programmatically if the image creation was successful.
Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>pixels</td>
<td>Numeric array</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
<tr>
<td>csys</td>
<td>Coordinate System. Default is unset.</td>
</tr>
<tr>
<td></td>
<td>allowed: record</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>linear</td>
<td>Make a linear Coordinate System if csys not given</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: false</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file?</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: false</td>
</tr>
<tr>
<td>log</td>
<td>Write image creation messages to logger</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: true</td>
</tr>
</tbody>
</table>

Returns

bool

Example

```python
# make an image with a default RA/Dec/Stokes/Frequency coordinate system
# having all pixels set to 2.5.
ary = ia.makearray(v=2.5, shape=[64, 64, 4, 128])
# the ia tool does not need to reference an image in this case (ie open())
# need not have been called), if it does reference another image, that reference
# will be lost and replaced with a reference to the newly created image.
res = ia.fromarray(outfile='test.data', pixels=ary, overwrite=true)
if res:
    # perform operations on the newly created image if desired and make sure
    # to close it when done to free up system resources (eg memory)
    ia.shape()
```
Example

```python
# create an image using the coordinate system from another image
ia.open("myexistingimage.im")

mycs = ia.coordsys()
# the number of dimensions in the array and the coordinate system must
# be the same. For this example to work, mycs.naxes() must return 4.
ia.done()
ary = ia.makearray(v=2.5, shape=[64, 64, 4, 128])
res = ia.fromarray(pixels=ary, csys=mycs.torecord())
mycs.done()
if (res):
    # do things with the newly created temporary image before closing it
    ia.shape()
ia.done()
```

```
image.fromascii.html

**image.fromascii - Function**

1.1.1 This function converts a pre-existing ascii file into a CASA image.

**Description**

This function is used to create a CASA image from a pre-existing ASCII file. You might want to use this if you just want to create a quick image to use to see what various image analysis methods do. The image analysis tool on which the method is called will always reference the created image if this method is successful. Thus, calling open() on that tool is not necessary, but if the tool is already open, referencing another image, that reference will be silently destroyed and replaced with a reference to the image created by fromascii(). If outfile is given, the image is also written to the specified disk file. If outfile is unset, the image analysis tool on which this method was called references a temporary image. This temporary image may be in memory or on disk, depending on its size. When you close the image analysis tool (with the close function) the temporary image is deleted.

You must specify the shape of the image. The image must be stored in the ascii file. If the shape of the image having N axes is to be \([s_0, s_1, s_2, ..., s_{(N-1)}]\), where \(s_i\) is the integral number of pixels along axis number \(i\), the file must have \((s_1 * s_2 * ... * s_{(N-1)})\) rows and each row must have \(s_0\) numerical values delimited by the value specified by the sep parameter. Pixel locations are incremented by row number in such a way that the second axis changes fastest. As an example, say we want to create an image with shape\([3,4,2]\). There must be \(4^*2 = 8\) rows in the ascii file, and each row must have 3 space-delimited numerical values. The first row represents values of pixels \([0, 0, 0], [1, 0, 0], [2, 0, 0]\). The second row represents values of pixels \([0, 1, 0], [1, 1, 0], [2, 1, 0]\). The fifth row represents values of pixels \([0, 0, 1], [1, 0, 1], [2, 0, 1]\). The sixth values of pixels \([0, 1, 1], [1, 1, 1], [2, 1, 1]\). And so on.

To further illustrate, say this is our file:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
```

When read with ia.fromascii(), ary = ia.getchunk() would return the following array:

```
ary[0, 0, 0] = 1 ary[1, 0, 0] = 2 ary[2, 0, 0] = 3 ary[0, 1, 0] = 4 ary[1, 1, 0] = 5 ary[2, 1, 0] = 6 ary[0, 2, 0] = 7 ary[1, 2, 0] = 8 ary[2, 2, 0] = 9 ary[0, 3, 0] = 10 ary[1, 3, 0] = 11 ary[2, 3, 0] = 12 ary[0, 0, 1] = 13 ary[1, 0, 1] = 14 ary[2, 0, 1] = 15 ary[0, 1, 1] = 16 ary[1, 1, 1] = 17 ary[2, 1, 1] = 18 ary[0, 2, 1] = 19 ary[1, 2, 1] = 20 ary[2, 2, 1] = 21 ary[0, 3, 1] = 22 ary[1, 3, 1] = 23 ary[2, 3, 1] = 24
```

70
The coordinate system, provided as a coordsys tool converted to a record with coordsys torecord, is optional. If you provide it, it must be dimensionally consistent with the pixels array you give (see also coordsys). If you don’t provide the coordinate system, a default coordinate system is made for you. If linear=F (the default) then it is a standard RA/DEC/Stokes/Spectral Coordinate System depending exactly upon the shape of the pixels array (Stokes axis must be no longer than 4 pixels and you may find the spectral axis coming out before the Stokes axis if say, shape=[64,64,32,4]). Extra dimensions are given linear coordinates. If linear=T, then all axes are linear in the resulting coordinate system.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
<td>string</td>
<td>unset</td>
</tr>
<tr>
<td>infile</td>
<td>Input ascii disk file name. Must be specified.</td>
<td>string</td>
<td>unset</td>
</tr>
<tr>
<td>shape</td>
<td>Shape of image. Must be specified.</td>
<td>intArray</td>
<td>-1</td>
</tr>
<tr>
<td>sep</td>
<td>Separator in ascii file. Default is space character.</td>
<td>string</td>
<td>:</td>
</tr>
<tr>
<td>csys</td>
<td>Coordinate System record from coordsys torecord(). Default is unset.</td>
<td>record</td>
<td>unset</td>
</tr>
<tr>
<td>linear</td>
<td>Make a linear Coordinate System if csys not given</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file?</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

Returns

bool

Example


# say the above file presented above is stored in a file 
# named image.txt
ia.fromascii(outfile="myimage.im", infile='image.txt', shape=[3,4,2], sep=" ", overwrite=true) 
# should return [3, 4, 2]
ia.shape()
# call other ia methods then close the tool
ia.done()

"""
image.fromfits.html

**image.fromfits - Function**

1.1.1 Construct a CASA image by conversion from a FITS image file

**Description**

This function is used to convert a FITS disk image file (Float, Double, Short, Long are supported) to an CASA image file. If outfile is given, the image is written to the specified disk file. If outfile is unset, the Image tool is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you close the Image tool (with the close function) this temporary image is deleted.

This function reads from the FITS primary array (when the image is at the beginning of the FITS file; whichhdus=0), or an image extension (when the image is elsewhere in the FITS file, whichhdus > 0).

By default, any blanked pixels will be converted to a mask value which is false, and a pixel value that is NaN. If you set zeroblanks=T then the pixel value will be zero rather than NaN. The mask will still be set to false. See the function replacemaskedpixels if you need to replace masked pixel values after you have created the image.

**Arguments**

73
### Inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>DEFAULT:</td>
<td></td>
</tr>
<tr>
<td>infile</td>
<td>Input FITS disk file name. Must be specified.</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>DEFAULT:</td>
<td></td>
</tr>
<tr>
<td>whichrep</td>
<td>If this FITS file contains multiple coordinate representations, which one should we read (0-based).</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>DEFAULT:</td>
<td>0</td>
</tr>
<tr>
<td>whichhdu</td>
<td>If this FITS file contains multiple images, which one should we read (0-based).</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>DEFAULT:</td>
<td>0</td>
</tr>
<tr>
<td>zeroblanks</td>
<td>If there are blanked pixels, set them to zero instead of NaN</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>DEFAULT:</td>
<td>false</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>DEFAULT:</td>
<td>false</td>
</tr>
</tbody>
</table>

### Returns

bool

### Example

```python
""
#
print "\t----\t fromfits Ex 1 \t----"
datafile=pathname+'/data/demo/Images/imagetestimage.fits'
ia.fromfits('./myimage', datapath, overwrite=true)
print ia.summary()
s = ia.miscinfo()
print s.keys() # prints any unrecognized field names
ia.close()
# ""
```

74
The FITS image is converted to a \casa\ \imagefile\ and access is provided via the default \imagetool\ called {stf ia}. Any FITS header keywords which were not recognized or used are put in the miscellaneous information bucket accessible with the miscinfo function. In the example we list the names of the fields in this record.
image.fromimage.html

**image.fromimage - Function**

1.1.1 Construct a (sub)image from a region of a CASA image

**Description**

This function applies a region-of-interest to an image file, creates a new image file containing the (sub)image, and associates the image tool with it. The input image file may be in native CASA, FITS, or Miriad format. Look here for more information on foreign images.

If `outfile` is given, the (sub)image is written to the specified disk file.

If `outfile` is unset, the Image tool actually references the input image file. So if you deleted the input image disk file, it would render this tool useless.

When you close this tool (with the close function) the reference connection is broken.

Sometimes it is useful to drop axes of length one (degenerate axes). Use the `dropdeg` argument if you want to do this.

The output mask is the combination (logical OR) of the default input pixel mask (if any) and the OTF mask. Any other input pixel masks will not be copied. Use function maskhandler if you need to copy other masks too.

See also the subimage function.

**Arguments**
Inputs

outfile  Output sub-image file name. Default is unset.
  allowed:  string
  Default:

infile  Input image file name. Must be specified.
  allowed:  string
  Default:

region  Region selection. See "help par.region" for details. Default is to use the full image.
  allowed:  any
  Default:  variant

mask  Mask to use. See help par.mask. Default is none.
  allowed:  any
  Default:  variant

dropdeg  Drop degenerate axes
  allowed:  bool
  Default:  false

overwrite  Overwrite (unprompted) pre-existing output file?
  allowed:  bool
  Default:  false

Returns

bool

Example

```python
# print "\t----\t fromimage Ex 1 \t----"
innerquarter = rg.box([0.25,0.25],[0.75,0.75],frac=true)
ia.close()
ia.fromimage(outfile='image.small', infile='test.data', region=innerquarter, overwrite=true)
ia.close()
# ***
```

The specified \region\ takes a quarter by area of the first two axes of the image, and all pixels of other axes.
image.fromshape.html

image.fromshape - Function

1.1.1 Construct an empty CASA image from a shape

Description

This function creates a CASA image file with the specified shape. All the pixel values in the image are set to 0. One may create either an image with float valued pixels (type='f') or a complex valued image (type='c'). If outfile is given, the image is written to the specified disk file. If outfile is unset, the Image tool is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you close the Image tool (with the close function) this temporary image is deleted. The Coordinate System, provided as a Coordsys tool, is optional. If you provide it, it must be dimensionally consistent with the shape that you specify (see also coordsys). If you don’t provide the Coordinate System, a default Coordinate System is made for you. If linear=F (the default) then it is a standard RA/DEC/Stokes/Spectral Coordinate System depending exactly upon the shape (Stokes axis must be no longer than 4 pixels and you may find the spectral axis coming out before the Stokes axis if say, shape=[64,64,32,4]). Extra dimensions are given linear coordinates. If linear=T then you get a linear Coordinate System. The method returns True if successful, False otherwise.

Arguments
Inputs

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Name of output image file. Default is unset.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>shape</td>
<td>Shape of image. Must be specified.</td>
<td>intArray</td>
<td>0</td>
</tr>
<tr>
<td>csys</td>
<td>Coordinate System. Default is unset.</td>
<td>record</td>
<td></td>
</tr>
<tr>
<td>linear</td>
<td>Make a linear Coordinate System if csys not given?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>log</td>
<td>Write image creation messages to logger</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>type</td>
<td>Type of image. 'f' means Float, 'c' means complex.</td>
<td>string</td>
<td>f</td>
</tr>
</tbody>
</table>

Returns

bool

Example

```python
print "\t----\t fromshape Ex 1 \t----"
ia.fromshape('test2.data', [64,64,128], overwrite=true)
mycs = ia.coordsys(axes=[0,2])
ia.close()
ia.fromshape(shape=[10, 20], csys=mycs.torecord())
mycs.done()
ia.close()
#
"""
The first example creates a zero-filled image named test.data of shape [64,64,128]. If you examine the header with ia.summary() you will see the RA/DEC/Spectral coordinate information.

In the second example, a Coordinate System describing the first and third axes of the image test.data is created and used to create a 2D temporary image.
Construct a CASA image from a test FITS file

This function converts a FITS file resident in the CASA system into a CASA image. If outfile is given, the image is written to the specified disk file. If outfile is unset, the Image tool is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you close the Image tool (with the close function) this temporary image is deleted.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file?</td>
</tr>
</tbody>
</table>

Returns

bool

Example

```python
print "\t---- maketestimage Ex 1 \t----"
ia.maketestimage()  # make virtual image
ia.close()
ia.maketestimage('tmp', overwrite=true)
ia.close()  # close to unlock disk image
```
image.adddegaxes.html

**image.adddegaxes - Function**

| L1.1 | Add degenerate axes of the specified type to the image |

**Description**

This method adds degenerate axes (i.e. axes of length 1) of the specified type. Sometimes this can be useful although you will generally need to modify the coordinate system of the added axis to give it the coordinate you want (do this with the Coordsys tool). This method supports both float and complex valued images.

You specify which type of axes you want to add. You can’t add an axis type that already exists in the image. For the Stokes axis, the allowed value (a string such as I, Q, XX, RR) can be found in the Coordsys newcoordsys function documentation.

If outfile is given, the image is written to the specified disk file. If outfile is unset, the on-the-fly Image tool returned by the function is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you destroy the generated Image tool (with the done function) this temporary image is deleted.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>direction</td>
<td>Add direction axes?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>spectral</td>
<td>Add spectral axis?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>stokes</td>
<td>Add Stokes axis?</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>linear</td>
<td>Add linear axis?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>tabular</td>
<td>Add tabular axis?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>silent</td>
<td>Skip silently existing axes?</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

image

**Example**

```python
# print "\t----\t adddegaxes Ex 1 \t----"
ia.maketestimage()
print ia.shape()
# [113L, 76L]
mycs=ia.coordsys()
print mycs.axiscoordinatetypes()
# ['Direction', 'Direction']
```
mycs.done()
im2 = ia.adddegaxes(spectral=T)
print im2.shape()
# [113L, 76L, 1L]
mycs = im2.coordsys()
print mycs.axiscoordinatetypes()
['Direction', 'Direction', 'Spectral']
mycs.done()
im3 = im2.adddegaxes(stokes='Q')
print im3.shape()
# [113L, 76L, 1L, 1L]
mycs = im3.coordsys()
print mycs.axiscoordinatetypes()
# ['Direction', 'Direction', 'Spectral', 'Stokes']
mycs.done()
im2.done()
im3.done()
ia.close()
#

In this example, all the images are virtual (temporary images).
**image.addnoise - Function**

1.1.1 Add noise to the image

**Description**

This function adds noise to the image. You may zero the image first before the noise is added if you wish.

The noise can be drawn from one of many distributions.

For each distribution, you must supply the type via the `type` argument (minimum match is active) and parameters via the `pars` argument. Briefly:

- **binomial** – the binomial distribution models successfully drawing items from a pool. Specify two parameters, \( n \) and \( p \), respectively. \( n \) is the number of items in the pool, and \( p \), is the probability of each item being successfully drawn. It is required that \( n > 0 \) and \( 0 \leq p \leq 1 \).

- **discreteuniform** – models a uniform random variable over the closed interval. Specify two parameters, the low and high values, respectively. The low parameter is the lowest possible return value and the high parameter is the highest. It is required that \( \text{low} < \text{high} \).

- **erlang** – Specify two parameters, the mean and variance, respectively. It is required that the mean is non-zero and the variance is positive.

- **geometric** – Specify one parameter, the probability. It is required that \( 0 \leq \text{probability} < 1 \).

- **hypergeometric** – Specify two parameters, the mean and the variance. It is required that the variance is positive and that the mean is non-zero and not bigger than the square-root of the variance.

- **normal** – Specify two parameters, the mean and the variance. It is required that the variance is positive.

- **lognormal** – Specify two parameters, the mean and the variance. It is required that the supplied variance is positive and that the mean is non-zero.

- **negativeexponential** – Supply one parameter, the mean.

- **poisson** – Specify one parameter, the mean. It is required that the mean is non-negative.
• **uniform** – Model a uniform random variable over a closed interval. Specify two parameters, the low and high values. The low parameter is the lowest possible return value and the high parameter can never be returned. It is required that \( low < high \).

• **weibull** – Specify two parameters, alpha and beta. It is required that the alpha parameter is not zero.

The random number generator seeds may be specified as an array of integers. Only the first two values are used. If none or a single value is provided, the necessary remaining value(s) are generated based on the current time, using the algorithm

\[
\begin{align*}
\text{seedBase} &= 1e7 \times \text{MJD} \\
\text{seed}[1] &= \text{Int}(\text{seedBase}) \\
\text{# and if seed}[0] \text{ is also not supplied} \\
\text{seed}[0] &= \text{Int}(1e7 \times (\text{seedBase} - \text{seed}[1]))
\end{align*}
\]

where MJD is the Modified Julian Day.

### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>type</strong></td>
<td>Type of distribution, normal</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>normal</td>
</tr>
<tr>
<td><strong>pars</strong></td>
<td>Parameters of distribution</td>
</tr>
<tr>
<td>allowed:</td>
<td>doubleArray</td>
</tr>
<tr>
<td>Default:</td>
<td>0.0 1.0</td>
</tr>
<tr>
<td><strong>region</strong></td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
<tr>
<td><strong>zero</strong></td>
<td>Zero image first?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td><strong>seeds</strong></td>
<td>Seeds to use for the random number generator. If not specified, seeds are created based on the current time.</td>
</tr>
<tr>
<td>allowed:</td>
<td>intArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

### Returns

bool
Example

```python
ia.maketestimage()
ia.addnoise(type='normal', pars=[0.5, 1], zero=T, seeds=[1,2])
ia.statistics()
ia.close()
```
image.convolve.html

**image.convolve - Function**

Convolve image with an array or another image

**Description**

This function performs Fourier-based convolution of an image file by the given kernel. If outfile is given, the image is written to the specified disk file. If outfile is left unset, the on-the-fly Image tool generated by this function is associated with a temporary image. This temporary image may be stored in memory or on disk, depending on its size. When the user destroys the generated Image tool (with the done function) this temporary image is deleted.

The kernel is provided as a multi-dimensional array or as the filename of a disk-based image file. The provided kernel can have fewer dimensions than the image being convolved. In this case, it will be padded with degenerate axes. An error will result if the kernel has more dimensions than the image. No additional scaling of the kernel is provided yet.

The scaling of the output image is determined by the argument scale. If this is left unset, then the kernel is normalized to unit sum. If scale is not left unset, then the convolution kernel will be scaled (multiplied) by this value. Masked pixels will be assigned the value 0.0 before convolution.

The output mask is the combination (logical OR) of the default input pixel mask (if any) and the OTF mask. Any other input pixel masks will not be copied. The function maskhandler should be used if there is a need to copy other masks too.

See also the other convolution functions: convolve2d, sepconvolve and hanning.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>kernel</td>
<td>Convolution kernel - An array or an image filename. Must be specified by the</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
<tr>
<td>scale</td>
<td>Scale factor. The default behavior is to autoscale (specified with -1.0).</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>-1.0</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the</td>
</tr>
<tr>
<td>allowed:</td>
<td>variant</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask for more details. The default setting is none.</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) the pre-existing output file?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>stretch</td>
<td>Stretch the mask if necessary and possible? See help par.stretch for more</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**
image

**Example**

```bash
""
#
print "\t----\t convolve Ex 1 \t----"
# This example presupposes the existence of an input image file, testdata, and a kernel image
```
# Open the input image file:
    ia.open(infile='testdata')
# Set up a region to be operated upon (in this case, the whole image):
    r1 = rg.box()
# Perform the convolution:
    im2 = ia.convolve (outfile = 'convout', overwrite = true, region = r1, kernel = 'kerneldata')
    ia.close()
    im2.done()

#
print "\t----\t convolve Ex 2 \t----"
# This example uses an array as the convolution kernel, and presupposes the existence of an
# Next, create a Python array of some kind to use as a convolution kernel. For example:
    from numpy import arange
    kernelarray = arange(10)**3
# Set up a region to be operated upon (in this case, the whole image):
    r1 = rg.box()
# Perform the convolution:
    im3 = ia.convolve (outfile = 'convout2', overwrite = true, region = r1, kernel = kernelarray)
    ia.close()
    im3.done()
#"
image.boundingbox.html

**image.boundingbox - Function**

1.1.1 Get the bounding box of the specified region

**Description**

This function finds the bounding box of a region of interest when it is applied to a particular image. Both float and complex valued images are supported. It is returned in a record which has fields ‘blc’, ‘trc’, ‘inc’, ‘bbShape’, ‘regionShape’, ‘imageShape’, ‘blcf’ and ‘trcf’ containing the bottom-left corner, the top-right corner (in absolute image pixel coordinates), the increment (stride) of the region, the shape of the bounding box, the shape of the region, the shape of the image, the blc in formatted absolute world coordinates and the trc in formatted absolute world coordinates, respectively. Note that the shape of the bounding box will be different from the shape of the region if a non-unit stride (increment) is involved (see the example below). Note that the integer size of the elements in blc, trc, inc, regionShape, bbShape, and imageShape are 32 bits, even on a 64 bit machine. This means that, on 64 bit machines, you may have to convert them to 64 bit ints using eg numpy.int64, before being able to use them as direct input to other methods such as ia.getchunk().

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**
Example

```python
print "\t----\t boundingbox Ex 1 \t----"
ia.maketestimage()  # Create image tool
x=['3pix','6pix','9pix','6pix','5pix','5pix','3pix']  # X vector in abs pixels
y=['3pix','4pix','7pix','9pix','7pix','5pix','5pix']  # Y vector in abs pixels
mycs = ia.coordsys()
ri=rg.wpolygon(x=x,y=y,csys=mycs.torecord())  # Create polygonal world region
mycs.done()
bb = ia.boundingbox(ri)  # Find bounding box
print bb
#{'regionShape': array([7, 7]), 'trc': array([9, 9]),
# 'imageShape': array([113, 76]),
# 'blcf': '00:00:27.733, -00.06.48.000',
# 'trcf': '00:00:24.533, -00.05.36.000', 'bbShape': array([7, 7]),
# 'blc': array([3, 3]), 'inc': array([1, 1])}
ia.close()
#
```

```python
print "\t----\t boundingbox Ex 2 \t----"
ia.maketestimage()
b = rg.box([10,10],[20,20],[2,3])
print ia.boundingbox(b)
#{'regionShape': array([6, 4]), 'trc': array([20, 19]),
# 'imageShape': array([113, 76]),
# 'blcf': '00:00:24.000, -00.05.24.000',
# 'trcf': '00:00:18.667, -00.03.36.000', 'bbShape': array([11, 10]),
# 'blc': array([10, 10]), 'inc': array([2, 3])}
ia.close()
#
```
In this example we see the difference between bbShape and regionShape because of the increment (stride). See also that the trc is modified by the increment.
image.boxcar.html

image.boxcar - Function

1.1.1 Convolve one axis of image with a boxcar kernel

Description

This application performs boxcar convolution of one axis of an image defined by

\[ z[i] = \frac{(y[i] + y[i+1] + \ldots + y[i+w])}{w} \]

where \( z[i] \) is the value at pixel \( i \) in the box car smoothed image, \( y[k] \) is the pixel value of the input image at pixel \( k \), and \( w \) is a positive integer representing the width of the boxcar in pixels. Both float and complex valued images are supported. The length of the axis along which the convolution is to occur must be at least \( w \) pixels in the selected region, unless decimation using the mean function is chosen in which case the axis length must be at least \( 2w \) (see below). Masked pixel values are set to zero prior to convolution. All nondefault pixel masks are ignored during the calculation. The convolution is done in the image domain (i.e., not with an FFT).

If drop=False (no decimation), the length of the output axis will be equal to the length of the input axis - \( w + 1 \). The pixel mask, ORed with the OTF mask if specified, is copied from the selected region of the input image to the output image. Thus for example, if the selected region in the input image has six planes along the convolution axis, if the specified boxcar width is 2, and if the pixel values, which are all unmasked, on a slice along this axis are \([1, 2, 5, 10, 17, 26]\), then the corresponding output slice will be of length five and the output pixel values will be \([1.5, 3.5, 7.5, 13.5, 21.5]\).

If drop=True and dmethod="copy", the output image is the image calculated if drop=True, except that only every \( w \)th plane is kept. Both the pixel and mask values of these planes are copied directly to the output image, without further processing. Thus for example, if the selected region in the input image has six planes along the convolution axis, the boxcar width is chosen to be 2, and if the pixel values, which are all unmasked, on a slice along this axis are \([1, 2, 5, 10, 17, 26]\), the corresponding output pixel values will be \([1.5, 7.5, 21.5]\).

If drop=True and dmethod="mean", first the image described in the drop=False case is calculated. Then, the \( ith \) plane of the output image is calculated by averaging the \( i*w \) to the \((i+1)*w-1\) planes of this intermediate image. Thus, for example, if the selected region in the input image has six planes along the convolution axis, the boxcar width is chosen to be 2, and if the pixel values, which are all unmasked, on a slice along this axis are \([1, 2, 5, 10, 17, 26]\), then the corresponding output pixel values will be \([2.5, 10.5]\). Any pixels at the end of the plane of the intermediate image that do not fall into a
complete bin of width \( w \) are ignored. Masked values are taken into consideration when forming this average, so if one of the values is masked, it is not used in the average. If at least one of the values in the intermediate image bin is not masked, the corresponding output pixel will not be masked. The smoothed image is written to disk with name `outfile`, if specified. If not, no image is written but the image is still accessible via the returned image analysis tool (see below).

This method always returns an image analysis tool which is attached to the smoothed image. This tool should always be captured and closed after any desired manipulations have been done. Closing the tool frees up system resources (eg memory), eg,

```python
smoothedim = ia.boxcar(...)  # do things (or not) with smoothedim ...
# close the returned tool promptly upon finishing with it.
smoothedim.done()
```

**Arguments**
## Inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Allowed Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. Default is none.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask. Default is none.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>axis</td>
<td>Zero-based axis along which to convolve. ia.coordsys().names() gives the order of the axes in the image. Less than 0 means use the spectral axis if there is one, if not an exception is thrown.</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>width</td>
<td>Width of the boxcar in pixels.</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>drop</td>
<td>Drop every nth pixel on output, where n is the width of the boxcar?</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>dmethod</td>
<td>If drop=True, method to use in plane decimation. &quot;copy&quot;: direct copy of every second plane, &quot;m(ean)&quot;; average planes n<em>i through n</em>(i+1) - 1 (inclusive) in the smoothed, non-decimated image to form plane i in the output image.</td>
<td>string</td>
<td>copy</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>stretch</td>
<td>Stretch the mask if necessary and possible? See help par.stretch. Default False</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

## Returns

image
Example

```python
ia.open("mynonsmoothed.im")
# smooth the spectral axis by 3 pixels, say it's axis 2 and only
# write every other pixel
boxcar = ia.boxcar(outfile="myboxcarsmoothed.im", axis=2, drop=True,
width=3, dmethod="c" overwrite=True)
# done with input
ia.done()
# do something with the output image, get statistics say
stats = boxcar.statistics()
# close the result image
boxcar.done()
```
image.brightnessunit.html

**image.brightnessunit - Function**

1.1.1 Get the image brightness unit

**Description**

This function gets the image brightness unit. Both float and complex valued images are supported.

**Arguments**

**Returns**

string

**Example**

```python
"""
# print "\t----\t brightnessunit Ex 1 \t----"
ia.maketestimage()
print ia.brightnessunit()
#Jy/beam
ia.close()
#
"""
```
image.calc.html

**image.calc - Function**

1.1.1 Image calculator

**Description**

This function is used to evaluate a mathematical expression involving CASA images, assigning the result to the current (already existing) image. Both float and complex valued images are supported, although the image which results from the calculation must have the same type of pixel values as the image already attached to the tool. That is, one cannot create a complex valued image using this method if the associated ia tool is currently attached to a float valued image. It complements the imagecalc function which returns a newly constructed on-the-fly image tool. See [note 223](#) which describes the the syntax and functionality in detail.

If the expression, supplied via the `pixels` argument, is not a scalar, the shapes and coordinates of the image and expression must conform.

If the image (that associated with the tool) has a `pixel mask`, then only pixels for which that mask is good will be changed. See the function `maskhandler` for managing image `pixel masks`.

Note that when multiple image are used in the expression, there is no guarantee about which of those images will be used to create the header of the output image. Therefore, one may have to modify the output header as needed if the input headers differ.

See the related functions `set` and `putregion`.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixels</td>
<td>LEL expression</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>verbose</td>
<td>Emit possibly useful messages.</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: True</td>
</tr>
</tbody>
</table>

**Returns**

`bool`
The first example shows that there are 2 \( \text{\textbackslash cf min} \) functions. One with a single argument returning the minimum value of that image. The other with 2 arguments returning an image containing \('aF'\) data clipped at the value of the 2nd argument. The second example sets all good pixels to unity.

This shows a mixed type expression. The real part of the complex image \('aC'\) is used in an expression that otherwise uses Float type.
image.calc.mask.html

**image.calc.mask - Function**

1.1.1 Image mask calculator

**Description**

This method is used to create a new pixel mask via a Boolean LEL expression. This gives you much more scope than the simple set and putregion functions. Both float and complex valued images are supported. See http://casa.nrao.edu/aips2/docs/notes/223/index.shtml which describes the the syntax and functionality of LEL in detail. Also in this document is a description of ways to escape image names that contain certain non-alphanumeric characters so they are compatible with LEL syntax.

If the expression is not a scalar, the shapes and coordinates of the image and expression must conform. If the expression is a scalar then the entire pixel mask will be set to that value.

By default (argument name) the name of a new pixel mask is made up for you. However, if you specify a pixel mask name (use function summary or maskhandler to see the mask names) then it is used. If the pixel mask already exists, it is overwritten.

You can specify whether the new pixel mask should be the default mask or not. By default, it is made the default pixel mask!

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask. Default is none.</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>name</td>
<td>Mask name. Default is auto new name.</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>asdefault</td>
<td>Make specified mask the default mask?</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: true</td>
</tr>
</tbody>
</table>

**Returns**

bool
Example

""
#
print "$t----\t calcmask Ex 1 \t----"
ia.maketestimage('zz', overwrite=true)
subim = ia.subimage() # Make "another" image
ia.calcmask('T')       # Specify 'True' mask as a string
ia.calcmask('zz>0')    # Mask of zz ignored
ia.calcmask('mask(zz) &\& zz>0') # Mask of zz included
ia.calcmask(subim.name(true)!='min('+subim.name(true)+')') # Use tool names
ia.calcmask('zz>min(zz:nomask)') # Mask of zz not used in scalar function
subim.done()
ia.close()
#
""

The first calcmask example is the equivalent of \{\texttt{ia.set(pixelmask=1)}\}. It sets the entire mask to True.

The second example creates a new \texttt{pixelmask} which is True when the pixel values in image \texttt{zz} are greater than 0.

Now for some subtlety. Read carefully! Any LEL expression can be thought of as having a value and a mask. Usually the value is Float and the mask Boolean. In this case, because the expression is Boolean itself, the value is also Boolean. The expression mask would just be the mask of \texttt{zz}. Now what \texttt{ia.calcmask} does is create a mask from the expression value (which is Boolean) and discards the expression mask. Therefore, the resulting mask is independent of any mask that \texttt{zz} might have.

If you wish the mask of the expression be honoured as well, then you can do as in the third example. It says the output \texttt{pixelmask} will be True if the current \texttt{pixelmask} of \texttt{zz} is True and the expression value is True.

The fourth example is like the second, except that we use the pixel values associated with the on-the-fly \{\texttt{subim} Image tool\ disk file. Note one further subtlety here. When the scalar function \texttt{min} evaluates a value from \texttt{subim.name()}, which in this case is just \texttt{zz}, the default mask of \texttt{subim.name()} \{it will\} be used. All the scalar
functions look at the mask. If you didn’t want the mask to be used you can use the special \{\cf :nomask\} syntax shown in the final example.
image.close.html

image.close - Function

1.1.1 Close the image tool

Description

This function closes the image tool. This means that it detaches the tool from its image file (flushing all the changes first). The image tool is “null” after this change (it is not destroyed) and calling any tool function other than open will result in an error.

Arguments

Returns

bool

Example

```python
###
# print "\t----\t close Ex 1 \t----"
ia.maketestimage('myimage',overwrite=true)  # First create an image and attach the image tool
ia.close()                           # The Image tool is detached from the image using the close tool.
print "!!!EXPECT ERROR HERE!!!"
ia.summary()                        # The image is not open, so attempting to display summary information fails.
ia.open('myimage')                  # The image tool is reattached to the image using the open tool.
ia.summary()                        # No error - the summary information is now displayed correctly.
ia.close()                           # The Image tool is detached from the image again, using the close tool.
#
###
```
image.continuumsub.html

image.continuumsub - Function

1.1.1 Image plane continuum subtraction

Description

This function packages the relevant image tool functionality for simple specification and application of image plane continuum subtraction. All that is required of the input image is that it have a non-degenerate spectral axis. The user specifies region, the region of the input image over which continuum subtraction is desired (otherwise the whole image will be treated); channels, the subset of channels on the spectral axis to use in the continuum estimation, specified as a vector; fitorder, the polynomial order to use in the estimation. Optionally, output line and continuum images may be written by specifying outline and outcont, respectively. If outline is not specified, a virtual image tool is all that is produced. If outcont is not specified, the output continuum image will be written in 'continuumsub.im'. Note that the pol parameter is no longer supported; one should use the region parameter if polarization selection is desired, in conformance with other ia tool methods.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outline</td>
<td>Output line image filename. Default is unset.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>outcont</td>
<td>Output continuum image filename</td>
<td>string</td>
<td>continuumsub.im</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
<td></td>
<td></td>
</tr>
<tr>
<td>channels</td>
<td>Channels to use for continuum estimation. Default is all.</td>
<td>intArray</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: intArray</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pol</td>
<td>THIS PARAMETER IS NO LONGER SUPPORTED. USE THE region PARAMETER TO CHOOSE WHICH POLARIZATIONS YOU WOULD LIKE TO PROCESS</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fitorder</td>
<td>Polynomial order for continuum estimation</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>overwrite</td>
<td>Auto-overwrite output files if they exist?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returns</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

Fit a second order polynomial (fitorder=2) to channels 3–8 and 54–60 (python’s range function includes the lower limit and excludes the upper limit) to an RA x Dec x Frequency x Stokes cube.

ia.open("myimage.im")
# select stokes plane 1 on which to perform the fit, as well
# as a box of pixels with blc=25,25 and trc=75,75 in the direction
# plane, and channels 0 to 100. This will be the portion of the cube
# from which the fit is subtracted
reg = rb.box(blc=[25, 25, 0, 1], trc=[75, 75, 100, 1])
csub = ia.continuumsub(region=reg, channels=range(3,9)+range(54,61), fitorder=2)

# do stuff with original image (ia) and csub image tools as necessary and
# finally close them
ia.done()
csub.done()
**image.convertflux - Function**

1.1.1 Convert peak intensity to/from flux density for a 2D Gaussian.

**Description**

This function interconverts between peak intensity and flux density for a Gaussian component. The image must hold a restoring beam.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Flux density to convert. Must be specified.</td>
<td>any</td>
<td>0Jy/beam</td>
</tr>
<tr>
<td>major</td>
<td>Major axis of component. Must be specified.</td>
<td>any</td>
<td>1arcsec</td>
</tr>
<tr>
<td>minor</td>
<td>Minor axis of component. Must be specified.</td>
<td>any</td>
<td>1arcsec</td>
</tr>
<tr>
<td>type</td>
<td>Type of component. String from Gaussian, Disk.</td>
<td>string</td>
<td>Gaussian</td>
</tr>
<tr>
<td>topeak</td>
<td>Convert to peak or integral flux density</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>channel</td>
<td>Channel to use if and only if image has per plane beams.</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>polarization</td>
<td>Zero-based polarization number to use for beam if and only if image has per plane beams.</td>
<td>int</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Returns**

record
Example

```
# print "\t----\t convertflux Ex 1 \t----"
ia.maketestimage('in.im', overwrite=true);
p1 = qa.quantity('1mJy/beam')
i1 = ia.convertflux(p1, major='30arcsec', minor='10arcsec', topeak=F);
p2 = ia.convertflux(i1, major='30arcsec', minor='10arcsec', topeak=T)
print 'peak, integral, peak = ', p1, i1, p2
#peak, integral, peak =  {'value': 1.0, 'unit': 'mJy/beam'}
#                   {'value': 0.00016396129551656742, 'unit': 'Jy'}
#                   {'value': 0.0010000000000000002, 'unit': 'Jy/beam'}

ia.close()
#
```
image.convolve2d.html

**image.convolve2d - Function**

Convolve image by a 2D kernel

### Description

This function performs Fourier-based convolution of an image file using the provided 2D kernel. If `outfile` is left unset, the image is written to the specified disk file. If `outfile` is not given, the newly constructed on-the-fly Image tool is associated with a temporary image. This temporary image may be stored in memory or on disk, depending on its size. When the user destroys the on-the-fly Image tool (with the done function) this temporary image is deleted.

The user specifies which 2 pixel axes of the image are to be convolved via the `axes` argument. The pixels must be square or an error will result.

The user specifies the type of convolution kernel with `type` (minimum match is supported); currently only `'gaussian'` is available.

The user specifies the parameters of the convolution kernel via the arguments `major`, `minor`, and `pa`. These arguments can be specified in one of three ways:

- **Quantity** - for example `major=qa.quantity(1, 'arcsec')` Note that you pixel units can be used, viz. `major=qa.quantity(1, 'pix')`, see below.

- **String** - for example `minor='1km'` (i.e. one that the Quanta quantity function accepts).

- **Numeric** - for example `major=10`. In this case, the units of `major` and `minor` are assumed to be in pixels. Using pixel units allows the user to convolve unlike axes (see one of the provided example for this use case).

  For the position angle, units of degrees are assumed.

The interpretation of `major` and `minor` depends upon the kernel type.

- **Gaussian** - `major` and `minor` are the Full Width at Half Maximum (FWHM) of the major and minor axes of the Gaussian.

The position angle is measured North through East when a plane holding a celestial coordinate (the usual astronomical convention) is convolved. For other axis/coordinate combinations, a positive position angle is measured from +x to +y in the absolute pixel coordinate frame (x is the first axis that is specified, with argument `axes`).
In the case of a Gaussian, the beam parameter offers an alternate way of describing the convolving Gaussian. If used, neither major, stf, nor minor, nor pa can be specified. The beam parameter must have exactly three fields: "major", "minor", and "pa" (or "positionangle"). This is, not coincidentally, the record format for the output of ia.restoringbeam(). The scaling of the output image is determined by the argument scale. If this is left unset then autoscaling will be invoked.

If the user is not convolving the sky, then autoscaling means that the convolution kernel will be normalized to have unit volume so as to conserve flux.

If the user is convolving the sky, then there are two cases for which autoscaling is useful:

Firstly, if the input image units are Jy/pixel, then the output image will have units of Jy/beam and be appropriately scaled. In addition, the restoring beam of the output image will be the same as the convolution kernel.

Secondly, if the input image units are Jy/beam, then the output image will also have units of Jy/beam and be appropriately scaled. In addition, the restoring beam of the output image will be the convolution of the input image restoring beam and the convolution kernel. In the case of an image with per-plane beams, for each plane, the kernel is convolved with the appropriate beam and the result is associated with that plane in the output image.

If the user sets a value for scale, then the convolution kernel will be scaled by this value. Note that it has peak of unity before the application of this scale factor.

If the units on the original image include Jy/beam, the units on the output image will be rescaled by the ratio of the input and output beams as well as rescaling by the area of convolution kernel.

If the units on the original image include K, then only the image convolution kernel rescaling is done.

If targetres=True and type="gaussian" and the input image has a restoring beam, this method will interpret the values of major, minor, and pa as the resolution of the final image and will calculate the parameters of the Gaussian to use in the convolution so that this target resolution is achieved.

Masked pixels will be assigned the value 0.0 before convolution. The output mask is the combination (logical OR) of the default input pixel mask (if any) and the OTF mask. Any other input pixel masks will not be copied. The function maskhandler can be used if there is a need to copy other masks too.

See also the other convolution functions: convolve, hanning, and sepconvolve.

Arguments
<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. The default value is unset.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>axes</td>
<td>Axes to convolve. The default setting is [0,1].</td>
<td>intArray</td>
<td>01</td>
</tr>
<tr>
<td>type</td>
<td>Type of convolution kernel to be used.</td>
<td>string</td>
<td>gaussian</td>
</tr>
<tr>
<td>major</td>
<td>Major axis, Quantity, string, numeric (e.g. 10arcsec, 20pix, 3km, etc.). This must be specified by the user.</td>
<td>any</td>
<td>variant 0deg</td>
</tr>
<tr>
<td>minor</td>
<td>Minor axis, Quantity, string, numeric (e.g. 10arcsec, 20pix, 3km, etc.). This must be specified by the user.</td>
<td>any</td>
<td>variant 0deg</td>
</tr>
<tr>
<td>pa</td>
<td>Position Angle, Quantity, string, numeric (e.g. 10deg). The default value is 0deg.</td>
<td>any</td>
<td>variant 0deg</td>
</tr>
<tr>
<td>scale</td>
<td>Scale factor. The default setting (-1) is to autoscale.</td>
<td>double</td>
<td>-1</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask for more details. The default option is none.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) the pre-existing output file?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>stretch</td>
<td>Stretch the mask if necessary and possible? See help par.stretch.</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>targetres</td>
<td>If True and type=&quot;gaussian&quot;, major, minor, and pa are interpreted as the image resolution that the user wants to achieve.</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>beam</td>
<td>Alternate way of describing a Gaussian. Must have fields &quot;major&quot;, &quot;minor&quot;, &quot;pa&quot; (or &quot;positionangle&quot;)</td>
<td>record</td>
<td></td>
</tr>
</tbody>
</table>
Returns
image

Example

""
#
print "\t----\t convolve2d Ex 1 \t----"
ia.maketestimage('xy', overwrite=true)  # Create a simple RA/DEC test image
# Convolve axes 0 and 1 of the test image with a 20x10-arcsec, 45-degree Gaussian:
im2 = ia.convolve2d(outfile='xy.con', axes=[0, 1], type='gauss',
                   major='20arcsec', minor='10arcsec', pa='45deg',
                   overwrite=true);
# Clean up, by destroying the im2 tool and close the image tool:
im2.done()
ia.close()
#

ia.fromarray(outfile='xypf', pixels=ia.makearray(0, [64, 64, 4, 64]),
              overwrite=true)  # Create a simple RA/DEC/Pol/Freq test dataset
print "!!!EXPECT WARNING REGARDING INVALID SPATIAL RESTORING BEAM!!!"
# Convolve axes 0 and 3 of the test dataset with a 20x10-pixel, 45-degree Gaussian:
im2 = ia.convolve2d(outfile='xypf.con', axes=[0, 3], type='gauss',
                    major='20pix', minor='10pix', pa='45deg',
                    overwrite=true);
# Note that pixel units must be used in the above because axes 0 and 3 are unlike.
# Clean up, by destroying the im2 tool and close the image tool:
im2.done()
ia.close()
#
"""
image.coordsys.html

image.coordsys - Function

1.1.1 Get the Coordinate System of the image

Description

This function returns the Coordinate System of an image in a Coordsys tool. Both float and complex valued images are supported. By default, the Coordinate System describes all of the axes in the image. If you desire, you can select a subset of the axes, thus reducing the dimensionality of the Coordinate System. This may be useful if you are supplying a Coordinate System to the functions fromarray or fromshape.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Axes to which the Coordinate System pertains. Default is all axes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>axes</td>
<td>allowed: intArray</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
</tbody>
</table>

Returns

coordsys

Example

```python
""
# print "\t\t\t coordsys Ex 1 \t---" 
ia.maketestimage('hcn',overwrite=true) 
ia.summary()
mycs = ia.coordsys([0,1])
imshape = ia.shape()
ia.fromshape(outfile='test', shape=imshape, csys=mycs.torecord(), overwrite=true) 
ia.summary()
```
mycs.done()
ia.close()
#
"

In this example, we create a Coordinate System pertaining to the first two axes of the image and then we create a new (empty) 2D image with this Coordinate System using the \texttt{\cf fromshape} function.
image.coordmeasures.html

**image.coordmeasures - Function**

Convert from pixel to world coordinate wrapped as Measures

**Description**

You can use this function to get the world coordinates for a specified absolute pixel coordinate in the image. You specify a pixel coordinate (0-rel) for each axis in the image.

If you supply fewer pixel values then there are axes in the image, your value will be padded out with the reference pixel for the missing axes. Excess values will be ignored.

The parameters dframe and sframe allow one to specify to which reference frame the direction and spectral measures, respectively, should be converted. These values are case-insensitive. "native" means use the native reference frame of the coordinate in question. "cl" means use the conversion layer frame if one exists (if not, the native frame will be used).

The world coordinate is returned as a record of measures. This function is just a wrapper for the Coordsys tool toworld function (invoked with argument `format='m'`). Please see its documentation for discussion about the formatting and meaning of the measures.

This Image tool function adds two additional fields to the return record.

The `mask` field contains the value of the image pixel mask at the specified position. It is either T (pixel is good) or F (pixel is masked as bad or the specified position was off the image).

The `intensity` field contains the value of the image (at the nearest pixel to that given) and its units. This is actually stored as a Quantity. This field does not exist if the specified pixel coordinate is off the image.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pixel</td>
<td>Absolute pixel coordinate. Default is reference pixel. allowed: doubleArray Default: -1</td>
<td></td>
</tr>
<tr>
<td>dframe</td>
<td>Direction reference frame to which to convert the direction data. Case insensitive. &quot;cl&quot; means use the conversion layer, if present, of the image direction coordinate. &quot;native&quot; means use the native native direction frame of the image. Other examples are &quot;J2000&quot;, &quot;B1950&quot;, &quot;GALACTIC&quot;, etc. allowed: string Default: cl</td>
<td></td>
</tr>
<tr>
<td>sframe</td>
<td>Spectral reference frame to which to convert the spectral data. Case insensitive. &quot;cl&quot; means use the conversion layer, if present, of the image spectral coordinate. &quot;native&quot; means use the native spectral reference frame of the image. Other examples are &quot;LSRK&quot;, &quot;CMB&quot;, &quot;LGROUP&quot;, etc. allowed: string Default: cl</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
""
#
print "\t----\t coordmeasures Ex 1 \t----"
ia.maketestimage('myimage', overwrite=true)
s = ia.shape()
for i in range(len(s)):
    s[i] = 0.5*s[i]
meas = ia.coordmeasures(s)
print meas.keys()  # Get names of fields in record
#['intensity', 'mask', 'measure']
print meas['intensity']
#{'value': 1.3992415665802, 'unit': 'Jy/beam'}
print meas['measure']['direction']
#{'type': 'direction',
In this example we first find the world coordinates of the centre of the image. Then we use the Measures tool \texttt{me} to convert the \texttt{direction coordinate} field from J2000 to an azimuth and elevation at a particular location at a particular time.
image.decompose.html

**image.decompose - Function**

1.1.1 Separate a complex image into individual components

**Description**

This function is an image decomposition tool that performs several tasks, with the end result being that a strongly blended image is separated into components - both in the sense that it determines the parameters for each component (assuming a Gaussian model) and that it physically assigns each pixel in the image to an individual object. The products of these two operations are called the component list and the component map, respectively. The fitting process (which determines the component list) and the pixel-decomposition process (which determines the component map) are designed to work cooperatively to increase the efficiency and accuracy of both. The algorithm behind the decomposition is based on the function clfind, described in Williams et al 1994, which uses a contouring procedure whereby a closed contour designates a separate component. The program first separates the image into clearly distinct 'regions' of blended emission, then contours each region to determine the areas constituting each component and passes this information on to the fitter, which determines the component list. The contour deblending can optionally be replaced with a simpler local maximum scan, and the fitting can be replaced with a moment-based estimation method to speed up calculations on very large images or if either primary method causes trouble, but in general this will impede the accuracy of the fit.

The function works with both two and three dimensional images. The return value is a record (or dictionary) that has 3 keys: 'components', 'blc', 'trc'. The 'components' element is a matrix each row of which contains the gaussian parameters of the component fitted. The 'blc' element is a matrix of the bottom left corners (blc) of the regions found. Each row correspond to a region blc. The 'trc' element is a matrix of the top right corners (trc) of the regions found. Each row correspond to a region trc. **Please Note** that the returned blc's and trc's are relative to region defined by the user. A blc of [0,0] implies the bottom left of the region selected and not the bottom left of the image. Obviously if no region is defined then it is the bottom left of the image.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>allowed: variant</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
<tr>
<td></td>
<td><strong>mask</strong></td>
</tr>
<tr>
<td></td>
<td>Mask to use. See help par.mask. Default is none.</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
<tr>
<td></td>
<td><strong>simple</strong></td>
</tr>
<tr>
<td></td>
<td>Skip contour deblending and scan for local maxima</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: false</td>
</tr>
<tr>
<td></td>
<td><strong>threshold</strong></td>
</tr>
<tr>
<td></td>
<td>Value of minimum positive contour. Must be set and nonnegative.</td>
</tr>
<tr>
<td></td>
<td>allowed: double</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td></td>
<td><strong>ncontour</strong></td>
</tr>
<tr>
<td></td>
<td>Number of contours to use in deblending (&gt;= 2)</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 11</td>
</tr>
<tr>
<td></td>
<td><strong>minrange</strong></td>
</tr>
<tr>
<td></td>
<td>Minimum number of closed contours in a component (&gt;= 0)</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 1</td>
</tr>
<tr>
<td></td>
<td><strong>naxis</strong></td>
</tr>
<tr>
<td></td>
<td>Max number of perpendicular steps between contiguous pixels. Values of 1, 2 or 3 are allowed.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 2</td>
</tr>
<tr>
<td></td>
<td><strong>fit</strong></td>
</tr>
<tr>
<td></td>
<td>Fit to the components after deblending?</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: true</td>
</tr>
<tr>
<td></td>
<td><strong>maxrms</strong></td>
</tr>
<tr>
<td></td>
<td>Maximum RMS of fit residuals to not retry fit (&gt;= 0). Default is unset.</td>
</tr>
<tr>
<td></td>
<td>allowed: double</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td></td>
<td><strong>maxretry</strong></td>
</tr>
<tr>
<td></td>
<td>Maximum number of times to retry the fit (&gt;= 0). Default is unset.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td></td>
<td><strong>maxiter</strong></td>
</tr>
<tr>
<td></td>
<td>Maximum number of iterations allowed in a single fit (&gt;= 0)</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 256</td>
</tr>
<tr>
<td></td>
<td><strong>convcriteria</strong></td>
</tr>
<tr>
<td></td>
<td>Criterion to establish convergence (&gt;=0)</td>
</tr>
<tr>
<td></td>
<td>allowed: double</td>
</tr>
<tr>
<td></td>
<td>Default: 0.0001</td>
</tr>
<tr>
<td></td>
<td><strong>stretch</strong></td>
</tr>
<tr>
<td></td>
<td>Stretch the mask if necessary and possible? See help par.stretch.</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: false</td>
</tr>
</tbody>
</table>
Returns record

Example

""
#
print "\t----\t decompose Ex 1 \t----"
ia.maketestimage()
out=ia.decompose(threshold=2.5, maxrms=1.0)
#Attempt 1: Converged after 21 iterations
#Attempt 1: Converged after 15 iterations
#1: Peak: 17.955  Mu: [0.000327928, 8.62573e-05]
#   Axes: [0.00175981, 0.00142841] Rotation: 1.29539
#2: Peak: 19.8093  Mu: [1.67927e-06, -0.00374393]
#   Axes: [0.00179054, 0.00132541] Rotation: 1.78404
#3: Peak: 10.1155  Mu: [6.28252, -7.09688e-05]
#   Axes: [0.00180877, 0.00104523] Rotation: 1.78847
print out['components']
## [ 1.79549522e+01 3.27928370e-04 8.62573434e-05 1.75980886e+00
#  8.11686337e-01 1.29538655e+00]
# [ 1.98093319e+01 1.67927124e-06 -3.74393392e-04 1.79054437e+00]
# [ 7.40229547e-01 1.78403902e+00]
# [ 1.01155214e+01 6.28252172e+00 -7.09688029e-05 1.80877140e+00
#  5.77867746e-01 1.78847444e+00]
print out['blc']
## [37 31]
# [47 25]
# [67 33]]
print out['trc']
## [[54 47]
# [66 38]
# [78 40]]
ia.close()
#
"""
Deconvolve a componentlist from the restoring beam

Description

This method deconvolves (a record representation of) a Componentlist tool from the restoring beam, returning (a record representation of) a new Componentlist tool. If there is no restoring beam, a fail is generated. Currently, only deconvolution of Gaussian components is supported. For images with per-plane beam, the user must choose which beam is used for the deconvolution by setting channel and/or polarization. Only a single beam is used to deconvolve all components. See also functions setrestoringbeam and restoringbeam.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Componentlist to deconvolve</th>
</tr>
</thead>
<tbody>
<tr>
<td>complist</td>
<td>record</td>
</tr>
<tr>
<td>channel</td>
<td>Zero-based channel number to use for beam for per plane images. Not used if the image has a single beam.</td>
</tr>
<tr>
<td></td>
<td>int</td>
</tr>
<tr>
<td>polarization</td>
<td>Zero-based polarization number to use for beam for per plane images. Not used if the image has a single beam.</td>
</tr>
<tr>
<td></td>
<td>int</td>
</tr>
</tbody>
</table>

Returns

record

Example

125
print "\t----\t deconvolvecomponentlist Ex 1 \t----"
ia.maketestimage()
r = ia.fitcomponents()
c1 = r['results']
cl1 = r['results']
cl2 = ia.deconvolvecomponentlist(cl1) # of componentlists
print c11, cl2
cl.fromrecord(cl2) # set componentlist tool with record
ia.close()
cl.close()
#
image.deconvolvefrombeam - Function

Helper function to deconvolve the given source Gaussian from a beam Gaussian to return a model Gaussian

Description

This is a helper function. It is to provide a way to deconvolve gaussians from other gaussians if that is what is needed for example removing a beam Gaussian from a Gaussian source. To run this function the tool need not be attached to an image.

The return value is a record that contains the fit param and the return value is a boolean which is set to true if fit model is a point source

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th>Three quantities that define the source majoraxis, minoraxis and Position angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td>variant</td>
</tr>
<tr>
<td>beam</td>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td>variant</td>
</tr>
</tbody>
</table>

Returns

record

Example

```
#`
```

127
print "\t----\t deconvolvefrombeam Ex 1 \t----"

ia.maketestimage()
recout=ia.deconvolvefrombeam(source=['5arcmin', '3arcmin', '20.0deg'], beam=['50arcsec', '30arcsec', '15deg'])

ia.close()

print 'Is pointsource ', recout['return']
print 'major=',recout['fit']['major']
print 'minor=',recout['fit']['minor']
print 'pa=',recout['fit']['pa']

"""
image.beamforconvolvedsize.html

**image.beamforconvolvedsize - Function**

1.1.1 Determine the size of the beam necessary to convolve with the given source to reach the given convolved (source+beam) size

**Description**

Determine the size of the beam necessary to convolve with the given source to reach the given convolved (source+beam) size. Because the problem is completely specified by the input parameters, no image needs to be attached to the associated tool; e.g., `ia.open()` need not be called prior to calling this method.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>Three quantities that define the deconvolved source major axis, minor axis and position angle</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
<tr>
<td>convolved</td>
<td>Three quantities that define the convolved source (source+beam) major axis, minor axis and position angle. Do not specify if beam is specified.</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
# get the beam necessary to convolve the specified source with to achieve the target convolved source size
beam = ia.beamforconvolvedsize(source=["1arcsec", "1arcsec", "0deg"], convolved=\n"3arcsec", "2arcsec", "45deg")
```
image.commonbeam.html

**image.commonbeam - Function**

Determine a beam to which all beams in an image can be convolved.

**Description**

Determine a beam to which all beams in an image can be convolved. If the image does not have a beam, an exception will be thrown. If the image has a single beam, that beam will be returned. If the image has multiple beams, this will be the beam with the largest area in the image beam set if all the other beams can be convolved to that beam. If not, this is guaranteed to be the minimum area beam to which all beams in the set can be convolved if all but one of the beams in the set can be convolved to the beam in the set with the largest area. Otherwise, the returned beam may or may not be the smallest possible beam to which all the beams in the set can be convolved.

**Arguments**

**Returns**

record

**Example**

```python
ia.open("mymultibeamimage.im")
cb = ia.commonbeam()
# convolve all the planes in the image with that beam
ia.convolve2d(outfile="myconvolvedimage.im", major=cb["major", minor=cb["minor"], pa=cb[...
```
image.remove.html

**image.remove - Function**

Delete the image file associated with this image tool

**Description**

This function first closes the image tool which detaches it from its underlying image file. It then deletes that image file. If done=True, the image tool is still viable, and can be used with function open to open a new image file. Otherwise the image tool is destroyed. If verbose=True, the logger will receive a progress report.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>done</td>
<td>Destroy this tool after deletion</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>verbose</td>
<td>Send a progress report to the logger.</td>
<td>bool</td>
<td>true</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```plaintext
    #
    print "\t----\t remove Ex 1 \t----"
    ia.maketestimage('myimage',overwrite=true)
    ia.close()
    ia.maketestimage('myotherimage',overwrite=true)
    ia.close()
    ia.open('myimage') # Attach to 'myimage'
```

132
ia.remove(F)  # Close imagetool and delete 'myimage'
ia.open('myotherimage')  # Open new imagefile 'myotherimage'
ia.remove()
print "!!!EXPECT THE FOLLOWING TO GENERATE AN ERROR MESSAGE!!!"
ia.open('myimage')  # 'myimage' was deleted above
ia.close()
#
""
**image.removefile** - Function

Delete an unattached image file from disk. Note: use remove() if the image file is attached to the image tool.

### Description

This function deletes the specified image file.

### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Name of image file/directory to be removed. Must be specified.</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
</tbody>
</table>

**Returns**

bool

### Example

```
#  print "\t----\t removefile Ex 1 \t----"
ia.maketestimage('myimage',overwrite=true)
ia.close()
ia.removefile('myimage') # remove image 'myimage'
ia.maketestimage('myimage',overwrite=false) # error here if 'myimage' exists
ia.close()
ia.removefile('myimage')
#
```
image.done.html

**image.done - Function**

1.1.1 Destroy this image tool

**Description**

When the user no longer needs to use an image tool, calling this function will free up its resources. That is, it destroys the tool. This means that the user can no longer call any functions on the tool after it has been done.

If the Image tool is associated with a disk file, then (unlike the close function, the user can also choose to delete that by setting remove=true. By default, any associated disk file is not deleted.

Note that this function is different from the close function because the latter does not destroy the image tool. For example, the user can use the open function straight after the close function on the same tool.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remove</td>
<td>Delete the associated disk file as well?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>verbose</td>
<td>Send a progress report to the logger?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>true</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```bash

# Make a test image and create tool subim:
```

135
ia.maketestimage('myfile', overwrite=true)
subim = ia.subimage('myfile2', overwrite=true)

# Check that subim exists as intended by attempting to display its summary:
subim.summary()  # This displays a summary of the dataset.

# Use done to destroy the subim tool:
subim.done()

# Check that the subim tool has been detached as intended, by attempting to display its summary:
subim.summary()  # This should now throw an error.

ia.summary()  # This still works, though, as the ia tool is still open, and the dataset is still available.

ia.close()

#
image.fft.html

image.fft - Function

1.1.1 FFT the image

Description

This function fast Fourier Transforms the supplied image to the Fourier plane. Both float valued and complex valued images are supported. If the axes parameter is left unset, then the sky plane of the image (if there is one) is transformed. Otherwise, the user can specify which axes are to be transformed. Note that if a sky axis is to be transformed, both of them must be specified. The user specifies which form is desired in the result by specifying the desired output image file name(s). Before the FFT is performed, any masked pixels are set to values of zero. The output mask is the combination (logical OR) of the default input pixel mask (if any) and the OTF mask. Any other input pixel masks will not be copied. The function maskhandler can be used if there is a need to copy other masks too.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>real</td>
<td>Output real image file name. allowed: string Default:</td>
</tr>
<tr>
<td>imag</td>
<td>Output imaginary image file name. allowed: string Default:</td>
</tr>
<tr>
<td>amp</td>
<td>Output amplitude image file name. allowed: string Default:</td>
</tr>
<tr>
<td>phase</td>
<td>Output phase image file name. allowed: string Default:</td>
</tr>
<tr>
<td>axes</td>
<td>Specify the pixel axes that are to undergo the FFT. The default option (-1) is to transform the sky plane(s). allowed: intArray Default: -1</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image. allowed: any Default: variant</td>
</tr>
<tr>
<td>mask</td>
<td>The mask to be used. See &quot;help par.mask&quot; for more details. The default option is none. allowed: any Default: variant</td>
</tr>
<tr>
<td>stretch</td>
<td>Stretch the mask if it is necessary and possible. See &quot;help par.stretch&quot; for more details. allowed: bool Default: false</td>
</tr>
<tr>
<td>complex</td>
<td>Output complex valued image file name. allowed: string Default:</td>
</tr>
</tbody>
</table>

**Returns**
bool

**Example**
print "\t----\t fft Ex 1 \t----"
# Create a test image:
ia.maketestimage('gc.small', overwrite=true)
# Perform an FFT on the sky plane of the test image,
# writing out just the resulting real and amplitude images:
ia.fft(real='r.im', amp='a.im')
# Close the image tool when done:
ia.close()
# Lastly, clean up the example output files:
ia.removefile('r.im')
ia.removefile('a.im')
#
"

Example

"

print "\t----\t fft Ex 2 \t----"
# Create a zero-filled 3D test dataset and add noise to it:
ia.fromshape('gc.small', [64,64,128], overwrite=true)
ia.addnoise(type='normal', pars=[0.5, 1], zero=false)
# The following transforms only the third axis of the image.
# writing out only the amplitude and phase images.
ia.fft(amp='amp.im', phase='p.im', axes=[2])
# Close the image tool when done:
ia.close()
# Lastly, clean up the example output files:
ia.removefile('amp.im')
ia.removefile('p.im')
#
"
image.findsources.html

**image.findsources - Function**

1.1.1 Find point sources in the sky

**Description**

This function finds strong point sources in the image. The sources are returned in a record that can be used by a Componentlist tool. An efficient method is used to locate sources under the assumption that they are point-like and not too close to the noise. Only sources with a peak greater than the cutoff fraction of the strongest source will be found. Only positive sources will be found, unless the negfind=T whereupon positive and negative sources will be found.

After the list of point sources has been made, you may choose to make a Gaussian fit for each one (point=F) so that shape information can be recovered as well. You can specify the half-width of the fitting grid with argument width which defaults to 5 (fitting grid would then be [11,11] pixels). If you set width=0, this is a signal that you would still like Gaussian components returned, but a default width should be used for the Gaussian shapes. The default is such that the component is circular with a FWHM of width pixels. Thus, if point=T, the components in the returned Componentlist are Point components. If point=F then Gaussian components are returned.

The region-of-interest must be 2-dimensional and it must hold a region of the sky. Any degenerate trailing dimensions in the region are discarded.

See also the function fitcomponents (for which findsources can provide an initial estimate).

**Arguments**
Inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmax</td>
<td>Maximum number of sources to find, &gt; 0</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 20</td>
</tr>
<tr>
<td>cutoff</td>
<td>Fractional cutoff level</td>
</tr>
<tr>
<td></td>
<td>allowed: double</td>
</tr>
<tr>
<td></td>
<td>Default: 0.1</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask. Default is none.</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
<tr>
<td>point</td>
<td>Find only point sources?</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: true</td>
</tr>
<tr>
<td>width</td>
<td>Half-width of fit grid when point=F</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 5</td>
</tr>
<tr>
<td>negfind</td>
<td>Find negative sources as well as positive?</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: false</td>
</tr>
</tbody>
</table>

Returns

record

Example

```python
""
# print "\t\t\t\tfindsources Ex 1 \t\t\t"
ia.maketestimage()
clrec = ia.findsources(nmax=5, cutoff=0.5)
print clrec
# """
All sources stronger than 0.5 of the strongest will be found. We use the Componentlist GUI to look at the strongest component.
image.fitprofile.html

**image.fitprofile - Function**

1.1.1 Fit gaussians and/or polynomials to a 1-dimensional profile.

**Description**

This application simultaneously fits any number of gaussian singlets, any number of lorentzian singlets, and any number of gaussian multiplets, and/or a polynomial to one dimensional profiles using the non-linear, least squares Levenberg-Marquardt algorithm. A description of the fitting algorithm may be found in AIPS++ Note 224 (http://www.astron.nl/casacore/trunk/casacore/doc/notes/224.html) and in Numerical Recipes by W.H. Press et al., Cambridge University Press. A gaussian/lorentzian singlet is a gaussian/lorentzian whose parameters (amplitude, center position, and width) are all independent from any other feature that may be simultaneously fit. A gaussian multiplet is a set of two or more gaussian lines in which at least one (and possibly two or three) parameter of each line is dependent on the parameter of another, single (reference) profile in the multiplet. For example, one can specify a doublet in which the amplitude of the first line is 0.6 times the amplitude of the zeroth line and/or the center of the first line is 20 pixels from the center of the zeroth line, and/or the fwhm of the first line is identical (in pixels) to that of the zeroth line. There is no limit to the number of components one can specify in a multiplet (except of course that the number of parameters to be fit should be significantly less than the number of data points), but there can be only a single reference profile in a multiplet to which to tie constraints of parameters of the other profiles in the set.

Additionally, a power logarithmic polynomial (plp) or a logarithmic transformed polynomial (ltp) can be fit. In this case, each of these functions cannot be fit simultaneously with any other supported function. These functions are most often used for fitting the spectral index and higher order terms of a spectrum. A power logarithmic polynomial has the form

\[ y = c0*x/div**(c1 + c2*ln(x/div) + c3*ln(x/div)**2 + ... + cn*ln(x/div)**(n - 1)) \]

and a logarithmic transformed polynomial is simply the result of this equation after taking the natural log of both sides so that it has the form

\[ \ln(y) = c0 + c1*ln(x/div) + c2*ln(x/div)**2 + ... + cn*ln(x/div)**n \]

The coefficients of the two forms correspond with each other except that c0 in the second equation is equal to \( \ln(c0) \) of the first. In the case of fitting a spectral index, the spectral index, traditionally represented as alpha, is equal to c1.
In both cases, div is a numerical value used to scale abscissa values so they are closer to unity when they are sent to the fitter. This generally improves the probability that the fit will converge. This parameter may be specified via the div parameter. A value of 0 (the default) indicates that the application should determine a reasonable value for div, which is determined via
\[
div = 10^{*\text{int}(\log10(\sqrt{\text{min}(x)\cdot\text{max}(x)})))
\]
where min(x) and max(x) are the minimum and maximum abscissa values, respectively.

So, for example, if S(nu) is proportional to \(\text{nu}^{\alpha}\) and you expect \(\alpha\) to be near -0.8 and the value of \(S(\text{nu})\) is 1.5 at 1e9 Hz and your image(s) have spectral units of Hz, you would specify \(\text{spxest}=[1.5, -0.8]\) and \(\text{div}=1\text{e9}\) when fitting a plp function, or \(\text{spxest}=[0.405, -0.8]\) and \(\text{div}=1\text{e9}\) if fitting an ltp function.

More details of fitting all of these functions are described in following sections.

A CAUTIONARY NOTE Note that the likelihood of getting a reliable solution increases with the number of good data points as well as the goodness of the initial estimate. It is possible that the first solution found might not be the best one, and so, if a solution is found, it is recommended that the fit be repeated using the solution of the previous fit as the initial estimate for the new fit. This process should be repeated until the solutions from one fit to the next differ only insignificantly. The convergent solution is very likely the best solution.

AXIS The axis parameter indicates on which axis profiles should be fit; a value of 0 indicates the spectral axis should be used, or if one does not exist, that the zeroth axis should be used.

MINIMUM NUMBER OF PIXELS The minpts parameter indicates the minimum number of unmasked pixels that must be present in order for a fit to be attempted. When multifit=T, positions with too few good points will be masked in any output images.

ONE FIT OF REGION AVERAGE OR PIXEL BY PIXEL FIT The multifit parameter indicates if profiles should be fit at each pixel in the selected region (true), or if the profiles in that region should be averaged and the fit done to that average profile (false).

POLYNOMIAL FITTING The order of the polynomial to fit is specified only via the poly parameter. If poly=0, no polynomial will be fit. No initial estimates of coefficients can be specified; these are determined automatically.

GAUSSIAN SINGLET FITTING In the absence of an estimates file and no estimates being specified by the p*est parameters, and gmncomps=0 or is empty, the ngauss parameter indicates the maximum number of gaussian singlets that should be fit. The initial estimates of the parameters for these gaussians will be attempted automatically in this case. If it deems appropriate, the fitter will fit fewer than this number. In the case where an estimates file is supplied, ngauss is ignored (see below). ngauss is also ignored if the p*est parameters are specified or if gmncomps is not an empty array or, if an integer, is greater than zero. If estimates is not specified or the p*est parameters are not specified and ngauss=0, gmncomps is empty or 0, and
polyβ0, an error will occur as this indicates there is nothing to fit.

One can specify initial estimates of gaussian singlet parameters via an estimates file or the pampest, pcenterest, pfwhmest, and optionally, the pfix parameters. The latter is the recommended way to specify these estimates as support for estimates files may be deprecated in the future. No matter which option is used, an amplitude initial estimate must always be nonzero. A negative fwhm estimate will be silently changed to positive.

SPECIFYING INITIAL ESTIMATES FOR GAUSSIAN AND LORENTZIAN SINGLETS (RECOMMENDED METHOD) One may specify initial estimates via the pampest, pcenterest, and pfwhmest parameters. In the case of a single gaussian or lorentzian singlet, these parameters can be numbers. pampest must be specified in image brightness units, pcenterest must be given in the number of pixels from the zeroth pixel, and pfwhmest must be given in pixels. Optionally pfix can be specified and in the case of a single gaussian or lorentzian singlet can be a string. In it is coded which parameters should be held constant during the fix. Any combination of "p" (amplitude), "c" (center), or "f" (fwhm) is allowed; eg pfix=\"pc\" means fix both the amplitude and center during the fit. In the case of more than one gaussian and/or lorentzian singlets, these parameters must be specified as arrays of numbers. The length of the arrays indicates the number of singlets to fit and must be the same for all the p*est parameters. If no parameters are to be fixed for any of the singlets, pfix can be set to the empty string. However, if at least one parameter of one singlet is to be fixed, pfix must be an array of strings and have a length equal to the p*est arrays. Singlets which are not to have any parameters fixed should be represented as an empty string in the pfix array. So, for example, if one desires to fit three singlets and fix the fwhm of the middle one, one must specify pfix=\"", \"f", \"\"\", the empty strings indicating no parameters of the zeroth and second singlet should be held constant.

In the case of multifit=True, the initial estimates, whether from the p*est parameters or from a file (see below), will be applied to the location of the first fit. This is normally the bottom left corner of the region selected. If masked, not enough good points to perform a fit, or the attempted fit fails, the fitting proceeds to the next pixel with the pixel value of the lowest numbered axis changing the fastest. Once a successful fit has been performed, subsequent fits will use the results of a fit for a nearest pixel for which a previous fit was successful as the initial estimate for the parameters at the current location. The fixed parameter string will be honored for every fit performed when multifit=True.

One specifies what type of PCF profile to fit via the pfunc parameter. A PCF function is one that can be parameterized by a peak, center, and FWHM, as both gaussian and lorentzian singlets can. If all singlets to be fit are gaussians, one can set pfunc equal to the empty string and all singlets will be assumed to be gaussians. If at least one lorentzian is to be fit, pfunc must be specified as a string (in the case of a single singlet) or an array of strings (in the case of multiple singlets). The position of each string corresponds to the positions of
the initial estimates in the \texttt{p*est} and \texttt{pf}ix arrays. Minimal match ("g", "G", "I", or "L") is supported. So, if one wanted to simultaneously fit two gaussian and two lorentzian singlets, the zeroth and last of which were lorentzians, one would specify \texttt{pfunc=["L", "G", "G", "L"]}.

**ESTIMATES FILE FOR GAUSSIAN SINGLETS (NONRECOMMENDED METHOD)** Initial estimates for gaussian singlets can be specified in an estimates file. Estimates files may be deprecated in the future in favor of the \texttt{p*est} parameters, so it is recommended users use those parameters instead. If an estimates file is desired to be used, the \texttt{p*est} parameters must be 0 or empty and \texttt{mgncomps} must be 0 or empty. Only gaussian singlets can be specified in an estimates file. If one desires to fit one or more gaussian multiplets and/or one or more lorentzian singlets simultaneously, the \texttt{p*est} parameters must be used to specify the initial parameters of all gaussian singlets to fit; one cannot use an estimates file in this case. If an estimates file is specified, a polynomial can be fit simultaneously by specifying the \texttt{poly} parameter. The estimates file must contain initial estimates of parameters for all gaussian singlets to be fit. The number of gaussian singlets to fit is gotten from the number of estimates in the file. The file can contain comments which are indicated by a "#" at the beginning of a line. All non-comment lines will be interpreted as initial estimates. The format of such a line is:

\begin{verbatim}
[peak intensity], [center], [fwhm], [optional fixed parameter string]
\end{verbatim}

The first three values are required and must be numerical values. The peak intensity must be expressed in image brightness units, while the center must be specified in pixels offset from the zeroth pixel, and fwhm must be specified in pixels. The fourth value is optional and if present, represents the parameter(s) that should be held constant during the fit. Any combination of the characters 'p' (peak), 'c' (center), and 'f' (fwhm) are permitted, eg "fc" means hold the fwhm and the center constant during the fit. Fixed parameters will have no error associated with them. Here is an example file:

\begin{verbatim}
# estimates file indicating that two gaussians should be fit
# first guassian estimate, peak=40, center at pixel number 10.5, fwhm = 5.8 pixels, all parameters allowed to vary during fit
40, 10.5, 5.8
# second gaussian, peak = 4, center at pixel number 90.2, fwhm = 7.2 pixels, hold fwhm constant
4, 90.2, 7.2, f
# end file
\end{verbatim}

**GAUSSIAN MULTIPLET FITTING** Any number of gaussian multiplets, each containing any number of two or more components, can be simultaneously fit, optionally with a polynomial and/or any number of gaussian and/or lorentzian singlets, the only caveat being that the number of parameters to be fit should be significantly less than the number of data points. The \texttt{mgncomps} parameter indicates the number of multiplets to fit and the number of components in each multiplet. In the case of a single multiplet, an integer (\texttt{1}) can be specified. For example, \texttt{mgncomps=4} means fit a single quadruplet of
gaussians. In the case of 2 or more multiplets, an array of integers (all >1)
must be specified. For example, gmncomps=[2, 4, 3] means 3 separate
multiplets are to be fit, the zeroth being a doublet, the first being a quadruplet,
and the second being a triplet.
Initial estimates of all gaussians in all multiplets are specified via the gm*n*est
parameters which must be arrays of numbers. The order starts with the zeroth
component of the zeroth multiplet to the last component of the zeroth
multiplet, then the zeroth component of the first multiplet to the last
component of the first multiplet, etc to the zeroth component of the last
multiplet to the last element of the last multiplet. The zeroth element of a
multiplet is defined as the reference component of that multiplet and has the
special significance that it is the profile to which all constraints of all other
profiles in that multiplet are referenced (see below). So, in our example of
gmncomps=[2, 4, 3], gmampest, gmcenterest, and gmfwhmest must each be
nine (the total number of individual gaussian profiles summed over all
multiplets) element arrays. The zeroth, second, and sixth elements represent
parameters of the reference profiles in the zeroth, first, and second multiplet,
respectively.
The fixed relationships between the non-reference profile(s) and the reference
profile of a multiplet are specified via the gmampcon, gmcentercon, and
gmfwhmcon parameters. At least one, and any combination, of constraints can
be specified for any non-reference component of a multiplet. The amplitude
ratio of a non-reference line to that of the reference line is set in gmampcon.
The ratio of the fwhm of a non-reference line to that of the reference line is set
in gmfwhmcon. The offset in pixels of the center position of a non-reference
line to that of the reference line is set in gmcentercon. In the case where a
parameter is not constrained for any non-reference line of any multiplet, the
value of the associated parameter must be 0. In the case of a single doublet, a
constraint may be specified as a number or an array of a single number. For
example, mgncomps=2 and gmampcon=0.65 and gmcentercon=[32.4]
means there is a single doublet to fit where the amplitude ratio of the first to the
zeroth line is constrained to be 0.65 and the center of the first line is
constrained to be offset by 32.4 pixels from the center of the zeroth line. In
cases of a total of three or more gaussians, the constraints parameters must be
specified as arrays with lengths equal to the total number of gaussians
summed over all multiplets minus the number of reference lines (one per
multiplet, or just number of multiplets, since reference lines cannot be
constrained by themselves). In the cases where an array must be specified but
a component in that array does not have that constraint, 0 should be specified.
Here’s an example

```
gmncomps=[2, 4, 3] gmampcon=[ 0 , 0.2, 0 , 0.1, 4.5, 0 ]
gcentercon=[24.2, 45.6, 92.7, 0 , -22.8, -33.5] gmfwhmcon=""
```
In this case we have our previous example of one doublet, one quadruplet, and
one triplet. The first component of the doublet has the constraint that its
center is offset by 24.2 pixels from the zeroth (reference) component. The first
component of the quadruplet is constrained to have an amplitude of 0.2 times
that of the quadruplet’s zeroth component and its center is constrained to be offset by 45.6 pixels from the reference component. The second component of the quadruplet is constrained to have its center offset by 92.7 pixels from the associated reference component and the third component is constrained to have an amplitude of 0.1 times that of the associated reference component. The first component of the triplet is constrained to have an amplitude of 4.5 times that of its associated reference component and its center is constrained to be offset by -22.8 pixels from the reference component’s center. The second component of the triplet is constrained to have its center offset by -33.5 pixels from the center of the reference component. No lines have FWHM constraints, so the empty string can be given for that parameter. Note that using 0 to indicate no constraint for line center means that one cannot specify a line centered at the same position as the reference component but having a different FWHM from the reference component. If you must specify this very unusual case, try using a very small positive (or even negative) value for the center constraint.

Note that when a parameter for a line is constrained, the corresponding value for that component in the corresponding gm*est array is ignored and the value of the constrained parameter is automatically used instead. So let’s say, for our example above, we had specified the following estimates:

\[ \text{gmampest} = [1, 2, 1, .1, 3, 2, 5] \]
\[ \text{gmcenterest} = [20, 10, 30, 45, 609, -233, 7, 6.3, 1] \]

Before any fitting is done, the constraints would be taken into account and these arrays would be implicitly rewritten as:

\[ \text{gmampest} = [1, 2, 1, .1, 3, 13.5, 5] \]
\[ \text{gmcenterest} = [20, 44.2, 30, 75.6, 127.7, -233, 30, 7.2, -3.5] \]

The value of gmfwhmest would be unchanged since there are no FWHM constraints in this example.

In addition to be constrained by values of the reference component, parameters of individual components can be fixed. Fixed parameters are specified via the gmfix parameter. If no parameters are to be fixed, gmfix can be specified as the empty string or a zero element array. In the case where any parameter is to be fixed, gmfix must be specified as an array of strings with length equal to the total number of components summed over all multiplets. These strings encode which parameters to be fixed for the corresponding components. If a component is to have no parameters fixed, an empty string is used. In other cases one or more of any combination of parameters can be fixed using "p", "c", and/or "f" described above for fixing singlet parameters. There are a couple of special cases to be aware of. In the case where a non-reference component parameter is constrained and the corresponding reference component parameter is set as fixed, that parameter in the non-reference parameter will automatically be fixed even if it was specified not to be fixed in the gmfix array. This is the only way the constraint can be honored afterall. In the converse case of when a constrained parameter of a non-reference component is specified as fixed, but the corresponding parameter in the reference component is not specified to be fixed, an error will occur.
Fixing an unconstrained parameter in a non-reference component is always legal as is fixing any combination of parameters in a reference component (with the above caveat that corresponding constrained parameters in non-reference components will be silently held fixed as well).

The same rules that apply to singlets when multifit=True apply to multiplets.

**LIMITING RANGES FOR SOLUTION PARAMETERS** In cases of low (or no) signal to noise spectra, it is still possible for the fit to converge, but often to a nonsensical solution. The astronomer can use her knowledge of the source to filter out obviously bogus solutions. Any solution which contains a NaN value as a value or error in any one of its parameters is automatically marked as invalid.

One can also limit the ranges of solution parameters to known ”good” values via the goodamprange, goodcenterrange, and goodfwhmrange parameters. Any combination can be specified and the limit constraints will be ANDed together. The ranges apply to all PCF components that might be fit; choosing ranges on a component by component basis is not supported. If specified, an array of exactly two numerical values must be given to indicate the range of acceptable solution values for that parameter. goodamprange is expressed in terms of image brightness units. goodcenterrange is expressed in terms of pixels from the zeroth pixel in the specified region. goodfwhmrange is expressed in terms of pixels (only non-negative values should be given for FWHM range endpoints). In the case of a multiple-PCF fit, if any of the corresponding solutions are outside the specified ranges, the entire solution is considered to be invalid.

In addition, solutions for which the absolute value of the ratio of the amplitude error to the amplitude exceeds 100 or the ratio of the FWHM error to the FWHM exceeds 100 are automatically marked as invalid.

**POWER LOGARITHMIC POLYNOMIAL AND LOGARITHMIC TRANSFORMED POLYNOMIAL FITTING** Fitting of a single logarithmic polynomial or a single logarithmic transformed polynomial function is supported. No other functions may be fit simultaneously with either of these; if parameters relating to other functions are supplied simultaneously with parameters relating to these functions, an exception will occur. For details of the functional forms, see the introduction of this document.

The set of \( c_0 \ldots c_n \) coefficients (as defined previously) can be solved for. Initial estimates for the \( c \) values should be supplied via the plpest or ltpest parameters, depending on which form is being fit. The number of values given in this array will be the number of coefficients that are solved for. One may specify which coefficients should be held fixed during the fit in the plpfix or ltpfix array. If supplied, this array should have the same number of elements as its respective initial estimates array. A value of True means the corresponding coefficient will be held fixed during the fit. An empty array indicates that no parameters will be held fixed. This is the default. Because the logarithm of the ordinate values must be taken before fitting a logarithmic transformed polynomial, all non-positive pixel values are effectively masked for the purposes of fitting.
INCLUDING STANDARD DEVIATIONS OF PIXEL VALUES

If the standard deviations of the pixel values in the input image are known and they vary in the image (e.g., they are higher for pixels near the edge of the band), they can be included in the sigma parameter. This parameter takes either an array or an image name. The array or image must have one of three shapes: 1. the shape of the input image, 2. the same dimensions as the input image with the lengths of all axes being one except for the fit axis which must have length corresponding to its length in the input image, or 3. be one dimensional with length equal the the length of the fit axis in the input image. In cases 2 and 3, the array or pixels in sigma will be replicated such that the image that is ultimately used is the same shape as the input image. The values of sigma must be non-negative. It is only the relative values that are important. A value of 0 means that pixel should not be used in the fit. Other than that, if pixel A has a higher standard deviation than pixel B, then pixel A is noisier than pixel B and will receive a lower weight when the fit is done. The weight of a pixel is the usual

weight = 1/(sigma*sigma)

In the case of multifit=F, the sigma values at each pixel along the fit axis in the hyperplane perpendicular to the fit axis which includes that pixel are averaged and the resultant averaged standard deviation spectrum is the one used in the fit. Internally, sigma values are normalized such that the maximum value is 1. This mitigates a known overflow issue.

One can write the normalized standard deviation image used in the fit but specifying its name in outsigma. This image can then be used as sigma for subsequent runs.

RETURNED DICTIONARY STRUCTURE

The returned dictionary has a (necessarily) complex structure. First, there are keys "xUnit" and "yUnit" whose values are the abscissa unit and the ordinate unit described by simple strings. Next there are arrays giving a broad overview of the fit quality. These arrays have the shape of the specified region collapsed along the fit axis with the axis corresponding to the fit axis having length of 1:

- attempted: a boolean array indicating which fits were attempted (e.g., if too few unmasked points, a fit will not be attempted).
- converged: a boolean array indicating which fits converged. False if the fit was not attempted.
- valid: a boolean array indicating which solutions fall within the specified valid ranges of parameter space (see section LIMITING RANGES FOR SOLUTION PARAMETERS for details).
- niter: an int array indicating the number of iterations for each profile.
- ncomps: the number of components (gaussian singlets + lorentzian singlets + gaussian multiplets + polynomial) fit for the profile.
- direction: a string array containing the world direction coordinate for each profile.

There is a "type" array having number of dimensions equal to the number of dimensions in the above arrays plus one. The shape of the first n-1 dimensions is the same as the shape of the above arrays. The length of the last dimension is equal to the number of components fit. The values of this array are strings describing the components that were fit at each position ("POLYNOMIAL", "GAUSSIAN", etc.).
"GAUSSIAN" in the case of gaussian singlets, "LORENTZIAN" in the case of lorentzian singlets, and "GAUSSIAN MULTIPLET".

If any gaussian singlets were fit, there will be a subdictionary accessible via the "gs" key which will have subkeys "amp", "ampErr", "center", "centerErr", "fwhm", "fwhmErr", "integral", and "integralErr". Each of these arrays will have one more dimension than the overview arrays described above. The shape of the first n-1 dimensions will be the same as the shape of the arrays described above, while the final dimension will have length equal to the maximum number of gaussian singlets that were fit. Along this axis will be the corresponding fit result or associated error (depending on the array's associated key) of the fit for that singlet component number. In cases where the fit did not converge, or that particular component was excluded from the fit, a value of NAN will be present.

If any lorentzian singlets were fit, their solutions will be accessible via the "ls" key. These arrays follow the same rules as the "gs" arrays described above.

If any gaussian multiplets were fit, there will be subdictionaries accessible by keys "gm0", "gm1", ..., "gmn-1" where n is the number of gaussian multiplets that were fit. Each of these dictionaries will have the same arrays described above for gaussian singlets. The last dimension will have length equal to the number of components in that particular multiplet. Each pixel along the last axis will be the parameter solution value or error for that component number in the multiplet, eg the zeroth pixel along that axis contains the parameter solution or error for the reference component of the multiplet.

The polynomial coefficient solutions and errors are not returned, although they are logged.

If a power logarithmic polynomial was fit, there will be a subdictionary accessible via the "plp" key which will have subkeys "solution" and "error" which will each have an array value. Each of these arrays will have one more dimension than the overview arrays described above. The shape of the first n-1 dimensions will be the same as the shape of the overview arrays described above, while the final dimension will have length equal to the number of parameters that were fit. Along this axis will be the corresponding fit result or associated error (depending on the array's associated key) of the fit. In cases where the fit was not attempted or did not converge, a value of NAN will be present.

OUTPUT IMAGES In addition to the returned dictionary, optionally one or more of any combination of output images can be written. The model and residual parameters indicate the names of the model and residual images to be written; blank values indicate that these images should not be written.

One can also write none, any or all of the solution and error images for gaussian singlet, lorentzian singlet, and gaussian multiplet fits via the parameters amp, amperr, center, centererr, fwhm, fwhmerr, integral, integralerr when doing multi-pixel fits. For a power logarithmic polynomial or a logarithmic transformed polynomial fit, plpsol or ltpsol and plperr or ltpsol are the names of the solution and error images to write, respectively.

These images contain the arrays described for the associated parameter
solutions or errors described in previous sections. Each component is written
to a different image, and each image is distinguished by the component it
represents by its name ending in an underscore and the relevant component
number (".0", ".1", etc). In the case of Gaussian multiplets, the image name
ends with the number of the multiplet group followed by the number of the
component in that group (eg ".3.4" represents component 4 in multiplet group
3). In the case of lorentzian singlets, ".ls" is appended to the image names (but
before the identifying component number), in the case of gaussian multiplets.
Similarly ".gm" is included in the name of Gaussian multiplet images. Pixels
for which fits were not attempted, did not converge, or converged but have
values of NaN (not a number) or INF (infinity) will be masked as bad.
Writing analogous images for polynomial coefficients is not supported.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed Types</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>box</td>
<td>Rectangular region to select in direction plane. See &quot;help par.box&quot; for details. Default is to use the entire direction plane.</td>
<td>string</td>
<td>variant</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>chans</td>
<td>Channels to use. See &quot;help par.chans&quot; for details. Channels must be contiguous. Default is to use all channels.</td>
<td>string</td>
<td>variant</td>
</tr>
<tr>
<td>stokes</td>
<td>Stokes planes to use. See &quot;help par.stokes&quot; for details. Planes must be contiguous. Default is to use all stokes planes.</td>
<td>string</td>
<td>variant</td>
</tr>
<tr>
<td>axis</td>
<td>The profile axis. Default: use the spectral axis if one exists, axis 0 otherwise (&lt;0).</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask. Default is none.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>ngauss</td>
<td>Number of Gaussian elements. Default: 1.</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>poly</td>
<td>Order of polynomial element. Default: do not fit a polynomial (&lt;0).</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>estimates</td>
<td>Name of file containing initial estimates. Default: No initial estimates (&quot;&quot;&quot;).</td>
<td>string</td>
<td>variant</td>
</tr>
<tr>
<td>minpts</td>
<td>Minimum number of unmasked points necessary to attempt fit.</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>multifit</td>
<td>If true, fit a profile along the desired axis at each pixel in the specified region. If false, average the non-fit axis pixels and do a single fit to that average profile. Default False.</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>model</td>
<td>Name of model image. Default: do not write the model image (&quot;&quot;&quot;).</td>
<td>string</td>
<td>variant</td>
</tr>
<tr>
<td>residual</td>
<td>Name of residual image. Default: do not write the residual image (&quot;&quot;&quot;).</td>
<td>string</td>
<td>variant</td>
</tr>
</tbody>
</table>
Returns
record

Example

""
ia.open("myspectrum.im")
res = ia.fitprofile(ngauss=2, box="3,3,4,5", poly=2, multifit=true)
"""
1.1.1 Fit 2-dimensional models to an image.

**Description**

**OVERVIEW**
This application is used to fit one or more two dimensional gaussians to sources in an image as well as an optional zero-level offset. Fitting is limited to a single polarization but can be performed over several contiguous spectral channels. If the image has a clean beam, the report and returned dictionary will contain both the convolved and the deconvolved fit results.

When `dooff` is False, the method returns a dictionary with three keys, 'converged', 'results', and 'deconvolved'. The value of 'converged' is a boolean array which indicates if the fit converged on a channel by channel basis. The value of 'results' is a dictionary representing a component list reflecting the fit results. In the case of an image containing beam information, the sizes and position angles in the 'results' dictionary are those of the source(s) convolved with the restoring beam, while the same parameters in the 'deconvolved' dictionary represent the source sizes deconvolved from the beam. In the case where the image does not contain a beam, 'deconvolved' will be absent. Both the 'results' and 'deconvolved' dictionaries can be read into a component list tool (default tool is named `cl`) using the `fromrecord()` method for easier inspection using tool methods, e.g.
`cl.fromrecord(res['results'])`
although this currently only works if the flux density units are conformant with Jy.

There are also values in each component subdictionary not used by `cl.fromrecord()` but meant to supply additional information. There is a 'peak' subdictionary for each component that provides the peak intensity of the component. It is present for both 'results' and 'deconvolved' components. There is also a 'sum' subdictionary for each component indicated the simple sum of pixel values in the the original image enclosed by the fitted ellipse.

There is a 'channel' entry in the 'spectrum' subdictionary which provides the zero-based channel number in the input image for which the solution applies. In addition, if the image has a beam(s), then there will be a 'beam' subdictionary associated with each component in both the 'results' and 'deconvolved' dictionaries. This subdictionary will have three keys: 'beamarcsec' will be a subdictionary giving the beam dimensions in arcsec, 'beampixels' will have the value of the beam area expressed in pixels, and 'beamster' will have the value of the beam area epressed in steradians. Also, if
the image has a beam(s), in the component level dictionaries will be an 'ispoint' entry with an associated boolean value describing if the component is consistent with a point source.

If dooff is True, in addition to the specified number of gaussians, a zero-level offset will also be fit. The initial estimate for this offset is specified using the offset parameter. Units are assumed to be the same as the image brightness units. The zero level offset can be held constant during the fit by specifying fixoffset=True. In the case of dooff=True, the returned dictionary contains two additional keys, 'zerooff' and 'zeroofferr', which are both dictionaries containing 'unit' and 'value' keys. The values associated with the 'value' keys are arrays containing the the fitted zero level offset value and its error, respectively, for each channel. In cases where the fit did not converge, these values are set to NaN. The value associated with 'unit' is just the image brightness unit.

The region can either be specified by a box(es) or a region. Ranges of pixel values can be included or excluded from the fit. If specified using the box parameter, multiple boxes can be given using the format box="blcx1, blcy1, trcx1, trcy1, blcx2, blcy2, trcx2, trcy2, ... , blcxN, blcyN, trcxN, trcyN" where N is the number of boxes. In this case, the union of the specified boxes will be used.

If specified, the residual and/or model images for successful fits will be written. If an estimates file is not specified, an attempt is made to estimate initial parameters and fit a single Gaussian. If a multiple Gaussian fit is desired, the user must specify initial estimates via a text file (see below for details). The user has the option of writing the result of the fit to a log file, and has the option of either appending to or overwriting an existing file. The user has the option of writing the (convolved) parameters of a successful fit to a file which can be fed back to fitcomponents() as the estimates file for a subsequent run. The user has the option of writing the fit results in tabular format to a file whose name is specified using the summary parameter.

If specified and positive, the value of rms is used to calculate the parameter uncertainties, otherwise, the rms in the selected region in the relevant channel is used for these calculations.

The noisefwhm parameter represents the noise-correlation beam FWHM. If specified as a quantity, it should have angular units. If specified as a numerical value, it is set equal to that number of pixels. If specified and greater than or equal to the pixel size, it is used to calculate parameter uncertainties using the correlated noise equations (see below). If it is specified but less than a pixel width, the the uncorrelated noise equations (see below) are used to compute the parameter uncertainties. If it is not specified and the image has a restoring beam(s), the the correlated noise equations are used to compute parameter uncertainties using the geometric mean of the relevant beam major and minor axes as the noise-correlation beam FWHM. If noisefwhm is not specified and the image does not have a restoring beam, then the uncorrelated noise equations are used to compute the parameter uncertainties.
SUPPORTED UNITS
Currently only images with brightness units conformant with Jy/beam, Jy.km/s/beam, and K are fully supported for fitting. If your image has some other base brightness unit, that unit will be assumed to be equivalent to Jy/pixel and results will be calculated accordingly. In particular, the flux density (reported as Integrated Flux in the logger and associated with the "flux" key in the returned component subdictionary(ies)) for such a case represents the sum of pixel values.

Note also that converting the returned results subdictionary to a component list via cl.fromrecord() currently only works properly if the flux density units in the results dictionary are conformant with Jy. If you need to be able to run cl.fromrecord() on the resulting dictionary you can first modify the flux density units by hand to be (some prefix)Jy and then run cl.fromrecord() on that dictionary, bearing in mind your unit conversion.

If the input image has units of K, the flux density of components will be reported in units of [prefix]K*rad*rad, where prefix is an SI prefix used so that the numerical value is between 1 and 1000. To convert to units of K*beam, determine the area of the appropriate beam, which is given by pi/(4*ln(2))*bmaj*bmin, where bmaj and bmin are the major and minor axes of the beam, and convert to steradians (rad*rad). This value is included in the beam portion of the component subdictionary (key 'beamster'). Then divide the numerical value of the logged flux density by the beam area in steradians. So, for example

```python
# run on an image with K brightness units
res = imfit(...)
# get the I flux density in K*beam of component 0
comp = res['results']['component0']
flux_density_kbeam = comp['flux']['value'][0]/comp['beam']['beamster']
```

FITTING OVER MULTIPLE CHANNELS
For fitting over multiple channels, the result of the previous successful fit is used as the estimate for the next channel. The number of gaussians fit cannot be varied on a channel by channel basis. Thus the variation of source structure should be reasonably smooth in frequency to produce reliable fit results.

MASK SPECIFICATION
Mask specification can be done using an LEL expression. For example mask = "myimage”;5’ will use only pixels with values greater than 5.

INCLUDING AND EXCLUDING PIXELS
Pixels can be included or excluded from the fit based on their values using these parameters. Note that specifying both is not permitted and will cause an error. If specified, both take an array of two numeric values.

ESTIMATES
Initial estimates of fit parameters may be specified via an estimates text file. Each line of this file should contain a set of parameters for a single gaussian. Optionally, some of these parameters can be fixed during the fit. The format of each line is
peak intensity, peak x-pixel value, peak y-pixel value, major axis, minor axis, position angle, fixed

The fixed parameter is optional. The peak intensity is assumed to be in the same units as the image pixel values (e.g., Jy/beam). The peak coordinates are specified in pixel coordinates. The major and minor axes and the position angle are the convolved parameters if the image has been convolved with a clean beam and are specified as quantities. The fixed parameter is optional and is a string. It may contain any combination of the following characters 'f' (peak intensity), 'x' (peak x position), 'y' (peak y position), 'a' (major axis), 'b' (minor axis), 'p' (position angle).

In addition, lines in the file starting with a # are considered comments.

An example of such a file is:

```
# peak intensity must be in map units
120, 150, 110, 23.5arcsec, 18.9arcsec, 120deg
90, 60, 200, 46arcsec, 23arcsec, 140deg, fxp
```

This is a file which specifies that two gaussians are to be simultaneously fit, and for the second gaussian the specified peak intensity, x position, and position angle are to be held fixed during the fit.

ERROR ESTIMATES

Error estimates are based on the work of Condon 1997, PASP, 109, 166. Key assumptions made are: * The given model (elliptical Gaussian, or elliptical Gaussian plus constant offset) is an adequate representation of the data * An accurate estimate of the pixel noise is provided or can be derived (see above). For the case of correlated noise (e.g., a CLEAN map), the fit region should contain many "beams" or an independent value of rms should be provided. * The signal-to-noise ratio (SNR) or the Gaussian component is large. This is necessary because a Taylor series is used to linearize the problem. Condon (1997) states that the fractional bias in the fitted amplitude due to this assumption is of order 1/(S*S), where S is the overall SNR of the Gaussian with respect to the given data set (defined more precisely below). For a 5 sigma "detection" of the Gaussian, this is a 4% effect. * All (or practically all) of the flux in the component being fit falls within the selected region. If a constant offset term is simultaneously fit and not fixed, the region of interest should be even larger. The derivations of the expressions summarized in this note assume an effectively infinite region.

Two sets of equations are used to calculate the parameter uncertainties, based on if the noise is correlated or uncorrelated. The rules governing which set of equations are used have been described above in the description of the noiseFWHM parameter.

In the case of uncorrelated noise, the equations used are

\[
\begin{align*}
\text{f}(A) &= \frac{A}{f(I)} = \frac{f(M)}{f(m)} = \frac{k^*s(x)}{M} = \frac{k^*s(y)}{m} = \frac{(s(p)/\sqrt{2})*((M*M - m*m)/(M*m))}{\sqrt{2}/S} \\
\text{f}(A) &= \frac{A}{f(I)} = \frac{f(M)}{f(m)} = \frac{k^*s(x)}{M} = \frac{k^*s(y)}{m} = \frac{(s(p)/\sqrt{2})*((M*M - m*m)/(M*m))}{\sqrt{2}/S}
\end{align*}
\]

where \( s(z) \) is the uncertainty associated with parameter \( z \), \( f(z) = s(z)/\text{abs}(z) \) is the fractional uncertainty associated with parameter \( z \), \( A \) is the peak intensity,
I is the flux density, M and m are the FWHM major and minor axes, p is the position angle of the component, and \( k = \sqrt{8 \ln(2)} \). \( s(x) \) and \( s(y) \) are the direction uncertainties of the component measured along the major and minor axes; the resulting uncertainties measured along the principle axes of the image direction coordinate are calculated by propagation of errors using the 2D rotation matrix which enacts the rotation through the position angle plus 90 degrees. \( S \) is the overall signal to noise ratio of the component, which, for the uncorrelated noise case is given by
\[
S = \frac{A}{(k^2 h r)} \sqrt{\pi M m}
\]
where \( h \) is the pixel width of the direction coordinate and \( r \) is the rms noise (see the discussion above for the rules governing how the value of \( r \) is determined).

For the correlated noise case, the same equations are used to determine the uncertainties as in the uncorrelated noise case, except for the uncertainty in \( I \) (see below). However, \( S \) is given by
\[
S = \frac{A}{(2^a r N)} \sqrt{\pi M m} \left( 1 + \left( \frac{N^2}{M^2} \right)^a \right) \left( 1 + \left( \frac{N^2}{m^2} \right)^b \right)
\]
where \( N \) is the noise-correlation beam FWHM (see discussion of the noisefwhm parameter for rules governing how this value is determined). "**" indicates exponentiation and \( a \) and \( b \) depend on which uncertainty is being calculated. For sigma(A), \( a = b = 3/2 \). For M and x, \( a = 5/2 \) and \( b = 1/2 \). For m, y, and p, \( a = 1/2 \) and \( b = 5/2 \). \( f(I) \) is calculated in the correlated noise case according to
\[
f(I) = \sqrt{f(A)^2 + (N^2/M^2)(f(M)^2 + f(m)^2)}
\]
Note well the following caveats: * Fixing Gaussian component parameters will tend to cause the parameter uncertainties reported for free parameters to be overestimated. * Fitting a zero level offset that is not fixed will tend to cause the reported parameter uncertainties to be slightly underestimated. * The parameter uncertainties will be inaccurate at low SNR (a \( \sim 10\% \) for SNR = 3). * If the fitted region is not considerably larger than the largest component that is fit, parameter uncertainties may be mis-estimated. * An accurate rms noise measurement, \( r \), for the region in question must be supplied.

Alternatively, a sufficiently large signal-free region must be present in the selected region (at least about 25 noise beams in area) to auto-derive such an estimate. * If the image noise is not statistically independent from pixel to pixel, a reasonably accurate noise correlation scale, \( N \), must be provided. If the noise correlation function is not approximately Gaussian, the correlation length can be estimated using
\[
N = \sqrt{2 \ln(2) / \pi} \sqrt{\frac{\text{double-integral}(dx \ dy \ C(x,y))}{\sqrt{\text{double-integral}(dx \ dy \ C(x,y) \ C(x,y))}}}
\]
where \( C(x,y) \) is the associated noise-smoothing function * If fitted model components have significant spatial overlap, the parameter uncertainties are likely to be mis-estimated (i.e., correlations between the parameters of separate components are not accounted for). * If the image being analyzed is an interferometric image with poor uv sampling, the parameter uncertainties may be significantly underestimated.
The deconvolved size and position angle errors are computed by taking the maximum of the absolute values of the differences of the best fit deconvolved value of the given parameter and the deconvolved size of the eight possible combinations of (FWHM major axis +/- major axis error), (FWHM minor axis +/- minor axis error), and (position angle +/- position angle error). If the source cannot be deconvolved from the beam (if the best fit convolved source size cannot be deconvolved from the beam), upper limits on the deconvolved source size are sometimes reported. These limits simply come from the maximum major and minor axes of the deconvolved gaussians taken from trying all eight of the aforementioned combinations. In the case none of these combinations produces a deconvolved size, no upper limit is reported.

EXAMPLE:
Here is how one might fit two gaussians to multiple channels of a cube using the fit from the previous channel as the initial estimate for the next. It also illustrates how one can specify a region in the associated continuum image as the region to use as the fit for the channel.

```plaintext
imagename = "co_cube.im"
# specify region using region from continuum
region = "continuum.im:source.rgn"
chans = "2~20"
# only use pixels with positive values in the fit
excludepix = [-1e10,0]
# estimates file contains initial parameters for two Gaussians in channel 2
estimates = "initial_estimates.txt"
logfile = "co_fit.log"
# append results to the log file for all the channels
append = "True"
ia.open(imagename)
ia.fitcomponents(region=region, chans=chans, excludepix=excludepix, estimates=estimates, logfile=logfile, append=append)
Arguments
```
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>box</td>
<td>Rectangular region(s) to select in direction plane. See &quot;help par.box&quot; for details. Default is to use the entire direction plane.</td>
<td>string</td>
<td>variant</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>chans</td>
<td>Channels to use. See &quot;help par.chans&quot; for details. Default is 0 (first plane).</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>stokes</td>
<td>The stokes planes to use. See &quot;help par.stokes&quot; for details. Default is to use the first stokes plane.</td>
<td>string</td>
<td>variant</td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask. Default is none.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>includepix</td>
<td>Range of pixel values to include. Default is to include all pixels.</td>
<td>doubleArray</td>
<td>-1</td>
</tr>
<tr>
<td>excludepix</td>
<td>Range of pixel values to exclude. Default is to exclude no pixels.</td>
<td>doubleArray</td>
<td>-1</td>
</tr>
<tr>
<td>residual</td>
<td>Name of the residual image to write. Default is not to write the residual.</td>
<td>string</td>
<td>variant</td>
</tr>
<tr>
<td>model</td>
<td>Name of the model image to write. Default is not to write the model.</td>
<td>string</td>
<td>variant</td>
</tr>
<tr>
<td>estimates</td>
<td>Name of the input estimates file. Default is to auto-estimate in which case a single gaussian will be fit.</td>
<td>string</td>
<td>variant</td>
</tr>
<tr>
<td>logfile</td>
<td>File in which to log results. Default is not to write a logfile.</td>
<td>string</td>
<td>variant</td>
</tr>
<tr>
<td>append</td>
<td>Append results to logfile? Logfile must be specified. Default is to append. False means overwrite existing file if it exists.</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>newestimates</td>
<td>File to which to write results in &quot;estimates&quot; format suitable as estimates input for another run. Default is do not write an estimates file.</td>
<td>string</td>
<td>variant</td>
</tr>
</tbody>
</table>
Returns
record
image.fromrecord.html

image.fromrecord - Function

1.1.1 Generate an image from a record

Description

You can convert an associated image to a record (torecord) or imagepol tool functions will sometimes give you a record. This function (fromrecord) allows you to set the contents of an image tool to the content of the record. This and torecord are used for deserialization and serialization.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>record</td>
<td>Record containing the image</td>
<td>record</td>
<td></td>
</tr>
<tr>
<td>outfile</td>
<td>The name of the diskfile to be created for image from record</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool

Example

```python
"""
#
print "\t----\t fromrecord Ex 1 \t----"
ia.maketestimage('image.large', overwrite=true)
rec=ia.torecord()
ia.close()
ia.fromrecord(rec, "testimage")
```
**image.getchunk - Function**

1.1.1 Get the pixel values from a regular region of the image into an array

**Description**

This function returns the pixels (or optionally the pixel mask) from the image file between blc and trc inclusively. Both float and complex valued images are supported. An increment may be specified with inc. Note that if you retrieve too many pixels, you might cause swapping since the pixels are kept in memory.

Any illegal blc values are set to zero. Any illegal trc values are set to the end of the image. If any trc < blc, you get the whole image for that axis. Any illegal inc values are set to unity.

The argument axes can be used to reduce the dimensionality of the output array. It specifies which pixel axes of the image to average the data over. For example, consider a 3-D image. With axes=[0,1] and all other arguments left at their defaults, the result would be a 1-D vector, a profile along the third axis, with the data averaged over the first two axes.

A related function is getregion which retrieves the pixels or pixel mask from a potentially more complex region-of-interest. Function getchunk is retained because it is faster and therefore preferable for repeated operation in loops if the pixel mask is not required and the region is a simple box.

If you set getmask=T, the return value is the 'pixelmask' rather than the 'pixel' image.

**Arguments**

- **axes**
- **blc**
- **inc**
- **trc**
- **getmask**
### Inputs

- **blc** Bottom-Left-Corner (beginning) of pixel section. Default is start of image.
  - allowed: intArray
  - Default: -1

- **trc** Top-Right-Corner (end) of pixel section. Default is end of image.
  - allowed: intArray
  - Default: -1

- **inc** Increment (stride) along axes
  - allowed: intArray
  - Default: 1

- **axes** Axes to average over. Default is none.
  - allowed: intArray
  - Default: -1

- **list** List bounding box to logger?
  - allowed: bool
  - Default: false

- **dropdeg** Drop degenerate axes?
  - allowed: bool
  - Default: false

- **getmask** Get the pixel mask rather than the pixel values
  - allowed: bool
  - Default: false

### Returns

anyvariant

### Example

Suppose that we have a 3-dimensional image called `{sff im}`. Then:

```python
print "\t\tgetchunk Ex 1 \t----"

ia.fromshape(shape=[64,64,128])
pix = ia.getchunk() # all pixels
ia.calcmask('T')    # give image a mask
pix = ia.getchunk([1,1,1], [10,10,1]) # 10 by 10 section of plane # 1
pix = ia.getchunk([1,1], [1,1]) # first spectrum
pix = ia.getchunk(inc=[1,5]) # all planes, decimated by 5 in y
```
mask = ia.getchunk(getmask=T)  # Get pixelmask
ia.close()
#
"""
image.getregion.html

**image.getregion - Function**

1. Get pixels or mask from a region-of-interest of the image

**Description**

This function recovers the image pixel or **pixel mask** values in the given region-of-interest. Regardless of the shape of the **region** you have specified, the shape of the **pixels** and **pixelmask** arrays must necessarily be the bounding box of the specified region. If the region extends beyond the image, it is truncated.

Recall that the recovered **pixel mask** will reflect both the **pixel mask** stored in the image, and the **region-of-interest** (their masks are ‘anded’) – see the discussion in the introduction about this.

The argument **axes** can be used to reduce the dimensionality of the output array. It specifies which pixel axes of the image to average the data over. For example, consider a 3-D image. With **axes=[0,1]** and all other arguments left at their defaults, the result would be a 1-D vector, a profile along the third axis, with the data averaged over the first two axes.

This function differs in three ways from **getchunk**. First, the region can be much more complex (e.g. a union of polygons) than the simple **blc**, **trc**, and **inc** of **getchunk** (although such a region can be created of course). Second, it can be used to recover the **pixel mask** or the pixels. Third, it is less efficient than **getchunk** for doing the same thing as **getchunk**. So if you are interested in say, iterating through an image, getting a regular hyper-cube of pixels and doing something with them, then **getchunk** will be faster. This would be especially noticeable if you iterated line by line through a large image.

**Arguments**

168
### Inputs

<table>
<thead>
<tr>
<th>region</th>
<th>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>axes</th>
<th>Axes to average over. Default is none.</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>intArray</td>
</tr>
<tr>
<td>Default:</td>
<td>-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mask</th>
<th>Mask to use. See help par.mask. Default is none.</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>variant</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>list</th>
<th>List the bounding box to the logger</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dropdeg</th>
<th>Drop degenerate axes</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>getmask</th>
<th>Get the pixel mask rather than pixel values</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>stretch</th>
<th>Stretch the mask if necessary and possible? See help par.stretch. Default False</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>

### Returns

variant

### Example

Suppose that we have a 3-dimensional image called `cube` and wish to recover the pixel from a simple regular region.

```python
print "\t----\t getregion Ex 1 \t----"
ia.fromshape('cube', [64,64,64], overwrite=true)
#r1=rg.box(blc=[10,10,10],trc=[30,40]) # Create region
r1=rg.box([10,10,10],[30,40,40]) # Create region
pixels=ia.getregion(r1)
```
Example

In this example we recover first the pixels and then the pixel mask.
image.getprofile - Function

1.1.1 Get values and mask for a one dimensional profile along a specified image axis by applying an aggregate function.

Description

This application returns information on a one-dimensional profile taken along a specified image axis. The region of interest is collapsed (a'la ia.collapse) along all axes orthogonal to the one specified, and the specified aggregate function is applied to these pixels to generate the returned values.

The aggregate function must be one of the functions supported by ia.collapse; ie, 'flux', 'max', 'mean', 'median', 'min', 'rms', 'stdev', 'sum', and 'variance'. See the help for ia.collapse() for details regarding these functions. Minimum match and case insensitivity is supported.

One may specify the unit of the returned coordinate values. Unless axis is the spectral axis, unit must be conformant with the corresponding axis unit in the image coordinate system or it must be 'pixel' which signifies, pixel, rather than world, coordinate values should be calculated. If axis is the spectral axis, unit may be a velocity unit (assuming the coordinate system has a rest frequency or restfreq is specified) or a length unit. In these cases, the returned coordinate values will be converted to velocity or wavelength, respectively.

The parameter spectype may be used to specify the velocity or wavelength type for the returned coordinate values if profile is taken along spectral axis.

Supported (minimum match, case insensitive) values are "relativistic velocity", "beta", "radio velocity", "optical velocity", "wavelength", "air wavelength", "default". The "default" value is equivalent to "relativistic" if unit is a velocity unit or "wavelength" if unit is a length unit.

The restfreq parameter allows one to set the rest frequency for the coordinates to be returned if axis is the spectral axis and unit is a velocity unit. If blank, the rest frequency associated with the image coordinate system is used.

The frame allows one to specify which kinematic reference frame that the returned coordinate values should be calculated in. It is only used if axis is the spectral axis and unit is unspecified or is specified and a frequency unit. If blank, the reference frame associated with the image coordinate system is used.

The returned dictionary contains the keys:

values: one-dimensional array along the specified axis containing values resulting from applying the specified aggregate function to corresponding pixels at the same location along that axis. mask: one-dimensional array of booleans of the resulting mask after applying the aggregate function, formed in the same way as that formed by ia.collapse.

coords One-dimensional array
of corresponding coordinate values along the specified axis in the specified unit (or the unit associated with the axis in the image coordinate system if unspecified). xUnit The unit used for calculating the values the coords array.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Default</th>
<th>Allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>Axis along which to determine profile. Must be specified.</td>
<td>int</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
<td></td>
<td>int</td>
</tr>
<tr>
<td>function</td>
<td>Aggregate function to apply for collapse along axes orthogonal to specified axis.</td>
<td>string</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>Default: mean</td>
<td></td>
<td>string</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
<td>any</td>
<td>any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
<td></td>
<td>any</td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask. Default is none.</td>
<td>string</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
<td>string</td>
</tr>
<tr>
<td>unit</td>
<td>Unit of the returned abscissa values. Must either be 'pixel' or be conformant with image axis unit unless axis is the spectral axis. Default is the unit associated with axis in the image coordinate system.</td>
<td>string</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
<td>string</td>
</tr>
<tr>
<td>stretch</td>
<td>Stretch the mask if necessary and possible? See help par.stretch. Default False</td>
<td>bool</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>Default: false</td>
<td></td>
<td>bool</td>
</tr>
<tr>
<td>spectype</td>
<td>Velocity or wavelength type if profile taken along spectral axis. Supported (minimum match, case insensitive) values are &quot;relativistic velocity&quot;, &quot;beta&quot;, &quot;radio velocity&quot;, &quot;optical velocity&quot;, &quot;wavelength&quot;, &quot;air wavelength&quot;, &quot;default&quot;.</td>
<td>string</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>Default: default</td>
<td></td>
<td>string</td>
</tr>
<tr>
<td>restfreq</td>
<td>Rest frequency to use when calculating coordinate values. Used only if axis is spectral axis and unit is not the unit associated with the axis in the coordinate system. Empty string means use the rest frequency associated with the image coordinate system.</td>
<td>any</td>
<td>any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
<td></td>
<td>any</td>
</tr>
<tr>
<td>frame</td>
<td>Reference frame to use when calculating coordinate values. Used only if axis is spectral axis and unit is not the unit associated with the axis in the coordinate system. Empty string means use the reference frame associated with the image coordinate system</td>
<td>string</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>Default: 173</td>
<td></td>
<td>173</td>
</tr>
<tr>
<td>logfile</td>
<td>File to which to write profile.</td>
<td>string</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>Default: 173</td>
<td></td>
<td>173</td>
</tr>
</tbody>
</table>
Returns
record

Example

```python
ia.open('myimage')
# get the max pixel values along axis 2
res = ia.getprofile(axis=2, function='max')

# axis 2 is the spectral axis. Get the minimum pixel values along this axis
# and specify that the returned coordinate values should be optical velocities
# in km/s
res2 = ia.getprofile(axis=2, function='min', unit='km/s', spectype='optical')

ia.done()
```
image.getslice.html

**image.getslice - Function**

Get 1-D slice from the image

**Description**

This function returns a 1-D slice (the pixels and optionally the pixel mask) from the *image file*. The slice is constrained to lie in a plane of two cardinal axes (e.g. XY or YZ). At some point this constraint will be relaxed. A range of interpolation schemes are available.

You specify the slice as a polyline giving the x (x) and y (y) coordinates and the axes of the plane holding that slice (*axes*). As well, you must specify the absolute pixel coordinates of the other axes (*coord*). This defaults to the first pixel (e.g. first plane).

The return value is a record with fields 'pixels' (interpolated intensity), 'mask' (interpolated mask), 'xpos' (x-location in absolute pixel coordinates), 'ypos' (y-location in absolute pixel coordinates), 'distance' (distance along slice in pixels), 'axes' (the x and y axes of slice).

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Polyline x vertices in absolute pixel coordinates</td>
<td>allowed: doubleArray</td>
</tr>
<tr>
<td>y</td>
<td>Polyline y vertices in absolute pixel coordinates</td>
<td>allowed: doubleArray</td>
</tr>
<tr>
<td>axes</td>
<td>Pixel axes of plane holding slice. Default is first two axes.</td>
<td>allowed: intArray</td>
</tr>
<tr>
<td>coord</td>
<td>Specify pixel coordinate for other axes. Default is first pixel.</td>
<td>allowed: intArray</td>
</tr>
<tr>
<td>npts</td>
<td>Number of points in slice. Default is auto determination.</td>
<td>allowed: int</td>
</tr>
<tr>
<td>method</td>
<td>The interpolation method, String from 'nearest', 'linear', 'cubic'</td>
<td>allowed: string</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>record</td>
</tr>
</tbody>
</table>

**Example**

Suppose that we have a 2-dimensional image. Then:

```python
# print "\t----\t getslice Ex 1 \t----"
ia.maketestimage();
rec = ia.getslice (x=[1,20], y=[2,30])  # SLice from [1,2] -> [20,30]
print rec.keys()
#['distance', 'xpos', 'axes', 'mask', 'ypos', 'pixel']
rec = ia.getslice (x=[1,20,25,11], y=[2,30,32,40])  # Polyline slice
ia.close()
#```

176
image.hanning.html

**image.hanning - Function**

1.1.1 Convolve one axis of image with a Hanning kernel

**Description**

This application performs Hanning convolution of one axis of an image defined by
\[ z[i] = 0.25*y[i-1] + 0.5*y[i] + 0.25*y[i+1] \] (equation 1)
where \( z[i] \) is the value at pixel \( i \) in the hanning smoothed image, and \( y[i-1], y[i], \) and \( y[i+1] \) are the values of the input image at pixels \( i-1, i, \) and \( i+1 \) respectively. It supports both float and complex valued images. The length of the axis along which the convolution is to occur must be at least three pixels in the selected region. Masked pixel values are set to zero prior to convolution. All nondefault pixel masks are ignored during the calculation. The convolution is done in the image domain (i.e., not with an FFT).

If drop=False, the length of the output axis will be the same as that of the input axis. The output pixel values along the convolution axis will be related to those of the input values according to equation 1, except the first and last pixels. In that case,
\[ z[0] = 0.5*(y[0] + y[1]) \]
and,
\[ z[N-1] = 0.5*(y[N-2] + y[N-1]) \]
where \( N \) is the number of pixels along the convolution axis. The pixel mask, ORed with the OTF mask if specified, is copied from the selected region of the input image to the output image. Thus for example, if the selected region in the input image has six planes along the convolution axis, and if the pixel values, which are all unmasked, on a slice along this axis are \([1, 2, 5, 10, 17, 26]\), the corresponding output pixel values will be \([1.5, 2.5, 5.5, 10.5, 17.5, 21.5]\).

If drop=True and dmethod="copy", the output image is the image calculated if drop=True, except that only the odd-numbered planes are kept. Furthermore, if the number of planes along the convolution axis in the selected region of the input image is even, the last odd number plane is also discarded. Thus, if the selected region has \( N \) pixels along the convolution axis in the input image, along the convolution axis the output image will have \((N-1)/2\) planes if \( N \) is odd, or \((N-2)/2\) planes if \( N \) is even. In this case, the pixel and mask values are copied directly, without further processing. Thus for example, if the selected region in the input image has six planes along the convolution axis, and if the pixel values, which are all unmasked, on a slice along this axis are \([1, 2, 5, 10, 17, 26]\), the corresponding output pixel values will be \([2.5, 10.5]\).
If drop=True and dmethod="mean", first the image described in the
drop=False case is calculated. The first plane and last plane(s) of that image
are then discarded as described in the drop=True, dmethod="copy" case.
Then, the ith plane of the output image is calculated by averaging the \((2^i)\)th
and \((2^i + 1)\)th planes of the intermediate image. Thus for example, if the
selected region in the input image has six planes along the convolution axis,
and if the pixel values, which are all unmasked, on a slice along this axis are
\([1, 2, 5, 10, 17, 26]\), the corresponding output pixel values will be \([4.0, 14.0]\).
Masked values are taken into consideration when forming this average, so if
one of the values is masked, it is not used in the average. If at least one of the
values in the input pair is not masked, the corresponding output pixel will not
be masked.
The hanning smoothed image is written to disk with name outfile, if
specified. If not, no image is written but the image is still accessible via the
returned image analysis tool (see below).
This method always returns an image analysis tool which is attached to the
hanning smoothed image. This tool should always be captured and closed
after any desired manipulations have been done. Closing the tool frees up
system resources (eg memory), eg,
\[
\text{hanning\_image} = \text{ia.hanning(\ldots)}
\]

\[
\# \text{ do things (or not) with hanning\_image}
\]

\[
\ldots
\]

\[
\# \text{ close the returned tool promptly upon finishing with it.}
\]

\[
\text{hanning\_image.\_done(\ldots)}
\]

See also the other convolution functions convolve2d, sepconvolve and convolve.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed Types</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
<td>string</td>
<td>unset</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. Default is none.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>axis</td>
<td>Zero based axis to convolve. ia.coordsys().names() gives the order of the axes in the image. Less than 0 means use the spectral axis if there is one, if not an exception is thrown.</td>
<td>int</td>
<td>-10</td>
</tr>
<tr>
<td>drop</td>
<td>Drop every other pixel on output?</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>stretch</td>
<td>Stretch the mask if necessary and possible? See help par.stretch. Default False</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>dmethod</td>
<td>If drop=True, method to use in plane decimation. &quot;copy&quot;: direct copy of every second plane, &quot;mean&quot;: average planes 2<em>i and 2</em>i+1 in the smoothed, non-decimated image to form plane i in the output image.</td>
<td>string</td>
<td>copy</td>
</tr>
</tbody>
</table>

**Returns**

image

**Example**

180
ia.open("mynonsmoothed.im")
# smooth the spectral axis, say it's axis 2 and only write every other pixel
hanning = ia.hanning(outfile="myhanningsmoothed.im", axis=2, drop=True, overwrite=True)
# done with input
ia.done()
# do something with the output image, get statistics say
stats = hanning.statistics()
# close the result image
hanning.done()
image.haslock.html

**image.haslock - Function**

Does this image have any locks set?

**Description**

This function can be used to find out whether the image has a read or a write lock set. It is not of general user interest. It returns a vector of Booleans of length 2. Position 1 says whether a read lock is set, position 2 says whether a write lock is set.

In general locking is handled automatically, with a built in lock release cycle. However, this function can be useful in scripts when a file is being shared between more than one process. See also functions unlock and lock.

**Arguments**

**Returns**

boolArray

**Example**

```r
# print "\t----\t haslock Ex 1 \t----"
ia.maketestimage('xx',overwrite=true)
ia.lock(writelock=T)
print ia.haslock() #[True, True]
ia.unlock()
print ia.haslock() #[False, False]
ia.lock(F)
```
print ia.haslock()
# [True, False]
ia.close()
#

"""

This example acquires a read/write lock on the file and then unlocks it and acquires just a read lock.

"""
image.histograms.html

**image.histograms - Function**

1.1.1 Compute histograms from the image

**Description**

This method computes histograms of the pixel values in the image. The values are returned in a dictionary.

The chunk of the image over which you compute the histograms is specified by a vector of axis numbers (argument `axes`). For example, consider a 3-dimensional image for which you specify `axes=[0,2]`. The histograms would be computed for each XZ (axes 0 and 2) plane in the image. You could then examine those histograms as a function of the Y (axis 1) axis. Or perhaps you set `axes=[2]`, whereupon you could examine the histogram for each Z (axis 2) profile as a function of X and Y location in the image.

You have control over the number of bins for each histogram (`nbins`). The bin width is worked out automatically for each histogram and may vary from histogram to histogram (the range of pixel values is worked out for each chunk being histogrammed).

You have control over which pixels are included in the histograms via the `includepix` argument. This vector specifies a range of pixel values to be included in the histograms. If you only give one value for this, say `includepix=[b]`, then this is interpreted as `includepix=[-abs(b),abs(b)]`.

If you specify an inclusion range, then the range of pixel intensities over which the histograms are binned is given by this range too. This is a way to make the bin width the same for each histogram.

You can control if the histogram is cumulative or non-cumulative via the `cumu` parameter.

You have control over how the bin counts are returned. If `log = false`, the actual counts are returned. If true, the values returned are the log10 values of the actual counts.

The results are returned as a dictionary. The counts (field "counts") and the abscissa values (field "values") for all bins in each histogram are returned. The shape of the first dimension of those arrays contained in those fields is `nbins`.

The number and shape of the remaining dimensions are those of the display axes (the axes in the image for which you did not compute the histograms). For example, if one has a three dimensional image and sets `axes=[2]`, the display axes are 0 and 1, so the shape of each counts and values array is then `[nbins,nx,ny]`, where nx and ny are the length of the zeroth and first axes, respectively.
In addition, the mean (field "mean") and standard deviation (field "sigma") computed using the data in each histogram is returned. The shape of these arrays is equal to the shape of the display axes. So,

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axes</td>
<td>List of axes to compute histograms over. Default is all axes.</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask. Default is none.</td>
</tr>
<tr>
<td>nbins</td>
<td>Number of bins in histograms, &gt; 0</td>
</tr>
<tr>
<td>includepix</td>
<td>Range of pixel values to include. Default is to include all pixels.</td>
</tr>
<tr>
<td>cumu</td>
<td>If T the bin values are cumulative.</td>
</tr>
<tr>
<td>log</td>
<td>If true, the returned counts values will be the log10 values of the actual counts, if false, the actual counts will be returned.</td>
</tr>
<tr>
<td>stretch</td>
<td>Stretch the mask if necessary and possible? See help par.stretch. Default False</td>
</tr>
</tbody>
</table>

**Returns**

record
Example

```python
# obtain a histogram using the entire image
ia.maketestimage()
res = ia.histograms()
ia.close()

# obtain histograms for each plane along axis 1 with each
# histogram having 30 bins. Only pixel values in the range
# -0.001 to 0.001 are used in computing the histograms and the
# statistics. The counts in the returned data structure represent
# the cumulative number of data points in the current bin and in
# bins less than the current bin.
ia.open("myimage.im")
r = ia.histograms(axes=[0,2],nbins=30,includepix=1e-3,cumu=T)
ia.close()
```
image.history.html

**image.history - Function**

[1.1.1] Recover and/or list the history file

**Description**

This function allows you to access the history file.
If `browse=F` and `list=F`, the history is returned by the function as a vector of strings. If `list=T`, the history is sent to the logger.

**CASA** tools that modify the MeasurementSet or an image file will save history information. Also, you can directly annotate the history file with the function `sethistory`. History from FITS file conversions is also stored and listable here.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>List history to the logger?</td>
<td>bool</td>
<td>true</td>
</tr>
</tbody>
</table>

**Returns**

`stringArray`

**Example**

```python
#
print "\t----\t history Ex 1 \t----"
ia.maketestimage()
ia.history()  # List history to logger
h = ia.history(list=F)  # Recover history in variable h
ia.history(list=T, browse=F)  # List history to logger
#
"""
```

187
image.insert - Function

Description

This function inserts the specified image (or part of it) into the image referenced by this tool. The specified image may be given via argument `infile` as a disk file name (it may be in native CASA, FITS, or Miriad format; Look here for more information on foreign images).

If the `locate` vector is not given, then the images are aligned (to an integer pixel shift) by their reference pixels.

If `locate` vector is given, then those values that are given, give the absolute pixel in the output (this) image of the bottom left corner of the input (sub)image. For those values that are not given, the input image is symmetrically placed in the output image.

The image referenced by this tool is modified in place; no new image is created. The method returns True if successful.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of image to be inserted.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>infile</code></td>
<td>allowed: string</td>
</tr>
<tr>
<td><code>region</code></td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
</tr>
<tr>
<td><code>locate</code></td>
<td>Location of input image in output image. Default is centrally located.</td>
</tr>
<tr>
<td><code>verbose</code></td>
<td>Emit informational messages to logger?</td>
</tr>
</tbody>
</table>

Returns

189
Example

"""
#
print "\t----\tinsert Ex 1 \t----"
ia.maketestimage('myfile.insert', overwrite=true)
ia.close()
ia.fromshape(shape=[200,200])
ia.insert(infile='myfile.insert')  # Align by reference pixel
ia.newimagefromfile('myfile.insert')
ia.insert(infile=im2.name(), locate=[])  # Align centrally
# This time align axis 0 as given and axis 1 centrally
ia.insert(infile='myfile.insert', locate=[20])
ia.close()  # close default tool and
"""
image.isopen.html

**image.isopen - Function**

Is this Image tool open?

**Description**

This function can be used to find out whether the Image tool is associated with an image or not.

**Arguments**

- None

**Returns**

- bool

**Example**

```python
# Example
print "\t----\t isopen Ex 1 \t----"
ia.maketestimage('zz',overwrite=true)
print ia.isopen()
#True
ia.close()
print ia.isopen()
#False
ia.open('zz')
print ia.isopen()
#True
ia.close()
#`
```
image.ispersistent.html

image.ispersistent - Function

1.1.1 Is the image persistent?

Description

This function can be used to find out whether the image is persistent on disk or not. There is a subtle difference from the image being virtual. For example, a virtual image which references another which is on disk is termed persistent.

Arguments

Returns

bool

Example

```python
# print "\t----\t ispersistent Ex 1 \t----"
ia.fromshape(outfile='tmp', shape=[10,20], overwrite=true)
print ia.ispersistent()
#True
ia.close()
ia.fromimage(infile='tmp')
print ia.ispersistent()
#True
im3 = ia.subimage()
print im3.ispersistent()  # Persistent virtual image !
#True
im4 = ia.imagecalc(pixels='tmp+tmp')
print im4.ispersistent()
```
#False
im3.done()
im4.done()
ia.close(remove=true)
#
"""
image.lock.html

image.lock - Function

Acquire a lock on the image

Description

This function can be used to acquire a Read or a Read/Write lock on the image file. It is not of general user interest.
In general locking is handled automatically, with a built in lock release cycle. However, this function can be useful in scripts when a file is being shared between more than one process. See also functions unlock and haslock.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>writelock</td>
<td>Acquire a read/write (T) or a readonly (F) lock</td>
</tr>
<tr>
<td>allowed</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>nattempts</td>
<td>Number of attempts, &gt; 0. Default is unlimited.</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
</tbody>
</table>

Returns

bool

Example

"""
#
print "\t----\t lock Ex 1 \t----"
ia.maketestimage('xx', overwrite=true)
ia.lock(writelock=T)
ia.unlock()
ia.lock(writelock=F)
ia.close(remove=true)
"""
This acquires a read/write lock on the file. Then we unlock it and acquire a readonly lock.
image.makecomplex.html

**image.makecomplex - Function**

1.1.1 Make a complex image

**Description**

This function combines the current image with another image to make a complex image. The current image (i.e. that associated with this Image tool is assumed to be the Real image). You supply the Imaginary image; it must be disk-based at this time.

The output image cannot be associated with an Image tool (does not handle Complex images yet) and so the best you can do is write it to disk. The Viewer can view it.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output Complex (disk) image file name</td>
<td>string</td>
</tr>
<tr>
<td>imag</td>
<td>Imaginary image file name</td>
<td>string</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
<td>any</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file?</td>
<td>bool</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**
# makecomplex Ex 1

print "\t----\t makecomplex Ex 1 \t----"

# Imaginary image
ia.maketestimage('imag.im', overwrite=true)

# Associated real image
ia.maketestimage('real.im', overwrite=true)

# Complex image
ia.makecomplex('complex.im', 'imag.im', overwrite=true)

#
image.maskhandler.html

image.maskhandler - Function

1.1.1 Handle pixel masks

Description

This function is used to manage or handle pixel masks. A CASA image may contain zero, one or more pixel masks. Any of these masks can be designated the default pixel mask. The default mask is acted upon by CASA applications. For example, if you ask for statistics from an image, pixels which are masked as bad (F) will be excluded from the calculations.

This function has an argument (op) that specifies the behaviour. In all cases, you can shorten the operation string to three characters. It is not the job of this function to modify the values of masks.

- **default** - this retrieves the name of the default pixel mask as the return value of the function call.
- **get** - this retrieves the name(s) of the existing pixel masks as the return value of the function call (string or vector of strings).
- **set** - this lets you change the default pixel mask to that given by the name argument. If name is empty, then the default mask is unset (i.e. an all good mask is effectively applied).
- **delete** - this lets you delete the pixel masks specified by the name argument. To delete more than one mask, name can be a vector of strings. Any supplied pixel mask name that does not exist is silently ignored.
- **rename** - this lets you rename the mask specified by name[0] to name[1]. Thus the name argument must be a vector of length 2.
- **copy** - this lets you copy a mask to another in the same image, or copy a mask from another image into this image. Thus the name argument must be a vector of length 2.

For the first case, the first element of name must be the name of the mask to copy, and the second element must be the name of the pixel mask to which it will be copied.

For the second case, the first element of name must be the name of the input image and pixel mask with a colon delimiter (e.g. `hcn:mask2`). The second element must be the name of the pixel mask to which the input pixel mask will be copied.
Use the summary function to see the available pixel masks. You can do this either via the logger display, or via the returned record, which contains the mask names. In the logger display, any pixel mask which is not the default mask is listed in square brackets. If a default mask is set, it is listed first, and is not enclosed in square brackets.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>The operation. One of 'set', 'delete', 'rename', 'get', 'copy' or 'default'</td>
<td>string</td>
<td>default</td>
</tr>
<tr>
<td>name</td>
<td>Name of mask or masks.</td>
<td>stringArray</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

stringArray

**Example**

```python
""
# print "\t----\t maskhandler Ex 1 \t----"
ia.maketestimage('g1.app', overwrite=true)
ia.calcmask('T', name='mask1')
ia.close()
ia.maketestimage('myimage', overwrite=true)
ia.calcmask('T') # Create some masks
da.calcmask('T', name='mask1')
da.calcmask('T', name='mask2')
names = ia.maskhandler('get') # Get the mask names
print names
#['mask0', 'mask1', 'mask2']
name = ia.maskhandler('default') # Get the default mask name
print name
#mask2
ia.maskhandler('set', ['mask1']) # Make 'mask1' the default mask
```
ia.maskhandler('set', [''])  # Unset the default mask
ia.maskhandler('delete', ['mask1'])  # Delete 'mask1'
ia.calcmask('T', name='mask1')  # Make another 'mask1'
ia.maskhandler('delete', ['mask0', 'mask1'])  # Delete 'mask0' and 'mask1'
ia.calcmask('T', name='mask1')
ia.maskhandler('rename', ['mask1', 'mask0'])  # Rename 'mask1' to 'mask0'

# Copy 'mask1' from image 'g1.app' to 'mask10' in image 'myimage'
ia.maskhandler('copy', ['g1.app:mask1', 'mask10'])
ia.removefile('g1.app')  # Cleanup
ia.close()  
#  
"""
Get the miscellaneous information record from an image

Description

A CASA image file can accumulate miscellaneous information during its lifetime. This information is stored in a record called the miscinfo record. For example, the FITS filler puts header keywords it doesn’t otherwise use into the miscinfo record. This miscinfo record is not guaranteed to have any entries, so it’s up to you to check for any fields that you require.

You can also put things into this record (see setmiscinfo) yourself, to keep information that the system might not otherwise store for you.

When the image is written out to FITS, the items in the miscinfo record are written to the FITS file as keywords with the corresponding record field name.

Arguments

Returns

record

Example

```
#
print "\t----\t miscinfo Ex 1 \t----"
ia.maketestimage()
print ia.miscinfo()    # print the record
ia.setmiscinfo("testing")
print ia.miscinfo()    # print the record
header = ia.miscinfo()    # capture the record for further use
print header
```
ia.close()
#
###
image.modify.html

**image.modify - Function**

1.1.1 Modify image with a model

**Description**

This function applies a model of the sky to the image. You can add or subtract the model which is contained in a ComponentList tool. The pixel values are only changed where the total mask (combination of the default **pixel mask** [if any] and the OTF mask) is good (True). If the computation fails for a particular pixel (e.g. coordinate undefined) that pixel will be masked bad.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>model</strong></td>
<td>Record representation of a ComponentList model</td>
</tr>
<tr>
<td><strong>allowed:</strong></td>
<td>record</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>record</td>
</tr>
<tr>
<td><strong>region</strong></td>
<td>Region selection. See ”help par.region” for details. Default is to use the full image.</td>
</tr>
<tr>
<td><strong>allowed:</strong></td>
<td>any</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>variant</td>
</tr>
<tr>
<td><strong>mask</strong></td>
<td>Mask to use. See help par.mask. Default is none.</td>
</tr>
<tr>
<td><strong>allowed:</strong></td>
<td>any</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>variant</td>
</tr>
<tr>
<td><strong>subtract</strong></td>
<td>Subtract or add the model</td>
</tr>
<tr>
<td><strong>allowed:</strong></td>
<td>bool</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>true</td>
</tr>
<tr>
<td><strong>list</strong></td>
<td>List informative messages to the logger</td>
</tr>
<tr>
<td><strong>allowed:</strong></td>
<td>bool</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>true</td>
</tr>
<tr>
<td><strong>stretch</strong></td>
<td>Stretch the mask if necessary and possible? See help par.stretch. Default False</td>
</tr>
<tr>
<td><strong>allowed:</strong></td>
<td>bool</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

204
Example

```python
#
print "\t----\t modify Ex 1 \t----"
ia.maketestimage()
clrec = ia.fitcomponents()
ia.modify(clrec['results'])
ia.close()
#
""
```

In this example we subtract the model returned by the fitcomponents function.
image.maxfit - Function

1.1.1 Find maximum and do parabolic fit in the sky

Description

This function finds the pixel with the maximum value in the region, and then uses function findsources to generate a Componentlist with one component. The component will be of type Point (point=T) or Gaussian (point=F). If negfind=F the maximum pixel value is found in the region and fit. If negfind=T the absolute maximum pixel value is found in the region and fit. See function findsources for a description of arguments point and width. See also the function fitcomponents.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>point</td>
<td>Find only point sources?</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>width</td>
<td>Half-width of fit grid when point=F</td>
<td>int</td>
<td>5</td>
</tr>
<tr>
<td>negfind</td>
<td>Find negative sources as well as positive?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>list</td>
<td>List the fitted parameters to the logger?</td>
<td>bool</td>
<td>true</td>
</tr>
</tbody>
</table>

Returns

record
Example

"""
#
print "\t--\t maxfit Ex 1 \t--"
ia.maketestimage()
clrec = ia.maxfit()
print clrec  # There is only one component
ia.close()
#
"""
Description

Summary
The primary goal of this function is to enable you to analyze a multi-dimensional image by generating moments of a specified axis. This is a time-honoured spectral-line analysis technique used for extracting information about spectral lines.

You can generate one or more output moment images. The return value of this function is an on-the-fly Image tool holding the first of the output moment images.

The word ‘moment’ is used loosely here. It refers to collapsing an axis (the moment axis) to one pixel and setting the value of that pixel (for all of the other non-collapsed axes) to something computed from the data values along the moment axis. For example, take an RA-DEC-Velocity cube, collapse the velocity axis by computing the mean intensity at each RA-DEC pixel. This function offers many different moments and a variety of automatic methods to compute them.

We try to make a distinction between a ‘moment’ and a ‘method’. This boundary is a little blurred, but it claims to refer to the distinction between what you are computing, and how the pixels that were included in that computation were selected. For example, a ‘moment’ would be the average value of some pixel values in a spectrum. A ‘method’ for selecting those pixels would be a simple pixel value range specifying which pixels should be included.

There are many available moments, and you specify each one with an integer code as it would get rather cumbersome to refer to them via strings. In the list below, the value of the $i$th pixel of the spectrum is $I_i$, the coordinate of this pixel is $v_i$ (of course it may not be velocity), and there are $n$ pixels in the spectrum. The available moments are:

- $-1$ – the mean value of the spectrum
  \[ \frac{1}{n} \sum I_i \]

- $0$ – the integrated value of the spectrum
  \[ M_0 = \Delta v \sum I_i \]

208
where $\Delta v$ is the width (in world coordinate units) of a pixel along the moment axis.

- **1** – the intensity weighted coordinate (this is traditionally used to get ‘velocity fields’)

  $$M_1 = \frac{\sum I_i v_i}{M_0}$$

- **2** – the intensity weighted dispersion of the coordinate (this is traditionally used to get ‘velocity dispersion fields’)

  $$\sqrt{\frac{\sum I_i (v_i - M_1)^2}{M_0}}$$

- **3** – the median of $I$

- **4** – the median coordinate. Here we treat the spectrum as a probability distribution, generate the cumulative distribution, and then find the coordinate corresponding to the 50% value. This moment is not very robust, but it is useful for quickly generating a velocity field in a way that is not sensitive to noise. However, it will only give sensible results under certain conditions. The generation of the cumulative distribution and the finding of the 50% level really only makes sense if the cumulative distribution is monotonic. This essentially means only selecting pixels which are positive or negative. For this reason, this moment type is only supported with the basic method (see below – i.e. no smoothing, no windowing, no fitting) with a pixel selection range that is either all positive, or all negative

- **5** – the standard deviation about the mean of the spectrum

  $$\sqrt{\frac{1}{(n-1)} \sum (I_i - \bar{I})^2}$$

- **6** – the root mean square of the spectrum

  $$\sqrt{\frac{1}{n} \sum I_i^2}$$
- 7 – the absolute mean deviation of the spectrum
  \[ \frac{1}{n} \sum |(I_i - \bar{I})| \]
- 8 – the maximum value of the spectrum
- 9 – the coordinate of the maximum value of the spectrum
- 10 – the minimum value of the spectrum
- 11 – the coordinate of the minimum value of the spectrum

Smoothing
The purpose of the smoothing functionality is purely to provide a mask. Thus, you can smooth the input image, apply a pixel include or exclude range, and generate a smoothed mask which is then applied before the moments are generated. The smoothed data are not used to compute the actual moments; that is always done from the original data.

Basic Method
The basic method is to just compute moments directly from the pixel values. This can be modified by applying pixel value inclusion or exclusion ranges (arguments `includepix` and `excludepix`). You can then also convolve the image (arguments `smoothaxes`, `smoothtypes`, and `smoothwidths`) and find a mask based on the inclusion or exclusion ranges applied to the convolved image. This mask is then applied to the unsmoothed data for moment computation.

Window Method
The window method (invoked with argument `method='window'`) does no pixel-value-based selection. Instead a window is found (hopefully surrounding the spectral line feature) and only the pixels in that window are used for computation. This window can be found from the convolved or unconvolved image (arguments `smoothaxes`, `smoothtypes`, and `smoothwidths`). The moments are always computed from the unconvolved data. The window can be found (for each spectrum) automatically. The automatic methods are via Bosma’s converging mean algorithm (method='window') or by fitting Gaussians and taking ±3σ as the window (method='window,fit'). In Bosma’s algorithm, an initial guess for a range of pixels surrounding a spectral feature is refined by widening until the mean of the pixels outside of the range converges (to the noise).

Fit Method
The fit method (method='fit') fits Gaussians to spectral features automatically. The moments are then computed from the Gaussian fits (not the data themselves).

Other Arguments

- **outfile** - If you are creating just one moment image, and you specify outfile, then the image is created on disk with this name. If you leave outfile empty then a temporary image is created. In both cases, you can access this image with the returned Image tool. If you are making more than one moment image, then these images are always created on disk. If you specify outfile then this is the root for the output file names. If you don’t specify it, then the input image name is used as the root.

- **smoothing** - If you smooth the image to generate a mask, you specify the kernel widths via the smoothwidths argument in the same way as in the sepconvolve function. See it for details.

- **stddev** - Some of the automatic methods also require an estimate of the noise level in the image. This is used to assess whether a spectrum is purely noise or not, and whether there is any signal worth digging out. If you don’t give it via the stddev argument, it will be worked out automatically from a Gaussian fit to the bins above 25% from a histogram of the entire image.

- **includepix, excludepix** - The vectors given by arguments includepix and excludepix specify a range of pixel values for which pixels are either included or excluded. They are mutually exclusive; you can specify one or the other, but not both. If you only give one value for either of these, say includepix=b, then this is interpreted as includepix=[-abs(b),abs(b)].

The convolving point-spread function is normalized to have a volume of unity. This means that point sources are depressed in value, but extended sources that are large with respect to the PSF remain essentially on the same intensity scale; these are the structures you are trying to find with the convolution so this is what you want. If you convolve the image, then arguments like includepix select based upon the convolved image pixel values. If you are having trouble getting these right, you can output the convolved image (smoothout) and assess the validity of your pixel ranges. Note also that if you are Hanning convolving (usually used on a velocity axis), then the width for this kernel must be 3 pixels (triangular smoothing kernels of other widths have no valid theoretical basis).

- **doppler** - If you compute the moments along a spectral axis, it is conventional to compute the world coordinate (needed for moments 0, 1
and 2) along that axis in "km/s". The argument doppler lets you specify what doppler convention the velocity will be calculated in. You can choose from doppler=radio, optical, true. See function summary for the definitions of these codes. For other moment-axis types, the world coordinate is computed in the native units.

- **mask** - The total input mask is the combination of the default pixel mask (if any) and the OTF mask. Once this mask has been established, then the moment method may make additional pixel selections.

- **drop** - If this is true (the default) then the moment axis is dropped from the output image. Otherwise, the output images have a moment axis of unit length and coordinate information that is the same as for the input image. This coordinate information may be totally meaningless for the moment images.

Finally, if you ask for a moment which requires the coordinate to be computed for each profile pixel (these are the intensity weighted mean coordinate [moment 1] and the intensity weighted dispersion of the coordinate [moment 2]), and the profile axis is not separable then there will be a performance loss. Examples of non-separable axes are RA and Dec. If the axis is separable (e.g. a spectral axis) there is no penalty. In the latter case, the vector of coordinates for one profile is the same as the vector for another profile, and it can be precomputed (once).

Note that this function has no "virtual" output file capability. All output files are written to disk. The output mask for these images is good (T) unless the moment method fails to generate a value (e.g. the total input pixel mask was all bad for the profile) in which case it will be bad (F).

If an image has multiple (per-channel beams) and the moment axis is equal to the spectral axis, each channel will be convolved with a beam that is equal to the beam having the largest area in the beamset prior to moment determination.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>List of moments that you would like to compute. Default is integrated spectrum.</th>
</tr>
</thead>
<tbody>
<tr>
<td>moments</td>
<td>allowed: intArray</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
<tr>
<td>axis</td>
<td>The moment axis. Default is the spectral axis if there is one.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: -10</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask. Default is none.</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
<tr>
<td>method</td>
<td>List of windowing and/or fitting functions you would like to invoke. Vector of strings from 'window' and 'fit'. The default is to not invoke the window or fit functions.</td>
</tr>
<tr>
<td></td>
<td>allowed: stringArray</td>
</tr>
<tr>
<td></td>
<td>Default:.variant</td>
</tr>
<tr>
<td>smoothaxes</td>
<td>List of axes to smooth. Default is no smoothing.</td>
</tr>
<tr>
<td></td>
<td>allowed: intArray</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td>smoothtypes</td>
<td>List of smoothing kernel types, one for each axis to smooth. Vector of strings from 'gauss', 'boxcar', 'hanning'. Default is no smoothing.</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
<tr>
<td>smoothwidths</td>
<td>List of widths (full width for boxcar, full width at half maximum for gaussian, 3 for Hanning) in pixels for the smoothing kernels. Vector of numeric. Default is no smoothing.</td>
</tr>
<tr>
<td></td>
<td>allowed: doubleArray</td>
</tr>
<tr>
<td></td>
<td>Default: 0.0</td>
</tr>
<tr>
<td>includepix</td>
<td>Range of pixel values to include. Vector of 1 or 2 doubles. Default is include all pixel.</td>
</tr>
<tr>
<td></td>
<td>allowed: doubleArray</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td>excludepix</td>
<td>Range of pixel values to exclude. Default is exclude no pixels.</td>
</tr>
<tr>
<td></td>
<td>allowed: doubleArray</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td>peaksnr</td>
<td>The SNR ratio below which the spectrum will be rejected as noise (used by the window and fit functions only)</td>
</tr>
<tr>
<td></td>
<td>allowed: double</td>
</tr>
<tr>
<td></td>
<td>Default: 3.0</td>
</tr>
<tr>
<td>stddev</td>
<td>Standard deviation of the noise signal in the image (used by the window and fit functions only)</td>
</tr>
<tr>
<td></td>
<td>allowed: double</td>
</tr>
<tr>
<td></td>
<td>Default: 0.0</td>
</tr>
<tr>
<td>doppler</td>
<td>Velocity doppler definition for velocity computations along spectral axes</td>
</tr>
</tbody>
</table>
Returns
image

Example

```
#
print "\t----\t moments Ex 1 \t----"
ia.fromshape(shape=[32,32,32,32]) # replace with your own cube
im2 = ia.moments(moments=[-1,1,2], axis=2, smoothaxes=[0,1,2],
                 smoothtypes=['gauss','gauss','hann'],
                 smoothwidths=[5.0,5.0,3], excludepix=[1e-3],
                 smoothout='smooth', overwrite=true)
im2.done()
ia.close()
#
```

In this example, standard moments (average intensity, weighted velocity
and weighted velocity dispersion) are computed via the convolve (spatially
convolved by gaussians and spectrally by a Hanning kernel) and clip
method (we exclude any pixels with absolute value less than $0.001$).
The output file names are automatically created for us and
the convolved image is saved. The returned image tool holds the first
moment image.

Example

```
#
print "\t----\t moments Ex 2 \t----"
ia.fromshape(shape=[32,32,32,32])
im2 = ia.moments(moments=[3], method=['window'])
im2.done()
```
In this example, the median of each spectrum is computed, after pixel selection by the automatic window method. The output image is temporary and accessed via the returned Image tool.
image.name.html

**image.name - Function**

[1.1.1] Name of the image file this tool is attached to

**Description**

This function returns the name of the *image file*. By default, this function returns the full absolute path of the *image file*. You can strip this path off if you wish with the *strippath* argument and just recover the *image file* name itself.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strippath</td>
<td>Strip off the path before the actual file name?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

[string]

**Example**

```python
#
print "\t----\t name Ex 1 \t----"
ia.maketestimage('g1.app', overwrite=true)
print ia.name(strippath=F)
#/casa/code/xmlcasa/implement/images/scripts/g1.app
print ia.name(strippath=T)
#g1.app
ia.close()
#```
image.open.html

**image.open - Function**

1.1.1 Open a new image file with this image tool

**Description**

Use this function when you are finished analyzing the current image file and want to attach to another one. This function detaches the image tool from the current image file, and reattaches it (opens) to the new image file. The input image file may be in native CASA, FITS, or Miriad format. Look [here](#) for more information on foreign images. In the case of CASA images, both Float and Complex valued images are supported.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>infile</td>
<td>image file name</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```

""
#
print "\t----\t open Ex 1 \t----"
ia.maketestimage('anotherimage',overwrite=true) #first make 2nd image
ia.close()

ia.maketestimage('myimage',overwrite=true) #open image myimage
ia.open('anotherimage') # attach tool to 'anotherimage'
ia.close()
#
""
```
The \texttt{stff open} function first closes the old \texttt{imagefile}.
image.pad.html

**image.pad - Function**

1.1.1 Pad the perimeter of the direction plane with a number of pixels of specified value and mask.

**Description**

This method pads the directional plane of an image with a specified number of pixels on each side. The numerical and mask values of the padding pixels may also be specified. If a region is selected, a subimage of that region is created and then padded with the specified pixel parameters. Thus, padding an image of shape \((ra, dec, freq) = (512, 512, 10)\) specifying \(npixels = 3\) results in an image of size \((518, 518, 10)\), with the blc of the directional plane of the original pixel set corresponding to the directional pixel of \((3, 3)\) in the output. If `wantreturn` is True, an image analysis tool attached to the output image is returned. If False, none is returned.

**Arguments**
<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image name. If not specified, no persistent image is created.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>npixels</td>
<td>Number of pixels with which to pad each side of the direction plane.</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>value</td>
<td>Value given to the padding pixels.</td>
<td>double</td>
<td>0</td>
</tr>
<tr>
<td>padmask</td>
<td>Value of the mask for the padding pixels. True=&gt;good (unmasked), False=&gt;bad (masked).</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite the output if it exists? Default False</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>box</td>
<td>Rectangular region to select in direction plane. See &quot;help par.box&quot; for details. Default is to use the entire direction plane.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>chans</td>
<td>Channels to use. See &quot;help par.chans&quot; for details. Default is to use all channels.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>stokes</td>
<td>Stokes planes to use. See &quot;help par.stokes&quot; for details. Default is to use all stokes planes.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask. Default is none.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>stretch</td>
<td>Stretch the mask if necessary and possible? See help par.stretch. Default False</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>wantreturn</td>
<td>Return an image analysis tool attached to the created subimage?</td>
<td>bool</td>
<td>true</td>
</tr>
</tbody>
</table>
Returns
image

Example

```python
ia.fromshape("", [50, 50, 10])
# pad it with 5 pixels of value 2.5 all unmasked
padded = ia.pad(npixels=5, value=2.5, padmask=True)
ia.done()
# returns [60, 60, 10]
paddedshape = padded.shape()
padded.done()
```
image.crop.html

**image.crop - Function**

Crop masked pixels from the perimeter of an image.

**Description**

This method crops masked slices from the perimeter of an image. The axes parameter specifies which axes to consider. Axes not specified will not be cropped. An empty array implies that all axes should be considered. If wantreturn is True, an image analysis tool attached to the output image is returned. If False, none is returned.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image name. If not specified, no persistent image is created.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>axes</td>
<td>Axes to crop. Empty array means consider all axes.</td>
<td>intArray</td>
<td></td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite the output if it exists? Default False</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>box</td>
<td>Rectangular region to select in direction plane. See &quot;help par.box&quot; for details. Default is to use the entire direction plane.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>chans</td>
<td>Channels to use. See &quot;help par.chans&quot; for details. Default is to use all channels.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>stokes</td>
<td>Polarization selection. Default is all.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask. Default is none.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>stretch</td>
<td>Stretch the mask if necessary and possible? See help par.stretch. Default False</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>wantreturn</td>
<td>Return an image analysis tool attached to the created subimage?</td>
<td>bool</td>
<td>true</td>
</tr>
</tbody>
</table>

| Returns              | image                                                                         |
Example

```python
# myimage is of shape 20, 20, 20 with only the inner 16 x 14 x 12 pixels unmasked
ia.open("myimage")
# crop masked slices on all axes
cropped = ia.crop()
# returns [16, 14, 12]
cropped.shape()
cropped.done()
# crop only the masked slices at the edges of the image along axis 1
cropped2 = ia.crop(outfile="", axes=[1])
ia.done()
# returns [20, 14, 20]
cropped2.shape()
cropped2.done()
```
image.pixelvalue.html

**image.pixelvalue - Function**

1.1.1 Get value of image and mask at specified pixel coordinate

**Description**

This function gets the value of the image and the mask at the specified pixel coordinate. The values are returned in a record with fields 'value', 'mask' and 'pixel'. The value is returned as a quantity, the mask as a Bool (T is good). The 'pixel' field holds the actual pixel coordinate used.

If the specified pixel coordinate is off the image, "{}" is returned.

Excessive elements in pixel are silently discarded. Missing elements are given the (nearest integer) value of the reference pixel. This is reflected in the output record 'pixel' field.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Pixel coordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixel</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: intArray</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```
#
print "\t----\t pixelvalue Ex 1 \t----"
ia.maketestimage();
ia.pixelvalue()
#{'mask': True,
  # 'pixel': array([55, 37]),
  # 'value': {'unit': 'Jy/beam', 'value': 2.5064315795898438}}
```
print ia.pixelvalue([-1,-1])
# {}
print ia.pixelvalue([9])
#{'mask': True,
 # 'pixel': array([ 9, 37]),
 # 'value': {'unit': 'Jy/beam', 'value': 0.14012207090854645}}
print ia.pixelvalue([9,9])
#{'mask': True,
 # 'pixel': array([9, 9]),
 # 'value': {'unit': 'Jy/beam', 'value': -0.45252728462219238}}
ia.close()
#
image.putchunk.html

**image.putchunk - Function**

1.1.1 Put pixels from an array into a regular region of the image

**Description**

This function puts an array into the image file. If there is a default pixel mask it is ignored in this process. It is the complement of the getchunk function. You can specify the blc and inc if desired. If they are unspecified, they default to the beginning of the image and an increment of one. Any illegal blc values are set to zero. Any illegal inc values are set to unity. An error will result if you attempt to put an array beyond the extent of the image (i.e., it is not truncated or decimated).

If there are fewer axes in the array than in the image, the array is assumed to have trailing axes of length unity. Thus, if you have a 2D array and want to put it in as the YZ plane rather than the XY plane, you must ensure that the shape of the array is [1, nx, ny].

However, the argument replicate can be used to replicate the array throughout the image (from the blc to the trc). For example, if you provide a 2D array to a 3D image, you can replicate it through the third axis by setting replicate=T. The replication is done from the specified blc to the end of the image. Use function putregion if you want to terminate the replication at a trc value.

The argument locking controls two things. If True, then after the function is called, the image is unlocked (so some other process can acquire a lock) and it is indicated that the image has changed. The reason for having this argument is that the unlocking and updating processes are quite expensive. If you are repeatedly calling putchunk in a for loop, you would be advised to use this switch.

A related function is putregion which puts the pixels and masks into a more complex region-of-interest. Function putchunk is retained because it is faster and therefore preferable for repeated operation in loops if the pixel mask is not required.

See also the functions set and calc which can also change pixel values.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixels</td>
<td>Numeric array. Required input.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>blc</td>
<td>Bottom-Left-Corner (start) of location in image. Default is start of image.</td>
<td>intArray</td>
<td>-1</td>
</tr>
<tr>
<td>inc</td>
<td>Increment (stride) along axes</td>
<td>intArray</td>
<td>1</td>
</tr>
<tr>
<td>list</td>
<td>List bounding box to logger?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>locking</td>
<td>Unlock image after use?</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>replicate</td>
<td>Replicate array through image</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

| Returns           | bool                                                                        |

**Example**

We can clip all pixels to be \( \leq 5 \) as follows.

```python
# We can clip all pixels to be \( \leq 5 \)
print "\t----	 putchunk Ex 1	----"
ia.fromshape(shape=[10,10]) # create an example image
pix = ia.getchunk() # get pixels to modify from example image
for i in range(len(pix)):
    pix[i] = list(pix[i]) # convert tuple to list so it can be modified
    for j in range(len(pix[i])):
        pix[i][j] = i*10 + j
    pix[i] = tuple(pix[i]) # convert list back to tuple
ia.putchunk(pix) # put pixels back into example image
print pix # pixels have values 0-99
```
pix2 = ia.getchunk()  # get all pixels into an array (again)
for i in range(len(pix2)):
    pix2[i] = list(pix2[i])  # convert tuple to list so it can be modified
for j in range(len(pix2[i])):
    if pix2[i][j] > 5:
        pix2[i][j] = 5  # clip values to 5
        pix2[i] = tuple(pix2[i])  # convert list back to tuple
ia.putchunk(pix2)  # put array back into image
print ia.getchunk()
ia.close()
#
""

The above example shows how you could clip an image to a value. If all the pixels didn’t easily fit in memory, you would iterate through the image chunk by chunk to avoid exhausting virtual memory. Better would be to do this via LEL through function calc.

Suppose we wanted to set the fifth XY plane to 1.

We could do so as follows:

""
#
print "\t---\t putchunk Ex 2 \t---"
ia.fromshape(shape=[10,10,10])
imshape = ia.shape()
pix = ia.makearray(1, [imshape[0],imshape[1]])
ia.putchunk(pix, blc=[0,0,4])
print ia.getchunk()[0:3]
ia.close()
#
""

Suppose we wanted to set the first YZ plane to 2.

""
#
print "\t---\t putchunk Ex 3 \t---"
ia.fromshape(shape=[10,10,10])
imshape = ia.shape()
pix = ia.makearray(2, [1,imshape[1],imshape[2]])
ia.putchunk(pix)
print ia.getchunk()[0:3]
ia.close()
#
"""
image.putregion.html

image.putregion - Function

1.1.1 Put pixels and mask into a region-of-interest of the image

Description

This function replaces data and/or pixel mask values in the image in the specified region-of-interest. The pixels and/or pixelmask arrays must be the shape of the bounding box, and the whole bounding box is replaced in the image. The region-of-interest is really only used to specify the bounding box. If the region extends beyond the image, it is truncated. If the pixels or pixelmask array shapes do not match the bounding box, an error will result.

When you put a pixel mask, it either replaces the current default pixel mask, or is created. The pixel mask is put before the pixels.

The argument usemask is only relevant when you are putting pixel values and there is a pixel mask (meaning also the one you might have just put in place). If usemask=T then only pixels for which the mask is good (T) are altered. If usemask=F then all the pixels in the region are altered - the mask is ignored.

The argument replicate can be used to replicate the array throughout the image (from the blc to the trc). For example, if you provide a 2D array to a 3D image, you can replicate it through the third axis by setting replicate=T.

The replication is done in the specified region.

The argument locking controls two things. If True, then after the function is called, the image is unlocked (so some other process can acquire a lock) and it is indicated that the image has changed. The reason for having this argument is that the unlocking and updating processes are quite expensive. If you are repeatedly calling putregion in a for loop, you would be advised to use this switch (and to consider using putchunk).

See the related functions putchunk, set and calc.

Arguments
Inputs

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixels</td>
<td>The pixel values. Default is none.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>pixelmask</td>
<td>The pixel mask values. Default is none.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
<td>any</td>
<td>any</td>
</tr>
<tr>
<td>list</td>
<td>List the bounding box and any mask creation to the logger</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>usemask</td>
<td>Honour the mask when putting pixels</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>locking</td>
<td>Unlock image after use?</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>replicate</td>
<td>Replicate array through image</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

Returns

bool

Example

Suppose that we have a 2-dimensional image. First we recover the pixel and \pixelmask\ values from a polygonal region. Then we change the values in the array that are within the region to zero and replace the data.
print "\t----\t putregion Ex 1 \t----"

ia.maketestimage() # Attach an image to image tool

x = ['3pix', '6pix', '9pix', '6pix', '5pix', '5pix', '3pix'] # X vector abs pixels
y = ['3pix', '4pix', '7pix', '9pix', '7pix', '5pix', '3pix'] # Y vector abs pixels

mycs = ia.coordsys()

r1 = rg.wpolygon(x,y,csys=mycs.torecord()) # Create polygonal world region

mycs.done()

pixels = ia.getregion(r1) # Recover pixels

pixelmask = ia.getregion(r1, getmask=T) # and mask

for i in range(len(pixels)):
    pixels[i] = list(pixels[i]) # convert tuple to list for mods

for j in range(len(pixels[i])):
    if pixelmask[i][j]:
        pixels[i][j] = 0 # Set pixels where mask is T to zero

pixels[i] = tuple(pixels[i]) # convert list back to tuple

ia.putregion(pixels=pixels, pixelmask=pixelmask, region=r1) # Replace pixels only

ia.close()

#
image.rebin.html

**image.rebin - Function**

1.1.1 Rebin an image by the specified integer factors

**Description**

This application rebins the current image by the specified integer binning factors for each axis. It supports both float valued and complex valued images. The corresponding output pixel value is the average of the input pixel values. The output pixel will be masked bad if there were no good input pixels. A polarization axis cannot be rebinned.

The binning factors array must contain at least one element and no more elements than the number of input image axes. If the number of elements specified is less than the number of image axes, then the remaining axes not specified are not rebinned. All specified values must be positive. A value of one indicates that no rebinning of the associated axis will occur.

Binning starts from the origin pixel of the bounding box of the selected region or the origin pixel of the input image if no region is specified. The value of crop is used to determine how to handle cases where there are pixels at the end of the axis that do not form a complete bin. If crop=True, extra pixels at the end of the axis are discarded. If crop=False, the remaining pixels are averaged into the final bin along that axis. Should the length of the axis to be rebinned be an integral multiple of the associated binning factor, the value of crop is irrelevant.

A value of dropdeg=True will result in the output image not containing axes that are degenerate in the specified region or in the input image if no region is specified. Note that, however, the binning factors array must still account for degenerate axes, and the binning factor associated with a degenerate axis must always be 1.

If outfile is given, the image is written to the specified disk file. If outfile is unset, the Image tool is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you destroy the on-the-fly Image tool returned by this function (with the done function) this temporary image is deleted.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>bin</td>
<td>Binning factors for each axis</td>
<td>intArray</td>
<td></td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask. Default is none.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>dropdeg</td>
<td>Drop degenerate axes</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>stretch</td>
<td>Stretch the mask if necessary and possible? See help par.stretch. Default False</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>crop</td>
<td>Remove pixels from the end of an axis to be rebinned if there are not enough to form an integral bin?</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

*image*

**Example**

```
```

236
print "\t----\t rebin Ex 1 \t----"
ia.maketestimage();
im2 = ia.rebin(bin=[2,3]);
im2.done()
ia.close()
#
"""
**image.regrid - Function**

1.1.1 regrid this image to the specified Coordinate System

**Description**

This function regrids the current image onto a grid specified by the given Coordinate System. You can also specify the shape of the output image. The Coordinate System must be given via a Coordsys tool (using coodsys.torecord()). It is optional; if not specified, the Coordinate System from the input image (i.e. the one to which you are applying the regrid function) is taken. The order of the coordinates and axes in the output image is always the same as the input image. It simply 'finds' the relevant coordinate in the supplied Coordinate System in order to figure out the regridding parameters. The supplied Coordinate System must have at least as many coordinates as are required to accommodate the axes you are regridding (e.g. if you regrid the first two axes, and these belong to a Direction Coordinate, you need one Direction Coordinate in the supplied Coordinate System). Coordinates pertaining to axes that are not being regridded are supplied from the input image, not the given Coordinate System.

Reference changes are handled (e.g. J2000 to B1950, LSR to TOPO). In general, the conversion machinery attempts to work out how sophisticated it needs to be (e.g. am I regridding LSR to LSR or LSR to TOPO). However, it errs on the side of conservatism so that it can be that the conversion machine requires more information than it actually needs. For full frame conversions, one needs to know things like location on earth (e.g. observatory), direction of observation, and time of observation.

If you get the above errors and you are doing a frame conversion, then that means you must insert some extra information into the Coordinate System of your image. Most likely it’s the time (coordsys.setepoch) and location (coordsys.settelescope) that are missing. If you get these errors and you know that you are not specifying a frame change (e.g. regrid LSR to LSR) then try setting doref=F. This will (silently) bypass all possible frame conversions.

Note that if you are requesting a frame conversion and you set doref=F you are doing a bad thing (and you will get no warnings).

If you regrid a plane holding a Direction Coordinate and the units are Jy/pixel then the output is scaled to conserve flux (roughly; just one scale factor at the reference pixel is computed).

Regridding of complex-valued images is supported. The real and imaginary parts are regridded independently and the resulting regridded pixel values are combined to form the regridded, complex-valued image.
A variety of interpolation schemes are provided (you need only specify the first three characters to \textit{method}). The cubic interpolation is substantially slower than linear, and often the improvement is modest. By default you get linear interpolation.

You specify the shape of the output image (\textit{shape}) and which output axes you want to regrid (\textit{axes}). Note that a Stokes axis cannot be regridded (you will get a warning if you try).

The \textit{axes} argument cannot be used to discard axes from the output image; it can only be used to specify which \textit{output} axes are going to be regridded and which are not. Any axis that you are not regridding must have the same output shape as the input image shape for that axis.

The \textit{axes} argument can also be used to specify the order in which the \textit{output} axes are regridded. This may give you significant performance benefits. For example, imagine we are going to regrid a spectral-line cube of shape $[512,512,1204]$ to shape $[256,256,32]$. If you specified \texttt{axes=[0,1,2]} then first, the Direction axes would be regridded for each of the 1024 pixels (and stored in a temporary image). Then each profile at each spatial location in the temporary image would be regridded to 32 pixels. You could speed this process up significantly by setting \texttt{axes=[2,0,1]}. In this case, first each profile would be regridded to 32 pixels, and then each plane of the 32 pixels would be regridded. Note that the order of \textit{axes} does not affect the order of the \textit{shape} argument. I.e. it should be given in the natural pixel axis order of the image $[256,256,32]$ in both cases.

You can also specify a \textit{region-of-interest} to be applied to the input image. If you do this, you need to be careful with the output shape for non-regridded axes (must match that of the region - use function boundingbox to find that out).

If \texttt{outfile} is given, the image is written to the specified disk file. If \texttt{outfile} is unset, the on-the-fly Image \texttt{tool} returned by this function is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you destroy the on-the-fly Image \texttt{tool} (with the \textit{done} function) this temporary image is deleted.

The argument \texttt{replicate} can be used to simply replicate pixels rather than regridding them. Normally (\texttt{replicate=F}), for every output pixel, its world coordinate is computed and the corresponding input pixel found (then a little interpolation grid is generated). If you set \texttt{replicate=T}, then what happens is that for every output axis, a vector of regularly sampled input pixels is generated (based on the ratio of the output and input axis shapes). So this just means the pixels get replicated (by whatever interpolation scheme you use) rather than regridded in world coordinate space. This process is much faster, but its not a true world coordinate based regrid.

As described above, when \texttt{replicate} is False, a coordinate is computed for each output pixel; this is an expensive operation. The argument \texttt{decimate} allows you to decimate the computation of that coordinate grid to a sparse grid, which is then filled in via fast interpolation. The default for \texttt{decimate} is 10. The number of pixels per axis in the sparse grid is the number of output
pixels for that axis divided by the decimation factor. A factor of 10 does pretty well. You may find that for very non-linear coordinate systems (e.g. very close to the pole) that you have to reduce the decimation factor. You may also have to reduce the decimation factor if the number of pixels in the output image along an axis to be regridded is less than about 50, or the output image may be completely masked.

If one of the axes to be regridded is a spectral axis and asvelocity=T, the axis will be regridded to match the velocity, not the frequency, description of the template coordinate system. Thus the output pixel values will correspond only to the velocity, not the frequency, of the output axis.

Sometimes it is useful to drop axes of length one (degenerate axes). Use the dropdeg argument if you want to do this. It will discard the axes from the input image. Therefore the output shape and Coordinate System that you supply must be consistent with the input image after the degenerate axes are dropped.

Argument force can be used to force all specified axes to be regridded, even if the algorithm determines that they don’t need to be (because the input and output coordinate information is identical).

There is a useful function setreferencelocation that you can use to keep a specific world coordinate in the center of an image when regridding (see example below).

The output pixel mask will be good (T) unless the regridding failed to find a value for that output pixel in which case it will be bad (F). For example, if the total input mask (default input pixel mask plus OTF mask) for all of the relevant input pixels were masked bad then the output pixel would be masked bad (F).

**Multiple axis Coordinates limitation** – Some cooordinates pertain to more than one axis. E.g. a Direction Coordinate holds longitude and latitude. A Linear Coordinate can also hold many axes. When you regrid *any* axis from a Coordinate which holds multiple axes, you must fully specify the coordinate information for all axes in that Coordinate in the Coordinate System that you provide. For example, you have a Linear Coordinate with two axes and you want to regrid axis one only. In the Coordinate System you provide, the coordinate information for axis two (not being regridded) must correctly be a copy from the input coordinate system (it won’t be filled in for you).

If an image has per-plane beams and one attempts to regrid the spectral axis, an exception is thrown.

**IMPORTANT NOTE ABOUT FLUX CONSERVATION** in general regridding is inaccurate for images that the angular resolution is poorly sampled. A check is done for such cases and a warning message is emitted if a beam present. However, no such check is done if there is no beam present. To add a restoring beam to an image, use ia.setrestoringbeam().

**Arguments**

240
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>shape</td>
<td>Shape of output image. Default is input shape.</td>
<td>intArray</td>
<td>-1</td>
</tr>
<tr>
<td>csys</td>
<td>Coordinate System for output image. Default is input image coordinate system.</td>
<td>record</td>
<td></td>
</tr>
<tr>
<td>axes</td>
<td>The output pixel axes to regrid. Default is all.</td>
<td>intArray</td>
<td>-1</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask. Default is none.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>method</td>
<td>The interpolation method. String from 'nearest', 'linear', 'cubic'.</td>
<td>string</td>
<td>linear</td>
</tr>
<tr>
<td>decimate</td>
<td>Decimation factor for coordinate grid computation</td>
<td>int</td>
<td>10</td>
</tr>
<tr>
<td>replicate</td>
<td>Replicate image rather than regrid?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>doref</td>
<td>Turn on reference frame changes</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>dropdeg</td>
<td>Drop degenerate axes</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>force</td>
<td>Force specified axes to be regridded</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>asvelocity</td>
<td>Regrid spectral axis in velocity space rather than frequency space?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>stretch</td>
<td>Stretch the mask if necessary and possible? See help par.stretch. Default False</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>
Example

"""
#
print "\t----\t regrid Ex 1 \t----"
ia.maketestimage('radio.image', overwrite=true)
ia.maketestimage('optical.image', overwrite=true)
mycs = ia.coordsys(); # get optical image co-ordinate system
ia.open('radio.image')
imrr = ia.regrid(outfile='radio.regridded', csys=mycs.torecord(),
shape=ia.shape(), overwrite=true)
#viewer()
mycs.done()
imrr.done()
ia.close()
#
"""

In this example, we regrid a radio image onto the grid of an optical image - this probably (if the optical FITS image was correctly labelled !!) will involve a projection change (optical images are usually TAN projection, radio usually SIN).

Example

"""
#
print "\t----\t regrid Ex 2 \t----"
ia.maketestimage('radio.image', overwrite=true)
mycs = ia.coordsys();
print mycs.referencecode('dir')
#J2000

243
mycs.setreferencecode(value='B1950', type='dir', adjust=T)
im3 = ia.regrid(outfile='radio.regridded', csys=mycs.torecord(),
               shape=ia.shape(), overwrite=true)

mycs.done()
im3.done()
ia.close()
#
# ""

In this example, we regrid a radio image from J2000 to B1950. This is accomplished by first recovering the Coordinate System into a Coordsys tool, manipulating the reference code with that tool, and then supplying the new Coordinate System to the regrid function.

Example

""
#
print "\t----\t regrid Ex 3 \t----"
ia.maketestimage('zz', overwrite=true)
mycs = ia.coordsys();
p = ia.shape()
for i in range(len(p)):
    p[i] = p[i]/2.0 + 10
refval = ia.toworld(value=p, format='n') # Location of interest
inc = mycs.increment()
incx = inc['numeric']
for i in range(len(incx)):
    incx[i] = incx[i]/2.0 # Halve increment
mycs.setincrement(value=inc)
shp = ia.shape()
refpix = refval['numeric'][:]
refpix = list(refpix) # numpy makes this necessary
for i in range(len(shp)):
    shp[i] = shp[i] *2 # Double shape
    refpix[i] = int((shp[i]-1)/2.0 + 1); # New ref pix
# Center image on location of interest
mycs.setreferencelocation(pixel=refpix, world=refval)

244
imr = ia.regrid(cs=mycs.torecord(), shape=shp, overwrite=true)# Regrid
mycs.done()
imr.done()
ia.close()
#
"""
image.transpose - Function

1.1.1 Transpose the image.

Description

This method transposes the axes in the input image to the specified order. The associated pixel and mask values and coordinate system are transposed. If the outfile parameter is empty, only a temporary image is created; no output image is written to disk.

The order parameter describes the mapping of the input axes to the output axes. It can be one of three types: a non-negative integer, a string, or a list of strings. If a string or non-negative integer, it should contain zero-based digits describing the new order of the input axes. It must contain the same number of (unique) digits as the number of input axes. For example, specifying reorder="1032" or reorder=1032 for a four axes image maps input axes 1, 0, 3, 2 to output axes 0, 1, 2, 3. In the case of order being a nonnegative integer and the zeroth axis in the input being mapped to zeroth axis in the output, the zeroth digit is implicitly understood to be 0 so that to transpose an image where one would use a string order="0321", one could equivalently specify an int order=321. IMPORTANT: When specifying a non-negative integer and mapping the zeroth axis of the input to the zeroth axis of the output, do *not* explicitly specify the leading 0; eg, specify order=321 rather than order=0321. Python interprets an integer with a leading 0 as an octal number. Because of ambiguity for axes numbers greater than nine, using string or integer order specifications cannot handle images containing more than 10 axes. The order parameter can also be specified as a list of strings which uniquely minimally match, ignoring case, the image axis names (ia.coordsys().names()). So to reorder an image with right ascension, declination, and frequency axes, one could specify order=["d", "f", "r"] or equivalently ["decl", "freq", "right a"]. Note that specifying "ra" for the right ascension axis will result in an error because "ra" does not match the first two characters of right ascension. Axes can be simultaneously inverted in cases where order is a string or an array of strings by specifying negative signs in front of the axis/axes to be inverted. So, in a 4-D image, order="-10-3-2" maps input axes 1, 0, 3, 2 to output axes 0, 1, 2, 3 and reverses the direction and values of input axes 1, 3, and 2.

Arguments

246
Inputs

<table>
<thead>
<tr>
<th>outfile</th>
<th>Output image file name. Default is unset.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>order</th>
<th>Zero-based order of axes in output image (e.g. &quot;120&quot; =&gt; input: 0-&gt;2, 1-&gt;0, 2-&gt;1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
</tbody>
</table>

Returns

image

Example

```python
""
# swap stokes (axis 2) and spectral (axis 3) axes in a 4 dimensional image
ia.open("myimage.fits")
reordim = ia.transpose(outfile="my_reordered_image.im", order="0132")
ia.done()
""
```
image.rotate.html

**image.rotate - Function**

1.1.1 rotate the direction coordinate axes attached to the image and regrid the image to the rotated Coordinate System

**Description**

This function rotates two axes of an image. These axes are either those associated with a Direction coordinate or with a Linear coordinate. The Direction coordinate takes precedence. If rotating a Linear coordinate, it must hold precisely two axes. The method is that the Coordinate is rotated and then the input image is regridded to the rotated Coordinate System.

If the image brightness units are Jy/pixel then the output is scaled to conserve flux (roughly; just one scale factor at the reference pixel is computed). A variety of interpolation schemes are provided (you need only specify the first three characters to method). The cubic interpolation is substantially slower than linear. By default you get cubic interpolation.

You can specify the shape of the output image (shape). However, all axis that are not regridded retain the same output shape as the input image shape for that axis. Only the direction coordinate axes are regridded.

You can also specify a region-of-interest to be applied to the input image. If you do this, you need to be careful with the output shape for non-regridded axes (must match that of the region - use function boundingbox to find that out).

If outfile is given, the image is written to the specified disk file. If outfile is unset, the on-the-fly Image tool returned by this function is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you destroy the on-the-fly Image tool (with the done function) this temporary image is deleted.

The argument replicate can be used to simply replicate pixels rather than regridding them. Normally (replicate=F), for every output pixel, its world coordinate is computed and the corresponding input pixel found (then a little interpolation grid is generated). If you set replicate=T, then what happens is that for every output axis, a vector of regularly sampled input pixels is generated (based on the ratio of the output and input axis shapes). So this just means the pixels get replicated (by whatever interpolation scheme you use) rather than regridded in world coordinate space. This process is much faster, but its not a true world coordinate based regrid.

As described above, when replicate is False, a coordinate is computed for each output pixel; this is an expensive operation. The argument decimate allows
you to decimate the computation of that coordinate grid to a sparse grid, which is then filled in via fast interpolation. The default for `decimate` is 0 (no decimation). The number of pixels per axis in the sparse grid is the number of output pixels for that axis divided by the decimation factor. A factor of 10 does pretty well. You may find that for very non-linear coordinate systems (e.g., very close to the pole) that you have to reduce the decimation factor. The output pixel mask will be good (T) unless the regridding failed to find a value for that output pixel in which case it will be bad (F). For example, if the total input mask (default input pixel mask plus OTF mask) for all of the relevant input pixels were masked bad then the output pixel would be masked bad (F).

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>shape</td>
<td>Shape of output image. Default is shape of input image.</td>
<td>intArray</td>
<td>-1</td>
</tr>
<tr>
<td>pa</td>
<td>Angle by which to rotate. Default is no rotation.</td>
<td>any</td>
<td>0deg</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask. Default is none.</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td>method</td>
<td>The interpolation method. String from 'nearest', 'linear', or 'cubic'.</td>
<td>string</td>
<td>cubic</td>
</tr>
<tr>
<td>decimate</td>
<td>Decimation factor for coordinate grid computation</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>replicate</td>
<td>Replicate image rather than regrid?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>dropdeg</td>
<td>Drop degenerate axes</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>stretch</td>
<td>Stretch the mask if necessary and possible? See help par.stretch. Default False</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td></td>
</tr>
</tbody>
</table>
Example

```python
ia.maketestimage()
imr=ia.rotate(outfile="rotated.im", pa='45deg')
imr.done()
ia.close()
```

In this example, we rotate the direction coordinate axes (RA/Dec) of a test image by 45 degrees and regrid the image onto the axes.
**image.rotatebeam.html**

**image.rotatebeam - Function**

1.1.1 rotate the image's beam(s) counterclockwise through the specified angle.

**Description**

This method rotates the attached image's beam(s) counterclockwise through the specified angle. This is the same thing as increasing the position angle(s) of the beam(s) by the specified angle. If the image does not have a beam, no changes to the image are made. If the image has multiple beams, all the beams are rotated through the same angle.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Angle by which to rotate image’s beam(s). Default is no rotation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>angle</td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant 0deg</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
# rotate any and all beams in the image (increase their position angles) by 30 degrees.
ia.open("my.im")
ia.rotatebeam("30deg")
ia.done()
```
image.rename.html

**image.rename - Function**

[1.1.1] Rename the image file associated with this image tool

**Description**

This function renames the image file associated with the image tool. If a file with name name already exists, you can overwrite it with the argument overwrite; otherwise a fail will result.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The new image file name</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite target file if it already exists</td>
</tr>
</tbody>
</table>

**Allowed**

- name: string
- overwrite: bool

**Default**

- name: string
- overwrite: bool

**Returns**

bool

**Example**

```
# print "\t----\t rename Ex 1 \t----"
ia.maketestimage('myimage',overwrite=T)
print ia.name(strippath=T)
#myimage
ia.rename('newimage', overwrite=T)
print ia.name(strippath=T)
#newimage
# 253
```
image.replacemaskedpixels.html

**image.replacemaskedpixels - Function**

[1.1.1] replace the values of pixels which are masked bad

**Description**

This application replaces the values of all pixels whose total input mask (default input `pixel mask` and OTF mask) is bad (F) with the specified value. It supports both float valued and complex valued images.

If the argument `update` is F (the default), the actual `pixel mask` is left unchanged. That is, masked pixels remain masked. However, if you set `update=T` then the `pixel mask` will be updated so that the `pixel mask` will now be T (good) where the total input mask was F (bad).

See maskhandler for information on how to set the default `pixel mask`.

There are a few ways in which you can specify what to replace the masked pixel values by.

- First, you can give the `pixels` argument a simple numeric scalar (e.g. `pixels=1.0`). Then, all masked values will be replaced by that value.

- Second, you can give a scalar `LEL` expression string (e.g. `pixels='min(myimage)'`). Then, all masked values will be replaced by the scalar that results from the expression. If the scalar expression is illegal (e.g. in the expression `pixels='min(myimage)'` there were no good pixels in `myimage`) then the value 0 is used for replacement.

- Third, you can give a `LEL` expression string which has the same shape as the `image file` you are applying the function to. For example, putting `pixels='myotherimage'` means replace all masked pixels in this `image file` with the equivalent pixel in the `image file` called `myotherimage`.

Your expression might be quite complex, and you can think of it as producing another masked lattice. However, in the replace process, the mask of that expression lattice is ignored. Thus, only the mask of the `image file` you are replacing and the pixel values of the expression lattice are relevant.

The expression must conform with the subimage formed by applying the `region-of-interest` to the image (i.e. that associated with this Image tool). If you use the `mask` argument as well, the `region-of-interest` is applied to it as well (see examples).

**Arguments**

255
### Inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixels</td>
<td>The new value(s), Numeric scalar or LEL expression</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask. Default is none.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>update</td>
<td>Update mask as well?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>list</td>
<td>List the bounding box to the logger</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>stretch</td>
<td>Stretch the mask if necessary and possible? See help par.stretch. Default False</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

### Returns

bool

### Example

```


t---- replacemaskedpixels Ex 1 t----
ia.maketestimage('zz1', overwrite=true)
ia.calcmask('zz1<0')
ia.replacemaskedpixels(0.0)
ia.replacemaskedpixels('min(zz1)')
ia.close()
t----
```

* * *
These examples replace all masked pixels by the specified scalar. In the second case, the scalar comes from a LEL expression operating on \{sff zz1\} (or it could be from an LEL expression operating on some other image).

Example

```python
print "\t----\t replacemaskedpixels Ex 2 \t----"
ia.maketestimage('zz2',overwrite=true)
ia.close()
ia.maketestimage('zz1',overwrite=true)
#ia.calcmask('zz1<0')
ia.replacemaskedpixels(0.0, mask='zz2>0')
ia.close()
#
```

Let us say that \{sff zz1\} has no mask. By using the \{stfaf mask\} argument, we generate a transient mask which is T (good) when the pixel values are positive. This means that all non-positive values (when that mask is F [bad]) will be replaced with the value 0. If \{sff zz1\} did have a mask it would be applied as well as the transient mask (the masks would be logically ORed).

Example

```python
print "\t----\t replacemaskedpixels Ex 3 \t----"
```

257
The specified region takes one quarter of the image by area centered on the image center. The region is applied to the \{sf af mask\} expression as well - this means that any images in the \{sf af mask\} expression must conform with the \{sf f zz1\} image. The replacement of the scalar is then done only within that region. Note that in the \{sf af mask\} expression we have specified the image with the Image tool \{sf im2\} via im2.name() (rather than referring to its disk file name \{sf f zz2\}).

Example

In this example, the replacement values are taken from a LEL expression adding two other images together. The expression must conform with the
image {sff zz1}.
image.beamarea.html

**image.beamarea** - Function

Get the beam area.

**Description**

Get the area of the image’s restoring beam.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channel</td>
<td>The zero-based spectral channel number for a per-plane beam. Default -1</td>
</tr>
<tr>
<td>polarization</td>
<td>The zero-based polarization plane number for a per-plane beam. Default -1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Returns**

record
Get the restoring beam(s).

Description

This function gets the restoring beam(s), if any. If the image has a traditional restoring beam, that is returned no matter what channel and polarization are set to. If the image has per-plane beams and at least one of channel or polarization is set to a non-negative value, the beam for that particular plane is returned. In both these cases, the returned record contains fields 'major', 'minor' and 'positionangle'. Each of these fields contains a quantity.

If the image contains multiple beams and both channel and polarization are negative, a record containing all the beams is returned. This record contains three fields. "nChannels" contains an integer value equal to the number of channels, "nStokes" contains an integer value equal to the number of polarizations, and "beams" contains a record of information for all beams. If the image has no polarization axis or no spectral axis, the fields in the "beams" record run from "*0" to the number of spectral channels or number of polarizations less one, eg "*31" for an image with 32 channels. Each of these fields references a beam subrecord with the structure described above for a single beam.

If the image contains both a spectral axis and a polarization axis, the record returned contains fields running from "*0" to the number of spectral channels less one, eg "*31" for an image with 32 spectral channels. Each of these fields has an associated subrecord with fields running from "*0" to the number of polarizations less one, eg "*3" in an image with 4 polarizations. Each one of those fields is associated with a beam record for that corresponding channel and polarization. The beam record has a structure described above for a single beam.

If there is no restoring beam, this function returns an empty record.

You can set the restoring beam with function setrestoringbeam.

Arguments
Inputs

channel The zero-based spectral channel number for a per-plane beam. Default -1
allowed: int
Default: -1

polarization The zero-based polarization plane number for a per-plane beam. Default -1
allowed: int
Default: -1

Returns

record

Example

```python
# print "\t----\t restoringbeam Ex 1 \t----"
ia.maketestimage()
print ia.restoringbeam()
#{'major': {'unit': 'arcsec', 'value': 53.500004857778549},
  # 'minor': {'unit': 'arcsec', 'value': 34.199998900294304},
  # 'positionangle': {'unit': 'deg', 'value': 6.0}}
ia.close()
#
```
Description

This function does Fourier-based convolution of an image file by a specified separable kernel. If outfile is given, the image is written to the specified disk file. If outfile is unset, the on-the-fly Image tool returned by this function is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you destroy the Image tool (with the done function) this temporary image is deleted.

You specify which axes of the image you wish to convolve, by what kernel of what width. The kernel types can be shortened to ‘gauss’, ‘hann’ and ‘box’. You specify the widths of the convolution kernels via the argument widths. The values can be specified as a vector of three different types.

- Quantity - for example widths=qa.quantity("1arcsec 0.00001rad"). Note that you can use pixel units, viz. widths=qa.quantity("10pix 0.00001rad") see below.

- String - for example widths="1km 2arcsec" (i.e. a string that qa.quantity() accepts).

- Numeric - for example widths=[10,20]. In this case, the units of the widths are assumed to be in pixels.

The interpretation of widths depends upon the kernel type.

- Gaussian - the specified width is the full-width at half-maximum.

- Boxcar (tophat) - the specified width is the full width.

- Hanning - The kernel is $z[i] = 0.25*y[i-1] + 0.5*y[i] + 0.25*y[i+1]$. The width is always 3 pixels, regardless of what you give (but you still have to give it !).

The scaling of the output image is determined by the argument scale. If you leave it unset, then autoscaling will be invoked which means that the convolution kernels will all be normalized to have unit volume to as to conserve flux.
If you do not leave \texttt{scale} unset, then the convolution kernel will be scaled by this value (it has peak unity before application of this scale factor). Masked pixels will be assigned the value 0.0 before convolution. The output mask is the combination (logical OR) of the default input \texttt{pixel mask} (if any) and the OTF mask. Any other input \texttt{pixel masks} will not be copied. Use function \texttt{maskhandler} if you need to copy other masks too. See also the other convolution functions \texttt{convolve2d}, \texttt{convolve} and \texttt{hanning}.

\textbf{Arguments}
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>axes</td>
<td>Axes to convolve. Default is [0,1,...].</td>
</tr>
<tr>
<td>allowed</td>
<td>intArray</td>
</tr>
<tr>
<td>Default</td>
<td>-1</td>
</tr>
<tr>
<td>types</td>
<td>Type of convolution kernel. Vector of strings from 'boxcar', 'gaussian', and 'hanning'. Default is appropriately sized vector of 'gaussian'.</td>
</tr>
<tr>
<td>allowed</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>widths</td>
<td>Convolution kernel widths, Vector of numeric, quantity or string</td>
</tr>
<tr>
<td>allowed</td>
<td>any</td>
</tr>
<tr>
<td>Default</td>
<td>variant</td>
</tr>
<tr>
<td>scale</td>
<td>Scale factor. Default is autoscale.</td>
</tr>
<tr>
<td>allowed</td>
<td>double</td>
</tr>
<tr>
<td>Default</td>
<td>-1</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
</tr>
<tr>
<td>allowed</td>
<td>any</td>
</tr>
<tr>
<td>Default</td>
<td>region</td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask. Default is none.</td>
</tr>
<tr>
<td>allowed</td>
<td>any</td>
</tr>
<tr>
<td>Default</td>
<td>variant</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file?</td>
</tr>
<tr>
<td>allowed</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>stretch</td>
<td>Stretch the mask if necessary and possible? See help par.stretch. Default False</td>
</tr>
<tr>
<td>allowed</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

image

**Example**

265
print "\t----\t sepconvolve Ex 1 \t----"
ia.maketestimage('xyv', overwrite=true)
im2 = ia.sepconvolve(outfile='xyv.con', axes=[0,1], types=['gauss','box'], widths=[10,20], overwrite=true)
im2.done()
ia.close()
#
###
image.set - Function

[1.1.1] Set pixel and/or mask values with a scalar in a region-of-interest of the image

Description

This function replaces data and/or mask values within the image in the specified region-of-interest. You can think of it as a simplified version of the image calculator. Unlike the calc function, you can only set a scalar value for all pixels in the specified region-of-interest. For example, it can be useful to set a whole image to one value, or a mask in a region-of-interest to one value.

Although you could do that with the related functions putregion and putchunk, you would have to make an array of the shape of the image and if that is large, it could be resource expensive.

The value for the pixels is specified with the pixels argument. It can be given as either a Lattice Expression Language (or LEL) expression string or a simple numeric scalar. See note 223 for a detailed description of the LEL expression syntax. If you give a LEL expression it must be a scalar expression.

Note that any default mask is ignored by this function when you set pixel values. This is different from calc where the extant mask is honoured.

The value for the pixel mask is specified with the pixelmask argument (T, F, unset). If it’s unset then the mask is not changed.

If you specify pixelmask= T or F, then the mask that is affected is the current default mask (see maskhandler). If there is no mask, a mask is created for you and made the default mask.

Arguments
Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Default</th>
<th>Allowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixels</td>
<td>The pixel value, LEL scalar expression or numeric scalar. Default is unset.</td>
<td></td>
<td>variant</td>
</tr>
<tr>
<td>pixelmask</td>
<td>The pixel mask value. Either 0 or 1 if set. Default is unset.</td>
<td>-1</td>
<td>int</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
<td></td>
<td>any</td>
</tr>
<tr>
<td>list</td>
<td>List the bounding box and any mask creation to the logger</td>
<td>false</td>
<td>bool</td>
</tr>
</tbody>
</table>

Returns

bool

Example

```python
# "\t----\t set Ex 1 \t----"
ia.maketestimage('yy', overwrite=true)
ia.fromshape('xx', [10,20], overwrite=true)
r1 = rg.box([2,2],[6,8]) # Make a box region
ia.set(pixels=1.0) # Set all pixels to 1
ia.set(pixels='2.0', region=r1) # Set all pixels to 2 in the region
ia.set(pixels='min(yy)') # Set all pixels to minimum of image yy
ia.set(pixels='min('+ia.name(strippath=T)+')', region=r1) # Set pixels in region to minimum of image xx
ia.set(pixelmask=T) # Set mask to all T
ia.set(pixels=0, pixelmask=F, region=r1) # Set pixels and mask in region
ia.close()
# "\t----\t set Ex 1 \t----"
```
image.setbrightnessunit.html

**image.setbrightnessunit - Function**

1.1.1 Set the image brightness unit

**Description**

This function sets the image brightness unit. Both float and complex valued images are supported. You can get the brightness unit with function brightnessunit.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>New brightness unit</td>
</tr>
</tbody>
</table>

- allowed: string
- Default:

**Returns**

bool

**Example**

```python
""
#
print "\t----\t setbrightnessunit Ex 1 \t----"
ia.fromshape(shape=[10,10])
ia.setbrightnessunit('km')
print ia.brightnessunit()
#km
#
""
```

---

270
image.setcoordsys - Function

1.1.1 Set new Coordinate System

Description

This function replaces the coordinate system in the image. It is supported for both float and complex valued images. Coordinate systems are manipulated with the cs tool. The coordinate system can be recovered from an image via the coordsys function.

Note that changing the cs tool has no effect on the original image, until it is replaced with this function; the value returned by coordsys() is a copy of, not a reference to, the image’s coordinate system.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Record describing new Coordinate System</th>
</tr>
</thead>
<tbody>
<tr>
<td>csys</td>
<td>allowed: record</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

bool

Example

```python
# 
print "\t----\t setcoordsys Ex 1 \t----"
ia.fromshape(shape=[10,20])  # Make image
mycs = ia.coordsys();  # Recover Coordinate System
incr = mycs.increment('n');  # Get increment as numeric vector
incrn = incr['numeric']
for i in range(len(incrn)):
    incrn[i] = 2*incrn[i]
```

271
incr['numeric']=incrn
mycs.setincrement(value=incr);  # Set new increment in Coordinate System
ia.setcoordsys(mycs.torecord());  # Set new Coordinate System in image
mycs.done()
ia.close()
#
""
image.sethistory.html

**image.sethistory - Function**

1.1.1 Set the history for an image

**Description**

A CASA *image* file can accumulate history information from an input FITS file or by you writing something into it explicitly with this function. Each element of the input vector is one line of history. The new history is appended to the old. You can recover the history information with function history.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>origin</td>
<td>Used to set message origin. Default is image::sethistory.</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td>history</td>
<td>New history</td>
</tr>
<tr>
<td></td>
<td>allowed: stringArray</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```plaintext

```
#

##
image.setmiscinfo.html

**image.setmiscinfo - Function**

Set the miscellaneous information record for an image

**Description**

A CASA image file can accumulate miscellaneous information during its lifetime; it is stored in a record called the miscinfo record. For example, the FITS reader (fromfits) puts header keywords it doesn’t otherwise use into the miscinfo record. The miscinfo record is not guaranteed to have any entries, so it’s up to you to check for any fields that you require.

This function sets the miscinfo record of the image file. Note that this function replaces the record, it doesn’t add to it, so if you want to augment the existing record, you should first capture it with the miscinfo function, add to the record, and then put it back. The FITS writer will attempt to write all the fields in the miscinfo record to the FITS file. It can do so for scalars and 1-dimensional arrays. Records will be omitted, and multi-dimensional arrays will be flattened into 1-dimensional arrays.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>info</td>
<td>Miscellaneous REPLACEMENT header</td>
</tr>
<tr>
<td>allowed:</td>
<td>record</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
print "\t----\t setmiscinfo Ex 1 \t----"
ia.maketestimage('myfile',overwrite=true)
```

275
info = ia.miscinfo()  # capture the miscinfo record
info['extra'] = 'a test entry'  # add our information
ia.setmiscinfo(info)  # put it back into the image
ia.close()
image.shape.html

**image.shape - Function**

1.1.1 Length of each axis in the image

**Description**

The shape of an image is a vector holding the length of each axis of the image. Although this information is also available in the summary function, it is so useful that it can be obtained directly. Both Float and Complex valued images are supported.

**Arguments**

**Returns**

intArray

**Example**

```python

# print "\t----\t shape Ex 1 \t----"
ia.fromshape(shape=[10,20,30])
imshape = ia.shape()
print imshape
# [10L, 20L, 30L]
# npixels = imshape[0]*imshape[1]*...*imshape[n-1]
npixels=1
for i in range(len(imshape)):
    npixels=npixels*imshape[i]
ia.close()
# 
""
```

277
image.setrestoringbeam.html

**image.setrestoringbeam - Function**

1.1.1 Set the restoringbeam

**Description**

This function sets the restoring beam(s) for an image. You may supply the beam in one of two ways.

First, you can use the argument **beam** which you must assign to a record containing fields 'major', 'minor' and 'positionangle'. Each of these fields contains a quantity. This record is in the same format as one returned by function restoringbeam. If **beam** is used, the arguments **major**, **minor**, & **pa** are ignored.

Second, you can use the arguments **major**, **minor** and **pa**. Only the ones that you assign are used. Each argument should be assigned either a quantity or a float (units are implicitly those of the current beam - or if none, arcsec for the axes and degrees for the position angle). These parameters are only used if **beam** is not specified.

An image must have exactly one of the following states:
1. An image can have a single "traditional" beam. In that case, the beam applies to every channel and polarization in the image.
2. If an image has more than one spectral channel or more than one polarization, it can have a set of beams. In this case, each channel and/or polarization will have its own beam.
3. An image can have neither a traditional beam nor a beam set. It is never permissible for an image to have both a traditional (global) beam and a set of per-plane beams. Task and method behavior is undefined in that case and any resulting products are considered corrupt.

**RULES FOR BEAM MODIFICATION**

If an image has no beams, a traditional (global) beam can be added by setting both channel and polarization to negative values.

If an image has no beams, a set of per-plane beams can be added by setting either or both channel and/or polarization to a non-negative value. In this case, a number of per-plane beams are added consistent with the image and they are all set to be the same with parameters equal to those specified by either the beam or major/Minor/pa parameters.

If an image has a traditional beam, it can be modified by setting both channel and polarization to negative values. If one or both is not set to a negative value, an exception is thrown, and nothing is modified.

If an image has a set of per plane beams, one at a time of these can be modified by setting the appropriate channel number and/or polarization.
number. All the per-plane beams can be modified to the same values in one go by setting both channel and polarization to negative values. Also, in the case where an image has multiple channels, the beams associated with all channels for a given polarization can be modified to the same beam by setting polarization equal to the desired polarization plane number and by setting channel to a negative value. Similarly, in the case where an image has multiple polarizations, the beams associated with all polarizations for a given spectral channel can be modified to the same beam by setting channel equal to the desired spectral channel number and by setting polarization to a negative value.

A beam or set of beams can be copied from another image using the imagename parameter to specify that image’s name. If both the current image and specified image have multiple beams, the current image shape must be consistent with the specified image beam set shape.

The traditional beam or a set of multiple beams can be deleted from an image by setting delete=T. If set to true, all other parameters are then ignored; all existing beams will be irrevocably deleted.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>major</td>
<td>Major axis FWHM, Quantity or float (e.g., 1arcsec). Default is unset.</td>
<td>any</td>
<td>variant 1arcsec</td>
</tr>
<tr>
<td>minor</td>
<td>Minor axis FWHM, Quantity or float (e.g., 1arcsec). Default is unset.</td>
<td>any</td>
<td>variant 1arcsec</td>
</tr>
<tr>
<td>pa</td>
<td>Position angle, Quantity or float (e.g., ’5deg’). Default is unset.</td>
<td>any</td>
<td>variant 0deg</td>
</tr>
<tr>
<td>beam</td>
<td>The complete restoring beam (output of restoring-beam()). Default is unset.</td>
<td>record</td>
<td>record</td>
</tr>
<tr>
<td>remove</td>
<td>Delete the restoring beam?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>log</td>
<td>Write new beam values to the logger?</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>channel</td>
<td>Zero-based channel number for which to set a per plane beam. If the image has a traditional beam, set to less than zero. Default -1.</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>polarization</td>
<td>Zero-based polarization number for which to set a per plane beam. If the image has a traditional beam, set to less than zero. Default -1.</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>imagename</td>
<td>Copy the beam(s) from the specified image to this image. If multiple beams, the current image must be able to hold a beam set of the shape in the specified image.</td>
<td>string</td>
<td>string</td>
</tr>
</tbody>
</table>

| Returns | bool |

| Example |

281
ia.maketestimage('hcn', overwrite=true)
rb = ia.restoringbeam()  # returns beam in record
print rb
#{'major': {'unit': 'arcsec', 'value': 53.500004857778549},
# 'minor': {'unit': 'arcsec', 'value': 34.199998900294304},
# 'positionangle': {'unit': 'deg', 'value': 6.0}}
rb['minor']['value'] = 12.5
# new beam specified in record
# NOTE This will not work for an image with multiple beams
ia.setrestoringbeam(beam=rb)
print ia.restoringbeam()
#{'major': {'unit': 'arcsec', 'value': 53.500004857778549},
# 'minor': {'unit': 'arcsec', 'value': 12.5},
# 'positionangle': {'unit': 'deg', 'value': 6.0}}
# beam specified using parameter
# NOTE This will only work for an image with a traditional beam
ia.setrestoringbeam(major='36arcsec')
print ia.restoringbeam()
#{'major': {'unit': 'arcsec', 'value': 36.0},
# 'minor': {'unit': 'arcsec', 'value': 12.5},
# 'positionangle': {'unit': 'deg', 'value': 6.0}}
ia.setrestoringbeam(remove=true)
print ia.restoringbeam()
#
ia.setrestoringbeam(major='53.5arcsec', minor='34.2arcsec', pa='6deg')
print ia.restoringbeam()
#{'major': {'unit': 'arcsec', 'value': 53.5},
# 'minor': {'unit': 'arcsec', 'value': 34.200000000000008},
# 'positionangle': {'unit': 'deg', 'value': 6.0}}
ia.close()

# Copy all beams from an image with multiple beams to another
# image with the same number of channels and polarizations
ia.open("multibeam.im")
ib = iatool()
ib.open("target.im")

# ensure target has no beam(s) at start, not always necessary
# but it doesn’t hurt to do it.
ib.setrestoringbeam(remove=True)
# Now copy the beams. This only will work correctly if both images
# have the same number of channels and polarizations. nchan is set to
# the number of channels and npol is set to the number of polarizations
for c in range(nchan):
    for p in range(npol):
        beam = ia.restoringbeam(channel=c, polarization=p)
        ib.setrestoringbeam(beam=beam, channel=c, polarization=p)

ia.done()
ib.done()
image.statistics.html

**image.statistics - Function**

1.1.1 Compute statistics from the image

**Description**

This function computes statistics from the pixel values in the image. You can then list them and retrieve them (into a record) for further analysis. The chunk of the image over which you evaluate the statistics is specified by an array of axis numbers (argument *axes*). For example, consider a 3-dimensional image for which you specify *axes*=[0,2]. The statistics would be computed for each XZ (axes 0 and 2) plane in the image. You could then examine those statistics as a function of the Y (axis 1) axis. Or perhaps you set *axes*=[2], whereupon you could examine the statistics for each Z (axis 2) profile as a function of X and Y location in the image.

Each statistic is stored in an array in one named field in the returned record. The shape of that array is that of the axes which you did **not** evaluate the statistics over. For example, in the second example above, we set *axes*=[2] and asked for statistics as a function of the remaining axes, in this case, the X and Y (axes 0 and 1) axes. The shape of each statistics array is then [nx,ny].

The names of the fields in this record are the same as the names of the statistics that you can plot:

- **npts** - the number of unmasked points used
- **sum** - the sum of the pixel values: $\sum I_i$
- **flux** - flux or flux density, see below for details
- **sumsq** - the sum of the squares of the pixel values: $\sum I_i^2$
- **mean** - the mean of pixel values: $\bar{I} = \sum I_i/n$
- **sigma** - the standard deviation about the mean: $\sigma^2 = (\sum I_i - \bar{I})^2/(n-1)$
- **rms** - the root mean square: $\sqrt{\sum I_i^2/n}$
- **min** - minimum pixel value
- **max** - the maximum pixel value
- **median** - the median pixel value (if robust=T)
- **medabsdevmed** - the median of the absolute deviations from the median (if robust=T)
• **quartile** - the inter-quartile range (if robust=T). Find the points which are 25% largest and 75% largest (the median is 50% largest).

• **q1** - The first quartile. Reported only if robust=T.

• **q3** - The third quartile. Reported only if robust=T.

• **blc** - the absolute pixel coordinate of the bottom left corner of the bounding box of the region of interest. If 'region' is unset, this will be the bottom left corner of the whole image.

• **blcf** - the formatted absolute world coordinate of the bottom left corner of the bounding box of the region of interest.

• **trc** - the absolute pixel coordinate of the top right corner of the bounding box of the region of interest.

• **trcf** - the formatted absolute world coordinate of the top right corner of the bounding box of the region of interest.

• **minpos** - absolute pixel coordinate of minimum pixel value

• **maxpos** - absolute pixel coordinate of maximum pixel value

• **minposf** - formatted string of the world coordinate of the minimum pixel value

• **maxposf** - formatted string of the world coordinate of the maximum pixel value

The last four fields only appear if you evaluate the statistics over all axes in the image. As an example, if the returned record is captured in 'mystats', then you could access the 'mean' field via `print mystats['mean']`. If there are no good points (e.g. all pixels are masked bad in the region), then the length of these fields will be 0 (e.g. `len(mystats['npts'])==0`).

You have no control over which statistics are listed to the logger, you always get the same selection. You can choose to list the statistics or not (argument list).

As well as the simple (and faster to calculate) statistics like means and sums, you can also compute some robust (quantile-like) statistics. Currently these are the median, median absolute deviations from the median, the first and third quartiles, and the inner-quartile range. Because these are computationally expensive, they are only computed if robust=True.

Note that if the axes are set to all of the axes in the image (which is the default) there is just one value per statistic.

You have control over which pixels are included in the statistics computations via the includepix and excludepix arguments. These vectors specify a range of pixel values for which pixels are either included or excluded. They are mutually exclusive; you can specify one or the other, but not both. If you only
give one value for either of these, say $\text{includepix}=b$, then this is interpreted as $\text{includepix}=[-\text{abs}(b),\text{abs}(b)]$.

This function generates a ‘storage’ lattice, into which the statistics are written. It is only regenerated when necessary. For example, if you run the function twice with identical arguments, the statistics will be directly retrieved from the storage lattice the second time. However, you can force regeneration of the storage image if you set $\text{force}=T$. The storage medium is either in memory or on disk, depending upon its size. You can force it to disk if you set $\text{disk}=T$, otherwise it decides for itself.

**ALGORITHMS**

Several types of statistical algorithms are supported:

* classic: This is the familiar algorithm, in which all unmasked pixels, subject to any specified pixel ranges, are used. One may choose one of two methods, which vary only by performance, for computing classic statistics, via the clmethod parameter. The "tiled" method is the old method and is fastest in cases where there are a large number of individual sets of statistics to be computed and a small number of data points per set. This can occur when one sets the axes parameter, which causes several individual sets of statistics to be computed. The "framework" method uses the new statistics framework to compute statistics. This method is fastest in the regime where one has a small number of individual sets of statistics to calculate, and each set has a large number of points. For example, this method is fastest when computing statistics over an entire image in one go (no axes specified). A third option, "auto", chooses which method to use by predicting which be faster based on the number of pixels in the image and the choice of the axes parameter.

* fit-half: This algorithm calculates statistics on a dataset created from real and virtual pixel values. The real values are determined by the input parameters center and lside. The parameter center tells the algorithm where the center value of the combined real+virtual dataset should be. Options are the mean or the median of the input image’s pixel values, or at zero. The lside parameter tells the algorithm on which side of this center the real pixel values are located. True indicates that the real pixel values to be used are $\bar{x} = \text{center}$. False indicates the real pixel values to be used are $\bar{x} = \text{center}$. The virtual part of the dataset is then created by reflecting all the real values through the center value, to create a perfectly symmetric dataset composed of a real and a virtual component. Statistics are then calculated on this resultant dataset. These two parameters are ignored if algorithm is not "fit-half". Because the maximum value is virtual if lside is True and the minimum value is virtual if lside is False, the value of the maximum position (if lside=True) or minimum position (if lside=False) is not reported in the returned record.

* hinges-fences: This algorithm calculates statistics by including data in a range between $Q_1 - f*D$ and $Q_3 + f*D$, inclusive, where $Q_1$ is the first quartile of the distribution of unmasked data, subject to any specified pixel ranges, $Q_3$ is the third quartile, $D = Q_3 - Q_1$ (the inner quartile range), and $f$ is the user-specified fence factor. Negative values of $f$ indicate that the full distribution is to be used (ie, the classic algorithm is used). Sufficiently large
values of $f$ will also be equivalent to using the classic algorithm. For $f = 0$, only data in the inner quartile range is used for computing statistics. The value of fence is silently ignored if algorithm is not "hinges-fences".

* chauvenet: The idea behind this algorithm is to eliminate outliers based on a maximum z-score value. A z-score is the number of standard deviations a point is from the mean of a distribution. This method thus is meant to be used for (nearly) normal distributions. In general, this is an iterative process, with successive iterations discarding additional outliers as the remaining points become closer to forming a normal distribution. Iterating stops when no additional points lie beyond the specified zscore value, or, if zscore is negative, when Chauvenet’s criterion is met (see below). The parameter maxiter can be set to a non-negative value to prematurely abort this iterative process. When verbose=T, the "N iter" column in the table that is logged represents the number of iterations that were executed.

Chauvenet’s criterion allows the target z-score to decrease as the number of points in the distribution decreases on subsequent iterations. Essentially, the criterion is that the probability of having one point in a normal distribution at a maximum z-score of $z_{\text{max}}$ must be at least 0.5. $z_{\text{max}}$ is therefore a function of (only) the number of points in the distribution and is given by

$$npts = \frac{0.5}{\text{erfc}(z_{\text{max}}/\sqrt{2})}$$

where erfc() is the complementary error function. As iterating proceeds, the number of remaining points decreases as outliers are discarded, and so $z_{\text{max}}$ likewise decreases. Convergence occurs when all remaining points fall within a z-score of $z_{\text{max}}$. Below is an illustrative table of $z_{\text{max}}$ values and their corresponding npts values. For example, it is likely that there will be a 5-sigma "noise bump" in a perfectly noisy image with one million independent elements.

<table>
<thead>
<tr>
<th>$z_{\text{max}}$</th>
<th>npts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0 11.5 3 2.0 10 2.5 40 3.0 185 3.5 1,074 4.0 7,893 4.5 73,579 5.0 872,138 5.5 13,165,126 6.0 253,398,672 6.5 6,225,098,696 7.0 195,341,107,722</td>
<td></td>
</tr>
</tbody>
</table>

NOTES ON FLUX DENSITIES AND FLUXES

Fluxes and flux densities are not computed if any of the following conditions is met:

1. The image does not have a direction coordinate 2. The image does not have an intensity-like brightness unit. Examples of such units are Jy/beam (in which case the image must also have a beam) and K. 3. There are no direction axes in the cursor axes that are used. 4. If the (specified region of the) image has a non-degenerate spectral axis, and the image has a tabular spectral axis (axis with varying increments) 5. Any axis that is not a direction nor a spectral axis that is included in the cursor axes is not degenerate within in the specified region

Note that condition 4 may be removed in the future.

In cases where none of the above conditions is met, the flux density(ies) (intensities integrated over direction planes) will be computed if any of the following conditions are met:

1. The image has no spectral coordinate 2. The cursor axes do not include the spectral axis 3. The spectral axis in the chosen region is degenerate
In the case where there is a nondegenerate spectral axis that is included in the
cursor axes, the flux (flux density integrated over spectral planes) will be
computed. In this case, the spectral portion of the flux unit will be the
velocity unit of the spectral coordinate if it has one (eg, if the brightness unit
is Jy/beam and the velocity unit is km/s, the flux will have units of Jy.km/s).
If not, the spectral portion of the flux unit will be the frequency unit of the
spectral axis (eg, if the brightness unit is K and the frequency unit is Hz, the
resulting flux unit will be K.arcsec2.Hz).
In both cases of flux density or flux being computed, the resulting numerical
value is assigned to the "flux" key in the output dictionary.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>axes</td>
<td>List of axes to evaluate statistics over. Default is all axes.</td>
<td>intArray</td>
<td>-1</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask. Default is none.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>includepix</td>
<td>Range of pixel values to include. Vector of 1 or 2 doubles. Default is to include all pixels.</td>
<td>doubleArray</td>
<td>-1</td>
</tr>
<tr>
<td>excludepix</td>
<td>Range of pixel values to exclude. Vector of 1 or 2 doubles. Default is exclude no pixels.</td>
<td>doubleArray</td>
<td>-1</td>
</tr>
<tr>
<td>list</td>
<td>If True print bounding box and statistics to logger.</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>force</td>
<td>If T then force the stored statistical accumulations to be regenerated</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>disk</td>
<td>If T then force the storage image to disk</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>robust</td>
<td>If T then compute robust statistics as well</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>verbose</td>
<td>If T then log statistics</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>stretch</td>
<td>Stretch the mask if necessary and possible? See help par.stretch. Default False</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>logfile</td>
<td>Name of file to which to write statistics.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>append</td>
<td>Append results to logfile? Logfile must be specified. Default is to append. False means overwrite existing file if it exists.</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>algorithm</td>
<td>Algorithm to use. Supported values are &quot;chauvenet&quot;, &quot;classic&quot;, &quot;fit-half&quot;, and &quot;hinges-fences&quot;. Minimum match is supported.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>fence</td>
<td>Fence value for hinges-fences. A negative value means use the entire data set (ie default to the &quot;classic&quot; algorithm). Ignored if algorithm is not &quot;hinges-fences&quot;.</td>
<td>double</td>
<td>-1</td>
</tr>
<tr>
<td>center</td>
<td>Center to use for fit-half. Valid choices are &quot;mean&quot;, &quot;median&quot;, and &quot;zero&quot;. Ignored if algorithm is not &quot;fit-half&quot;.</td>
<td>string</td>
<td>mean</td>
</tr>
<tr>
<td>lside</td>
<td>For fit-half, real data are (&lt; = ) center? If false, real data are (&gt; = ) center. Ignored if algorithm is not &quot;fit-half&quot;.</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>zscore</td>
<td>For chauvenet, this is the target maximum number of standard deviations data may have to be included. If negative, use Chauvenet's criterion. Ignored if algorithm is not &quot;chauvenet&quot;.</td>
<td>double</td>
<td>-1</td>
</tr>
<tr>
<td>maxiter</td>
<td>For chauvenet, this is the maximum number of iterations to attempt. Iterating will stop when either this limit is reached, or the zscore criterion is met. If negative, iterate until the zscore criterion is met. Ignored if algorithm is not &quot;chauvenet&quot;.</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>clmethod</td>
<td>Method to use for calculating classical statistics. Supported methods are &quot;auto&quot;, &quot;tiled&quot;, and &quot;framework&quot;. Ignored if algorithm is not &quot;classic&quot;.</td>
<td>string</td>
<td>auto</td>
</tr>
<tr>
<td>algorithm</td>
<td>Algorithm to use. Supported values are &quot;chauvenet&quot;, &quot;classic&quot;, &quot;fit-half&quot;, and &quot;hinges-fences&quot;. Minimum match is supported.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>fence</td>
<td>Fence value for hinges-fences. A negative value means use the entire data set (ie default to the &quot;classic&quot; algorithm). Ignored if algorithm is not &quot;hinges-fences&quot;.</td>
<td>double</td>
<td>-1</td>
</tr>
</tbody>
</table>
Returns record

Example

""
#
print "\t----\t statistics Ex 1 \t----"
ia.maketestimage()
ia.statistics()
ia.close()
#

# evaluate statistics for each spectral plane in an ra x dec x frequency image
ia.fromshape("", [20,30,40])
# give pixels non-zero values
ia.addnoise()
# These are the display axes, the calculation of statistics occurs
# for each (hyper)plane along axes not listed in the axes parameter,
# in this case axis 2 (the frequency axis)
# display the rms for each frequency plane (your mileage will vary with
# the values).
stats = ia.statistics(axes=[0,1])
stats["rms"]
Out[10]:
array([ 0.99576014, 1.03813124, 0.97749186, 0.97587883, 1.04189885,
       1.03784776, 1.03371549, 1.03153074, 1.00841606, 0.9471550 , 0.97335404,
       0.94389403, 1.0010221 , 0.97151822, 1.03942156,
       1.01158476, 0.96957082, 1.04212773, 1.00589049, 0.98661619,
       0.92088714, 0.96999902, 0.98661619, 1.01039267, 0.96842754,
       0.99464947, 1.01536798, 1.02466023, 0.96956468, 0.98090756,
       0.9835844 , 0.95698935, 1.05487967, 0.99846411, 0.99634868])
""

In this example, we ask to see statistics evaluated over the entire image.
Example

```python
#
print "\t----\t statistics Ex 2 \t----"
ia.maketestimage()
stats = ia.statistics(axes=[1],plotstats=["sigma","rms"],
                        includepix=[0,100],list=F)
#
```

In this example, let us assume the image has 2 dimensions. We want the standard deviation about the mean and the rms of Y (axes 1) for pixels with values in the range 0 to 100 as a function of the X-axis location. The statistics are not listed to the logger but are saved in the record `{\stf af 'stats'}`.
image.twopointcorrelation.html

image.twopointcorrelation - Function

1.1.1 Compute two point correlation function from the image

Description

This function computes two-point auto-correlation functions from an image. By default, the auto-correlation function is computed for the Sky axes. If there is no sky in the image, then the first two axes are used. Otherwise you can specify which axes the auto-correlation function lags are computed over with the `axes` argument (must be of length 2).

Presently, only the Structure Function is implemented. This is defined as:

$$ S(lx, ly) = < (data(i, j) - data(i + lx, j + ly))^2 > $$

where $lx, ly$ are integer lags in the x (0-axis) and y (1-axis) directions. The ensemble average is over all the values at the same lag pair. This process is extremely compute intensive and so you may have to be patient.

In an auto-correlation function image there are some symmetries. The first and third quadrants are symmetric, and the second and fourth are symmetric. So in principle, all the information is in the top or bottom half of the image. We just write it all out to look nice. The long lags don’t have a lot of contributing values of course.

Arguments
Inputs

outfile  Output image file name. Default is unset.
  allowed:  string
  Default:

region  Region selection. See "help par.region" for details. Default is to use the full image.
  allowed:  any
  Default:  variant

mask  Mask to use. See help par.mask. Default is none.
  allowed:  any
  Default:  variant

axes  The pixel axes to compute structure function over. The default is sky or first two axes.
  allowed:  intArray
  Default:  -1

method  The method of computation. String from 'structurefunction'.
  allowed:  string
  Default:  structurefunction

overwrite  Overwrite (unprompted) pre-existing output file?
  allowed:  bool
  Default:  false

stretch  Stretch the mask if necessary and possible? See help par.stretch. Default False
  allowed:  bool
  Default:  false

Returns

bool

Example

""
#
print "\t\t\twopointcorrelation Ex 1 \t\t"
ia.maketestimage();  # Output image is virtual
ia.twopointcorrelation()  # Output image is virtual
#
image.subimage.html

**image.subimage - Function**

1.1.1 Create a (sub)image from a region of the image

**Description**

This function copies all or part of the image to another on-the-fly Image tool. Both float and complex valued images are supported. If `outfile` is given, the subimage is written to the specified disk file. If `outfile` is unset, the returned Image tool actually references the input image file (i.e. that associated with the Image tool to which you are applying this function). So if you deleted the input image disk file, it would render this tool useless. When you destroy this tool (with the done function) the reference connection is broken.

Sometimes it is useful to drop axes of length one (degenerate axes). Use the `dropdeg` argument if you want to do this. Further control is provided via the `keepaxes` parameter. If `dropdeg=True`, you may specify a list of degenerate axes to keep in the `keepaxes` parameter. This allows you to drop only a subset of degenerate axes. This parameter is ignored if `dropdeg=False`. If `dropdeg=True`, all degenerate axes are dropped if `keepaxes` is set to an empty list (this is the default behavior). Nondegenerate axes are implicitly kept, even if they are included in the `keepaxes` list.

The output mask is the combination (logical OR) of the default input pixel mask (if any) and the OTF mask. Any other input pixel masks will not be copied. Use function maskhandler if you need to copy other masks too.

If the mask has fewer dimensions than the image and if the shape of the dimensions the mask and image have in common are the same, the mask will automatically have the missing dimensions added so it conforms to the image. If `stretch` is true and if the number of mask dimensions is less than or equal to the number of image dimensions and some axes in the mask are degenerate while the corresponding axes in the image are not, the mask will be stretched in the degenerate dimensions. For example, if the input image has shape [100, 200, 10] and the input mask has shape [100, 200, 1] and `stretch` is true, the mask will be stretched along the third dimension to shape [100, 200, 10]. However if the mask is shape [100, 200, 2], stretching is not possible and an error will result.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask. Default is none.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>dropdeg</td>
<td>Drop degenerate axes</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>list</td>
<td>List informative messages to the logger</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>stretch</td>
<td>Stretch the mask if necessary and possible?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>wantreturn</td>
<td>Return an image analysis tool attached to the created subimage</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>keepaxes</td>
<td>If dropdeg=True, these are the degenerate axes to keep. Nondegenerate axes are implicitly always kept.</td>
<td>intArray</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

image

**Example**

```
```
# print "\t----\t subimage Ex 1 \t----"
ia.maketestimage('myfile',overwrite=true)
im2 = ia.subimage() # a complete copy
r1 = rg.box([10,10],[30,40],[5,5]) # A strided pixel box region
im3 = ia.subimage(outfile='/tmp/foo', region=r1, overwrite=true)
        # Explicitly named subimage

im2.done()
im3.done()
ia.close()
#

# As an example of the usage of the keepaxes parameter, consider an image
# that has axes RA, Dec, Stokes, and Freq. The Stokes and Freq axes are both
# degenerate while the RA and Dec axes are not, and it is desired to make a
# subimage in which the Stokes axis is discarded. The following command will
# accomplish that.
ia.open("my.im")
subim = ia.subimage(outfile="discarded_stokes.im", dropdeg=True, keepaxes=[3])
ia.done()
subim.done()
image.summary.html

**image.summary - Function**

1.1.1 Summarize basic information about the image

**Description**

This function summarizes miscellaneous information such as shape, Coordinate System, restoring beams, and masks. If called without any arguments, this function displays a summary of the image header to the logger; where appropriate, values will be formatted nicely (e.g. HH:MM:SS.SS for the reference value of RA axes). For spectral axes, the information is listed as a velocity as well as a frequency. The argument `doppler` allows you to specify what velocity doppler convention it is listed in. You can choose from `radio`, `optical` and `true`. Alternative names are `z` for `optical`, and `beta` or `relativistic` for `true`. The default is `radio`. The definitions are

- radio: $1 - F$
- optical: $-1 + 1/F$
- true: $(1 - F^2)/(1 + F^2)$

where $F = \nu/\nu_0$ and $\nu_0$ is the rest frequency. If the rest frequency has not been set in your image, you can set it via a Coordsys tool with the function `setrestfrequency`. If the output of summary is saved to a variable, then the `header` field (for instance, `mysummary['header']`) has the following fields filled in:

- **ndim** Dimension of the image.
- **shape** Length of each axis in the image.
- **tileshape** Shape of the chunk which is most efficient for I/O.
- **axisnames** Name of each axis.
- **refpix** Reference pixel for each axis (0-relative)
- **refval** Reference value for each axis.
- **incr** Increment for each axis.
- **axisunits** Unit name for each axis.
unit  Brightness units for the pixels.

hasmask  True if the image has a mask.

defaultmask  The name of the mask which is applied by default.

masks  The names of all the masks stored in this image.

restoringbeam  The restoring beam(s) if present.

imagetype  The image type.

For an image with multiple beams, the restoringbeam field is a dictionary of
dictionaries with keys of names "*" followed by the channel number, if the
image has a spectral coordinate, or the polarization number if it does not.
That is, the keys have names "*0", "*1", "*2", etc. If the image has both a
spectral and a polarization coordinate, each of these dictionaries is a dictionary
with keys of the same form which range from 0 to the number of polarizations
minus 1; "*0", "*1", ... The dictionaries pointed to by the channel and/or
polarization number contain information for the beam at that position.
If you set list=F, then the summary will not be written to the logger. The
return value of the function, in the header field is a vector string containing
the formatted output that would normally have gone to the logger.
If verbose is True and the image contains multiple beams, the formatted
output, whether it is written to the logger or placed in the output record, will
have information on every beam in the dataset. If verbose=False and the
image has multiple beams, only a summary of beams for each polarization is
listed. In this case, the beams with the maximum area, the minimum area,
and the median area for each polarization are listed. However, all the beams
can still be found in the restoringbeam field of the returned dictionary. If the
image does not have multiple beams, verbose is not used.

Arguments
Inputs

doppler        If there is a spectral axis, list velocity too, with this doppler definition
    allowed:     string
    Default:     RADIO

list          List the summary to the logger
    allowed:     bool
    Default:     true

pixelorder    List axis descriptors in pixel or world axis order
    allowed:     bool
    Default:     true

verbose       Give a full listing of beams or just a short summary?
    Only used when the image has multiple beams.
    allowed:     bool
    Default:     false


Returns

record

Example

```python
"""
#
print "\t----\t summary Ex 1 \t----"
ia.maketestimage('myim1', overwrite=true)
ia.summary() # summarize to logging only
s = ia.summary(list=F) # store header in record
if s['header']['ndim'] == 2: # program using header values
    print s['header']['axisnames']
ia.close()
#
"""
image.tofits.html

**image.tofits - Function**

1.1.1 Convert the image to a FITS file

**Description**

This function converts the image into a FITS file. If the image has a rest frequency associated with it, it will always write velocity information into the FITS file. By default the frequency information will be primary as it is the internal native format. If you select `velocity=T` then by default the velocity is written in the optical convention, but if `optical=F` it will use the radio convention instead. Alternatively, if you use `velocity=F` and `wavelength=T`, the spectral axis will be written in wavelength.

The FITS definition demands equal increment pixels. Therefore, if you write wavelength or optical velocity information as primary, the increment is computed at the spectral reference pixel. If the bandwidth is large, this may incur non-negligible coordinate calculation errors far from the reference pixel if the spectral bins are not originally equidistant in wavelength. Images generated by the CASA clean task have spectral axes which are always equidistant in frequency.

By default the image is written as a floating point FITS file (`bitpix= -32`). Under rare circumstances you might want to save space and write it as scaled 16 bit integers (`bitpix = 16`). You can have `tofits` calculate the scaling factors by using the default `minpix` and `maxpix`. If you set `minpix` and `maxpix`, values outside of that range will be truncated. This can be useful if all of the FITS images dynamic range is being used by a few high or low values and you are not interested in preserving those values exactly. Besides the factor of two space savings you get by using 16 instead of 32 bits, integer images usually also compress well (for example, with the standard GNU software facility `gzip`).

If the specified `region-of-interest` extends beyond the image, it is truncated.

The output mask is the combination (logical OR) of the default input `pixel mask` (if any) and the OTF mask. Sometimes it is useful to drop axes of length one (degenerate axes) because not all FITS readers can handle them. Use the `dropdeg` argument if you want to do this. If you want to specifically only drop a degenerate Stokes axis, use the `dropstokes` argument.

If you want to place degenerate axes last in the FITS header, use the `deglast` argument. If you want to make sure that the Stokes axis is placed last in the FITS header, use the `stokeslast` argument.
Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>FITS file name. Default is input name + <code>.fits</code></td>
</tr>
<tr>
<td>velocity</td>
<td>prefer velocity (rather than frequency) as primary spectral axis?</td>
</tr>
<tr>
<td>optical</td>
<td>use the optical (rather than radio) velocity convention?</td>
</tr>
<tr>
<td>bitpix</td>
<td>Bits per pixel, -32 (floating point) or 16 (integer)</td>
</tr>
<tr>
<td>minpix</td>
<td>Minimum pixel value for BITPIX=16, Default is to autoscale if minpix &gt; maxpix.</td>
</tr>
<tr>
<td>maxpix</td>
<td>Maximum pixel value for BITPIX=16, Default is to autoscale if maxpix &lt; minpix.</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask. Default is none.</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file?</td>
</tr>
<tr>
<td>dropdeg</td>
<td>Drop degenerate axes?</td>
</tr>
<tr>
<td>deglast</td>
<td>Put degenerate axes last in header?</td>
</tr>
<tr>
<td>dropstokes</td>
<td>Drop Stokes axis?</td>
</tr>
<tr>
<td>stokeslast</td>
<td>Put Stokes axis last in header?</td>
</tr>
<tr>
<td>wavelength</td>
<td>Write spectral axis in units of wavelength (instead of velocity or frequency)?</td>
</tr>
<tr>
<td>airwavelength</td>
<td>When writing the spectral axis in units of wavelength, use air wavelength instead of vacuum wavelength?</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously?</td>
</tr>
</tbody>
</table>
Returns
bool

Example

""
#
print "\t----\t tofits Ex 1 \t----"
ia.maketestimage()
ok = ia.tofits('MYFILE.FITS', overwrite=true)
    # write FITS image file
ok = ia.tofits('MYFILE2.FITS', bitpix=16, overwrite=true)
    # Write as scaled 16 bit integers
ia.close()
#
"""
image.toASCII.html

**image.toASCII - Function**

1.1.1 Convert the image to an ASCII file

**Description**

This function converts the image into an ASCII file. The format is one image row per line (see fromascii).

The output mask is the combination (logical OR) of the default input pixel mask (if any) and the OTF mask. Because the mask is not transferred to the ascii file, you must specify what data value to use if a pixel is masked. By default, the underlying data value in the image is used. But this could be anything (and often it’s a NaN), so you could set, say, `maskvalue=-10000` as a magic value.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>ASCII file name. Default is input name + '.ascii'.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask. Default is none.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>sep</td>
<td>Separator of data in ascii file. Default is space character.</td>
<td>string</td>
<td>:</td>
</tr>
<tr>
<td>format</td>
<td>Format of data in ascii file</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>maskvalue</td>
<td>Value to replace masked pixels by. -999 is no change.</td>
<td>double</td>
<td>-999</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>stretch</td>
<td>Stretch the mask if necessary and possible? See help par.stretch. Default False</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```
#
print "\t----\t toASCII Ex 1 \t----"
ia.maketestimage()
```
ok = ia.toASCII('myfile.ascii', overwrite=true)
ia.close()
ia.fromascii('image.im', 'myfile.ascii', shape=[113,76], overwrite=true)
ia.toASCII('myfile2.ascii', overwrite=true)
#!diff myfile.ascii myfile2.ascii
#
"""
image.torecord.html

**image.torecord - Function**

1.1.1 Return a record containing the image associated with this tool

**Description**

You can convert an associated image to a record for manipulation or passing it to inputs of other function of other tools. This and fromrecord are used for serialization and deserialization.

**Arguments**

**Returns**

record

**Example**

```
#
print "\t----\t torecord Ex 1 \t----"
ia.maketestimage('image.large', overwrite=true)
rec=ia.torecord()
ia.close()
```

```
image.type.html

**image.type - Function**

[1.1.1] Return the type of this tool

**Description**

This function returns the string ‘image’. It can be used in a script to make sure this variable is an Image tool.

**Arguments**

**Returns**

string
image.topixel - Function

1.1.1 Convert from world to pixel coordinate

Description

This function converts from absolute world to pixel coordinate (0-rel). The world coordinate can be provided in many formats (numeric, string, quantum etc.) via the argument `value`. These match the output formats of function `toworld`. This function is just a wrapper for the Coordsys tool function `topixel` so see the documentation there for a description and more examples.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Absolute world coordinate, Numeric vector, vector of strings representing quantities, or record of format analogous to that produced by ia.toworld(). Default is reference value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>allowed: any</td>
</tr>
</tbody>
</table>

Returns

record

Example

```
#
print "\t----\t topixel Ex 1 \t----"
ia.maketestimage();
w = ia.toworld([10,10], 'n')    # Numeric vector
ia.topixel(w)
#{'ar_type': 'absolute',
```
Convert a pixel coordinate to world as floats and then back to pixel. Do the same with the world coordinate formatted as measures instead.
image.toworld.html

**image.toworld - Function**

Convert from pixel to world coordinate

**Description**

This function converts between absolute pixel coordinate (0-rel) and world (physical coordinate).

This function is just a wrapper for the Coordsys tool function toworld so see the documentation there for a description of the arguments and more examples.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Absolute pixel coordinate. Numeric vector is allowed. Temporary reference pixel.</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
<tr>
<td>format</td>
<td>What type of formatting? String from combination of <code>n</code> (numeric), <code>q</code> (quantity), <code>m</code> (measure), <code>s</code> (string).</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>n</td>
</tr>
<tr>
<td>dovelocity</td>
<td>Compute corresponding spectral velocities?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>True</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
"""
#
```

312
print "\t----\t toworld Ex 1 \t----"
ia.maketestimage('hcn', overwrite=true)
w = ia.toworld([10,10], 'n')
print w
#{'numeric': array([ 0.00174533, -0.0015708 ])}
w = ia.toworld([10,10], 'nmq')
print w
#{'measure': {'direction': {'m0': {'unit': 'rad',
# 'value': 0.0017453323593185704},
# 'm1': {'unit': 'rad',
# 'value': -0.0015707969259645381},
# 'refer': 'J2000',
# 'type': 'direction'}},
# 'numeric': array([ 0.00174533, -0.0015708 ]),
# 'quantity': {'*1': {'unit': 'rad', 'value': 0.0017453323593185704},
# '*2': {'unit': 'rad', 'value': -0.0015707969259645381}}}
ia.close()
#
""

Convert to a vector of floats and then to a record
holding a vector of floats, a vector of quantities
and a subrecord of measures.
image.unlock.html

**image.unlock - Function**

1.1.1 Release any lock on the image

**Description**

This function releases any lock set on the image file (and also flushes any outstanding I/O to disk). It is not of general user interest. It can be useful in scripts when a file is being shared between more than one process. See also functions lock and haslock.

**Arguments**

**Returns**

bool

**Example**

```
print "\t----\t unlock Ex 1 \t----"
ia.fromarray('xx', ia.makearray(0,[10,20]), overwrite=true)
ia.unlock()
ia.close()
```

This releases the write lock on the image file. Now some other process can gain immediate access to the image file.
image.newimagefromarray.html

image.newimagefromarray - Function

1.1.1 Construct a CASA image from an array

Description

This function converts an array of any size into a CASA image file. If outfile is given, the image is written to the specified disk file. If outfile is unset, the on-the-fly Image tool returned by this function is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you destroy the on-the-fly Image tool (with the done function) this temporary image is deleted.

At present, no matter what type the pixels array is, a real-valued image will be written (the input pixels will be converted to Float). In the future, Complex images will be supported.

The Coordinate System, provided as a record describing a Coordsys tool (via coordsys.torecord(), for instance) is optional. If you provide it, it must be dimensionally consistent with the pixels array you give (see also coordsys). If you don’t provide the Coordinate System (unset), a default Coordinate System is made for you. If linear=F (the default) then it is a standard RA/DEC/Stokes/Spectral Coordinate System depending exactly upon the shape of the pixels array (Stokes axis must be no longer than 4 pixels and you may find the spectral axis coming out before the Stokes axis if say, shape=[64,64,32,4]). Extra dimensions are given linear coordinates. If linear=T then you get a linear Coordinate System.

Arguments

315
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
<td>string</td>
<td>unset</td>
</tr>
<tr>
<td>pixels</td>
<td>A numeric array is required.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>csys</td>
<td>Coordinate System. Default is unset.</td>
<td>record</td>
<td>unset</td>
</tr>
<tr>
<td>linear</td>
<td>Make a linear Coordinate System if csys not given</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>log</td>
<td>Write image creation messages to logger</td>
<td>bool</td>
<td>true</td>
</tr>
</tbody>
</table>

Returns
- image

Example

```python
""
#
print "\t----\t newimagefromarray Ex 1 \t----"
im1=ia.newimagefromarray(outfile='test.data',
pixels=ia.makearray(0, [64, 64, 4, 128]),
overwrite=true)

cs1 = im1.coordsys(axes=[0,1])
im1.done()
im2 = ia.newimagefromarray(pixels=ia.makearray(1.0, [32, 64]),
 csys=cs1.torecord())

cs1.done()
im2.done()
#
""
```

316
The first example creates a zero-filled `imagefile` named `{\texttt{test.data}}` which is of shape `[64,64,4,128]`. If you examine the header with `{\texttt{ia.summary()}}` you will see the default RA/DEC/Stokes/Frequency coordinate information. In the second example, a Coordinate System describing the first two axes of the image `{\texttt{test.data}}` is created and used to create a 2D image temporary image.
Construct a CASA image by conversion from a FITS image file

Description

This function is used to convert a FITS disk image file (Float, Double, Short, Long are supported) to an CASA image file. If outfile is given, the image is written to the specified disk file. If outfile is unset, the on-the-fly Image tool returned by this function is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you destroy the on-the-fly Image tool (with the done function) this temporary image is deleted.

This function reads from the FITS primary array (when the image is at the beginning of the FITS file; whichhdu=0), or an image extension (when the image is elsewhere in the FITS file, whichhdu > 0).

By default, any blanked pixels will be converted to a mask value which is false, and a pixel value that is NaN. If you set zeroblanks=T then the pixel value will be zero rather than NaN. The mask will still be set to false. See the function replacemaskedpixels if you need to replace masked pixel values after you have created the image.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
</tr>
<tr>
<td>allowed: string</td>
<td>Default:</td>
</tr>
<tr>
<td>infile</td>
<td>Input FITS disk file name. Required.</td>
</tr>
<tr>
<td>allowed: string</td>
<td>Default:</td>
</tr>
<tr>
<td>whichrep</td>
<td>If this FITS file contains multiple coordinate representations, which one</td>
</tr>
<tr>
<td>allowed: int</td>
<td>should we read</td>
</tr>
<tr>
<td>whichhdu</td>
<td>If this FITS file contains multiple images, which one should we read (0-based).</td>
</tr>
<tr>
<td>allowed: int</td>
<td>should we read (0-based).</td>
</tr>
<tr>
<td>zeroblanks</td>
<td>If there are blanked pixels, set them to zero instead of NaN</td>
</tr>
<tr>
<td>allowed: bool</td>
<td>Default: false</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file?</td>
</tr>
<tr>
<td>allowed: bool</td>
<td>Default: false</td>
</tr>
</tbody>
</table>

**Returns**

image

**Example**

```python
# """
# print "\t----\t newimagefromfits Ex 1 \t----"
# Assume we can find test fits file using
# CASAPATH environment variable
pathname=os.environ.get("CASAPATH")
pathname=pathname.split()[0]
datapath=pathname+'/data/demo/Images/imagetestimage.fits'
im1=ia.newimagefromfits('./myimage', datapath, overwrite=true)
print im1.summary()
print im1.miscinfo()
print 'fields=', im1.miscinfo().keys()
```

319
The FITS image is converted to a \casa\ \imagefile\ and access is provided via the \imagetool\ called \{\stf im1\}. Any FITS header keywords which were not recognized or used are put in the miscellaneous information bucket accessible with the miscinfo function. In the example we list the names of the fields in this record.
image.newimagefromimage - Function

Construct an on-the-fly image tool from a region of a CASA image file

Description

This function applies a region-of-interest to a disk image file, creates a new image file containing the (sub)image, and associates a new image tool with it.

The input disk image file may be in native CASA, FITS (Float, Double, Short, Long are supported), or Miriad format. Look here for more information on foreign images.

If outfile is given, the (sub)image is written to the specified disk file.

If outfile is unset, the Image tool actually references the input image file. So if you deleted the input image disk file, it would render this tool useless.

When you destroy this on-the-fly tool (with the done function) the reference connection is broken.

Sometimes it is useful to drop axes of length one (degenerate axes). Use the dropdeg argument if you want to do this.

The output mask is the combination (logical OR) of the default input pixel mask (if any) and the OTF mask. Any other input pixel masks will not be copied. Use function maskhandler if you need to copy other masks too.

See also the subimage function.

Arguments
## Inputs

**infile**  
Input image file name. Required.  
**allowed:** string  
**Default:**

**outfile**  
Output sub-image file name. Default is unset.  
**allowed:** string  
**Default:**

**region**  
Region selection. See "help par.region" for details. Default is to use the full image.  
**allowed:** any  
**Default:** variant

**mask**  
Mask to use. See help par.mask. Default is none.  
**allowed:** any  
**Default:** variant

**dropdeg**  
Drop degenerate axes  
**allowed:** bool  
**Default:** false

**overwrite**  
Overwrite (unprompted) pre-existing output file?  
**allowed:** bool  
**Default:** false

## Returns

**image**

## Example

```python
# print "\t\tnewimagefromimage Ex 1 \t"
ia.maketestimage('test1',overwrite=true)
ia.maketestimage('test2',overwrite=true)
print ia.name()
# [...]test2
im1=ia.newimagefromimage('test1')
print im1.name()
# [...]test1
print im1.summary()
im2=ia.newimagefromimage('test2')
```

322
print im2.name()
#([...])test2
print im1.name()
#([...])test1
ia.close()
im1.done()
im2.done()
#
""
image.newimagefromshape.html

image.newimagefromshape - Function

1.1.1 Construct an empty CASA image from a shape

Description

This function creates a CASA image file with the specified shape. All the pixel values in the image are set to 0. Both float valued and complex valued images are supported; the data type of the image is specified via the type parameter.

If outfile is given, the image is written to the specified disk file. If outfile is unset, the on-the-fly Image tool returned by this function is associated with a temporary image. This temporary image may be in memory or on disk, depending on its size. When you destroy the on-the-fly Image tool (with the done function) this temporary image is deleted.

The Coordinate System, provided as a record describing a Coordsys tool (created via coordsys.torecord(), for instance), is optional. If you provide it, it must be dimensionally consistent with the pixels array you give (see also coordsys).

If you don’t provide the Coordinate System, a default Coordinate System is made for you. If linear=F (the default) then it is a standard RA/DEC/Stokes/Spectral Coordinate System depending exactly upon the shape (Stokes axis must be no longer than 4 pixels and you may find the spectral axis coming out before the Stokes axis if say, shape=[64,64,32,4]). Extra dimensions are given linear coordinates. If linear=T then you get a linear Coordinate System.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Name of output image file. Default is unset.</td>
</tr>
<tr>
<td>allowed: string Default:</td>
<td></td>
</tr>
<tr>
<td>shape</td>
<td>Shape of image. Required.</td>
</tr>
<tr>
<td>allowed: intArray Default: 0</td>
<td></td>
</tr>
<tr>
<td>csys</td>
<td>Record describing Coordinate System. Default is unset.</td>
</tr>
<tr>
<td>allowed: record Default:</td>
<td></td>
</tr>
<tr>
<td>linear</td>
<td>Make a linear Coordinate System if csys not given?</td>
</tr>
<tr>
<td>allowed: bool Default: false</td>
<td></td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite (unprompted) pre-existing output file?</td>
</tr>
<tr>
<td>allowed: bool Default: false</td>
<td></td>
</tr>
<tr>
<td>log</td>
<td>Write image creation messages to logger</td>
</tr>
<tr>
<td>allowed: bool Default: true</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>Type of image. 'f' means Float, 'c' means complex.</td>
</tr>
<tr>
<td>allowed: string Default: f</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

image

**Example**

```python
"""
# print "\t----\t newimagefromshape Ex 1 \t----"
im1=ia.newimagefromshape('test2.data', [64,64,128], overwrite=true)
cs1 = im1.coordsys(axes=[0,2])
im1.done()
im2 = ia.newimagefromshape(shape=[10, 20], csys=cs1.torecord())
cs1.done()
im2.done()
# """
```

325
The first example creates a zero-filled \texttt{imagefile} named \texttt{test.data} of shape [64,64,128]. If you examine the header with \texttt{ia.summary()} you will see the RA/DEC/Spectral coordinate information. In the second example, a Coordinate System describing the first and third axes of the image \texttt{test2.data} is created and used to create a 2D temporary image.
image.pbcor.html

**image.pbcor - Function**

Construct a primary beam corrected image from an image and a primary beam

**Description**

Correct an image for primary beam attenuation using an image of the primary beam pattern. The primary beam pattern can be provided as an image, in which case 1. it must have the same shape as the input image and its coordinate system must be the same, or 2. it must be a 2-D image in which case its coordinate system must consist of a (2-D) direction coordinate which is the same as the direction coordinate in the input image and its direction plane must be the same shape as that of the input image. Alternatively, pbimage can be an array of pixel values in which case the same dimensionality and shape constraints apply. An image tool referencing the corrected image is returned. The corrected image will also be written to disk if outfile is not empty (and overwrite=True if outfile already exists). One can choose between dividing the image by the primary beam pattern (mode=”divide”) or multiplying the image by the primary beam pattern (mode=”multiply”). One can also choose to specify a cutoff limit for the primary beam pattern. For mode=”divide”, for all pixels below this cutoff in the primary beam pattern, the output image will be masked. In the case of mode=”multiply”, all pixels in the output will be masked corresponding to pixels with values greater than the cutoff in the primary beam pattern. A negative value for cutoff means that no cutoff will be applied, which is the default.

**Arguments**
<table>
<thead>
<tr>
<th><strong>Inputs</strong></th>
<th><strong>Description</strong></th>
<th><strong>Default</strong></th>
<th><strong>Allowed</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>pbimage</td>
<td>Name of the primary beam image which must exist or array of values for the pb response.</td>
<td>variant</td>
<td>any</td>
</tr>
<tr>
<td>outfile</td>
<td>Output image name. If empty, no image is written.</td>
<td>False</td>
<td>string</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite the output if it exists?</td>
<td>False</td>
<td>bool</td>
</tr>
<tr>
<td>box</td>
<td>Rectangular region(s) to select in direction plane. See &quot;help par.box&quot; for details.</td>
<td>variant</td>
<td>string</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details.</td>
<td>variant</td>
<td>any</td>
</tr>
<tr>
<td>chans</td>
<td>Channels to use. See &quot;help par.chans&quot; for details.</td>
<td>variant</td>
<td>string</td>
</tr>
<tr>
<td>stokes</td>
<td>Stokes planes to use. See &quot;help par.stokes&quot; for details.</td>
<td>variant</td>
<td>string</td>
</tr>
<tr>
<td>mask</td>
<td>Mask to use. See help par.mask.</td>
<td>variant</td>
<td>string</td>
</tr>
<tr>
<td>mode</td>
<td>Divide or multiply the image by the primary beam image. Minimal match supported.</td>
<td>divide</td>
<td>string</td>
</tr>
<tr>
<td>cutoff</td>
<td>PB cutoff. If mode is &quot;d&quot;, all values less than this will be masked. If &quot;m&quot;, all values greater will be masked. Less than 0, no cutoff.</td>
<td>-1.0</td>
<td>float</td>
</tr>
<tr>
<td>stretch</td>
<td>Stretch the mask if necessary and possible?</td>
<td>False</td>
<td>bool</td>
</tr>
</tbody>
</table>


Returns
image

Example

```python
ia.open("attenuated.im")
pbia = ia.pbcor(pbimage="mypb.im", outname="pbcorred.im", mode="divide", cutoff=0.1)
ia.done()
# do stuff with ia tool attached to pb image and close it
pbia.done()
```
image.pv.html

**image.pv - Function**

1.1.1 Construct a position-velocity image between two points in the direction plane.

**Description**

Create a position-velocity image by specifying either two points between which a slice is taken in the direction coordinate or a center, position angle, and length describing the slice. The spectral extent of the resulting image will be that provided by the region specification or the entire spectral range of the input image if no region is specified. One may not specify a region in direction space; that is accomplished by specifying the start and end points or the center, length, and position angle of the slice. The parameters start and end may be specified as two element arrays of numerical values, in which case these values will be interpreted as pixel locations in the input image. Alternatively, they may be expressed as arrays of two strings each representing the direction. These strings can either represent quantities (e.g., "40.5deg", "0.5rad") or be sexagesimal format (e.g., "14h20m20.5s", "-30d45m25.4s"), 

("14:20:20.5s", "-30:45:25.4")]. In addition, they may be expressed as a single string containing the longitude and latitude values and optionally a reference frame value, eg "J2000 14:20:20.5s -30:45:25.4". The center parameter is specified in the same way. The length parameter may be specified as a single numerical value, in which case it is interpreted as the length in pixels, or a valid quantity, in which case it must have units conformant with the direction axes units. The pa (position angle) parameter must be specified as a valid quantity with angular units. The position angle is interpreted in the usual astronomical sense; ie measured from north through east. Either start/end or center/pa/length must be specified; if a parameter from one of these sets is specified, a parameter from the other set may not be specified. In either case, the end points of the segment must fall within the input image, and they both must be at least 2 pixels from the edge of the input image to facilitate rotation (see below).

One may specify a width, which is the number of pixels centered along and perpendicular to the direction slice that are used for averaging along the slice. The width may be specified as an integer, in which case it must be positive and odd. Alternatively, it may be specified as a valid quantity string (eg, "4arcsec") or quantity record (eg qa.quantity("4arcsec"). In this case, units must be conformant to the direction axes units (usually angular units) and the specified quantity will be rounded up, if necessary, to the next highest equivalent odd integer number of pixels. The default value of 1 represents no
averaging. A value of 3 means average one pixel on each side of the slice and the pixel on the slice. Note that this width is applied to pixels in the image after it has been rotated (see below for a description of the algorithm used). The end points of the specified segment must fail within the input image, and they both must be at least 2 pixels from the edge of the input image to facilitate rotation (see below).

One may specify the unit for the angular offset axis.

A true value for the wantreturn parameter indicates that an image analysis tool attached to the created image should be returned. Nothing is returned if wantreturn is false, but then outfile should be specified (unless perhaps you are debugging).

Internally, the image is first rotated, padding first if necessary to include relevant pixels that would otherwise be excluded by the rotation operation, so that the slice is horizontal, with the starting pixel left of the ending pixel. Then, the pixels within the specified width of the slice are averaged and the resulting image is written and/or returned. The output image has a linear coordinate in place of the direction coordinate of the input image, and the corresponding axis represents angular offset with the center pixel having a value of 0.

The equivalent coordinate system, with a (usually) rotated direction coordinate (eg, RA and Dec) is written to the output image as a table record. It can be retrieved using the table tool as shown in the example below.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image name. If empty, no image is written. Default &quot;&quot;</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>start</td>
<td>The starting point in the direction plane (array of two values). If specified, end must also be specified and none of center, pa, nor length may be specified.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>end</td>
<td>The ending point in the direction plane (array of two values). If specified, start must also be specified and none of center, pa, nor length may be specified.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>center</td>
<td>The center point in the direction plane (array of two values). If specified, length and pa must also be specified and neither of start nor end may be specified.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>length</td>
<td>The length of the segment in the direction plane. If specified, center and pa must also be specified and neither of start nor end may be specified.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>pa</td>
<td>The position angle of the segment in the direction plane, measured from north through east. If specified, center and length must also be specified and neither of start nor end may be specified.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>width</td>
<td>Width in pixels for averaging pixels perpendicular to the slice. Must be an odd integer &gt;= 1 (1 means only use the pixels along the slice), or a quantity which will be rounded up if necessary so it corresponds to the next highest odd number of pixels.</td>
<td>any</td>
<td>variant 1</td>
</tr>
<tr>
<td>unit</td>
<td>Unit for the offset axis in the resulting image. Must be a unit of angular measure.</td>
<td>string</td>
<td>arcsec</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite the output if it exists?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>region</td>
<td>Region selection. See &quot;help par.region&quot; for details. Default is to use the full image. No selection is permitted in the direction plane.</td>
<td>any</td>
<td>variant</td>
</tr>
</tbody>
</table>
Returns
image

Example

```plaintext
ia.open("my_spectral_cube.im")
# create a pv image with the position axis running from ra, dec pixel positions of [45, 50] to [100, 120] in the input image
mypv = ia.pv(outfile="pv.im", start=[45,50], end=[100,120], wantreturn=true)
ia.done()
# analyze the pv image, such as get statistics
pvstats = mypv.statistics()
# when done, close the tool to release system resources
mypv.done()

# get the alternate coordinate system information
tb.open("pv.im")
alternate_csyst_record = tb.getkeyword("misc")["secondary_coordinates"]
tb.done()
```
image.makearray.html

image.makearray - Function

1.1.1 Construct an initialized multi-dimensional array.

Description

This function takes two arguments. The first argument is the initial value for the new array. The second is a vector giving the lengths of the dimensions of the array.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>Value with which to initial array elements</td>
<td>double</td>
<td>0.0</td>
</tr>
<tr>
<td>shape</td>
<td>Vector containing array dimensions.</td>
<td>intArray</td>
<td>0</td>
</tr>
</tbody>
</table>

Returns

anyvariant

Example

A three dimensional array that is initialized to all zeros. Each of the three dimensions of the cube has a length of four.

```python
print "\t----\t makearray Ex 1 \t----"
cube = ia.makearray(0,[4,4,4])
```

"""
image.isconform.html

image.isconform - Function

1.1.1 Returns true of the shape, coordinate system, and axes order of the specified image matches this image.

Description

Returns True if the shape, coordinate system, and axes order of the specified image matches the current image.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>other</td>
<td>name of image to test</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

bool

Example

ia.isconform("my_mystery.image")
1.1.2 regionmanager - Tool

Create and manipulate regions of interest
Requires:

Synopsis

Description

Overview of Regionmanager functionality

- Create simple pixel-coordinate based regions with functions
  box
  quarter
  Create simple world-coordinate based regions with functions
  frombcs (range along one axis)
  wrange (range along one axis)
  wbox
  wpolygon
  wdbbox
  wpolygon
  Also related is function setcoordinates
- Convert pixel regions to world regions with
  pixeltoworldregion.
- There are some general utility functions, generally only of interest if you are writing scripts. These are
  absreltype
  def
  done
  extractsimpleregions
  ispixelregion
  isworldregion
• There are some functions relating to interactive creation of regions with the Viewer. These are generally only of interest if you are writing scripts, and are
  
displayedplane
psuedotoworldregion

• There are some functions relating to communications. These are generally only of interest if you are writing scripts and are
  
setselectcallback
getselectcallback

Regions and the Regionmanager
When working with an image, one is generally interested in some part of that image on which astrophysical analysis is performed. This region-of-interest (or more generically and simply, the ‘region’) might be the whole image, or some subset of it.
Regions come in a few varieties. There are simple regular shapes (box, ellipsoid), simple irregular shapes (polygon), as well as compound regions made by combining other regions (simple or compound). For example unions or intersections of regions. In addition, the simple regions can be pixel-coordinate based or world-coordinate based. However, a compound regions must always comprise world-coordinate based regions.
It is the task of the Regionmanager (tool) to manage your regions; it creates, lists and manipulates them. Apart from a Regionmanager, the only other way to create a region-of-interest is with the viewer tool (type viewer within the casapy environment). This allows you to interactively make, with the cursor and an image display, a box or polygonal region. The region so made may be collected by the Regionmanager (in fact the complete process can be initiated from the Regionmanager).
The Regionmanager has a command line interface, but there are plans to have it interact directly with the CASA viewer tool in the future. Currently the only way to interact is to save the regions created with the viewer to a file or as a region in the image, and use the Regionmanager function fromfiletorecord or fromtabletorecord, respectively, to bring the regions in the CLI.
It is probably fair to say that for the simplest regions, such as a pixel box, there is little to choose between making a region with the GUI or the command line interface. However, for the world regions, the GUI is significantly better; it hides the details of handling the coordinate information.

Simple Pixel and World regions
Pixel regions are specified purely in terms of pixel coordinates. Pixel coordinates are considered to run from 1 at the bottom left corner (blc) of an image to the image shape at the top-right corner (trc).
For example, a pixel region might be a box that runs from a bottom blc of [20,20] to a trc of [64,90] in a 2D image. Such a pixel region is NOT very portable though. If you are interested in a region about an astronomical source, pixel coordinates are only useful for a particular image. In some other image, the source may well have different pixel coordinates. However, pixel regions are easy to make and use, and they are valuable.

So far, we have been talking about absolute pixel coordinates. However, pixel regions can also be specified in relative coordinates. This is controlled through the argument `abserel` which may take on one of the values ‘abs’ (absolute coordinates) ‘relref’ (relative to the reference pixel of the image) or ‘relcen’ (relative to the center of the image). You can use the summary function of the image module to see the reference pixel. The sense of the offset is $rel = abs - ref$.

You may also define pixel regions in terms of fractional coordinates that are in the range [0,1] (blc to trc) - look for argument `frac` which can be T or F.

From the command line, let’s make a simple pixel box region with
- `ia.open('myimage')` - `csys = ia.coordsys()` - `r1 = rg.box(blc=[10,20], trc=[30,40])` - `ia.statistics(region=r1)`; Number points = 441 Flux density = -1.728895e-03 Jy Sum = -9.522969e-02 Mean = -2.159403e-04 Variance = 1.518106e-05 Sigma = 3.896288e-03 Rms = 3.897854e-03

Minimum value is -1.208747e-02 at [13, 32] Maximum value is 1.287862e-02 at [21, 37]

You have now created a region tool called `r1` which describes a 2D box from [10,20] to [30,40] in absolute pixel coordinates. The region is made ‘const’ as well so you can’t overwrite it by mistake. We then apply it to an image and evaluate some statistics in the specified region.

World regions are more portable, because the region is specified in terms of the world coordinates (e.g. RA and DEC, or Frequency etc). You can create a region with one image, and then apply it to another. For example, you might make an RA/DEC world box (i.e. blc and trc specified as RA and DEC) from an optical image. You can then apply it to your radio image of the same source. The software will look for the axes of the region (in this case RA and DEC), and then attempt to match them with the image to which the region is being applied. For each matching axis, the region will be applied.

When you make a world region, you must supply a coordinate system from the appropriate image to the Regionmanager. The coordinate system comes from the coordsys function of the Image tool. This associates a coordinate with an image pixel axis.

Now let’s do an example with a world box. It is better to make a world box with the GUI interface, but the command line is still manageable. First, let’s summarise the header of our image (the one from the previous example).
- `ia.open('myimage')` - `ia.summary()` Image name : myimage Image mask : Absent Image units : JY/BEAM Direction system : J2000 Name Proj Shape Tile Coord value at pixel Coord incr Units

338
And now we will make a world box with the blc at the reference pixel, and the trc offset somewhat. We use the quanta module for creating associations of values and units; in particular, the quantity function.

```python
- csys = ia.coordsys() - r2 = rg.wbox(blc=blc, trc=trc, pixelaxes=[0,1],
  csys=csys.torecord()) - ia.boundingbox(r2); [blc=[90 90], trc=[103 98], inc=[1
1], bbShape=[14 9], regionShape=[14 9], imageShape=[155 178]]
```

As well as passing in the blc and trc quantities, we have also told the Regionmanager which axes these values pertain to (they are numbered in the order in which they list in the summary function), and we also supplied it with the coordinate system of the image. Note that with the GUI interface, both the axes and coordinates handling is hidden. If we wished, we could have left out the pixelaxes argument, and it would have defaulted to [0,1]. We then find the bounding box of the region when it’s applied to the image. You can see from the summary listing that it is correct.

We generally think about our images as being in some order;
RA/DEC/Frequency (for the X/Y/Z axes) or whatever. In the application of world regions, this order is unimportant. If you created a world region from an RA/DEC/Frequency image, and applied it to a Frequency/DEC/RA image, that image order change would be accounted for.

Note that for pixel regions however, because the region is not tagged with a coordinate system, the order of the image is relevant. So if you made a pixel box from an RA/DEC image and applied it to a DEC/RA image, the RA range would be applied to the DEC axis and vice versa. The pixel regions are present for simple usage only.

**Simple pixel regions as world regions**

It is possible to create a world region that is specified in pixel coordinates, and maintains the coordinate axis information. This can be useful because compound regions must be world-coordinate based (see below). We can do this because we have defined in the Regionmanager, “new” world units, called “pix” (which are then known to the quanta module) and these are recognized in the world regions. Similarly, we have also defined a unit called “frac”. This allows you to specify world coordinates in units of the fraction of the image shape. For example, the blc of a 2D image has a “frac” coordinate of [0,0] and a trc of [1,1]. But be careful, such regions are still not portable to another image, because they are still effectively pixel-coordinate based (although masquerading as world regions).

```python
- ia.maketestimage() - csys = ia.coordsys() - r1 =
  rg.wbox(blc="10pix,10pix", trc="20pix,20pix", pixelaxes=[0,1],
  csys=csys.torecord()) - localstats = ia.statistics(region=r1, list=False,
  verbose=False) - print 'Region 1: number points = ', localstats['npts'][0]
Region 1: number points = 121 - - r2 = rg.wbox(blc="30pix 30pix",
  trc="40pix 40pix", pixelaxes=[0,1], csys=csys.torecord()) -
```

339
localstats=ia.statistics(region=r2, list=False, verbose=False) - print 'Region 2:
number points = ', localstats['npts'][0] Region 2: number points = 121
- regions= - regions[r1]=[r1] - regions[r2]=[r2] - r3 = rg.makeunion(regions)
- localstats=ia.statistics(region=r3, list=False, verbose=False) - print 'Region
3: number points = ', localstats['npts'][0] Region 3: number points = 242

In this example we make a union from two pixel boxes masquerading as world
regions.

Compound Regions
It is often desirable to make a region which combines others. These are called
compound regions. For example, give me the union of 3 regions and evaluate
the statistics in that union. Or intersect this region with that one and extract
the image data from that region. Compound regions are fully recursive; you
can nest them as deeply as you like.

You must be aware that compound regions can only be made from
world-coordinated based regions. You can convert a pixel region to a world
region with the pixeltoworldregion function (at some point this function will
move from the Regionmanager tool to the Image tool.

- ia.open('myimage') - csys ia.coordsys(); - - blc = "17:42:29.303,
rg.wbox(blc=blc,trc=trc,pixelaxes=[0,1],csys=csys.torecord()); -
- ia.boundingbox(r1); [blc=[90 90] , trc=[103 98] , inc=[1 1], bbShape=[14 9] ,
regionShape=[15 178] ] - - blc = "17:42:27.303,
rg.wbox(blc=blc,trc=trc,pixelaxes=[0,1],csys=csys.torecord()); -
- ia.boundingbox(r2); [blc=[116 90] , trc=[129 98] , inc=[1 1], bbShape=[14 9] ,
regionShape=[15 178] ] - - regions= - regions[r1]=[r1] -
regions[r2]=[r2] - r3 = rg.makeunion(regions) - bb = ia.boundingbox(r3) - bb
[blc=[90 90] , trc=[129 98] , regionShape=[40 9] , imageShape=[155 178] ] - -
s1 = ia.statistics(region=r1, list=False) - s2 = ia.statistics(region=r2, list=F)
- s3 = ia.statistics(region=r3, list=F) - - np =
(2bb[trc][0]-bb[blc][0]+1)*(2bb[trc][1]-bb[blc][1]+1) - print 'No. pts in r1,
r2, r3, bb(r3) = ', s1['npts'][0], s2['npts'][0], s3['npts'][0], np No. pts in r1, r2,
r3, bb(r3) = 126, 126, 252, 360 - - r4 = rg.intersection(r1, r2) -
- ia.boundingbox(r4) - r4
We make two world boxes and find the bounding box of each. Then we make
the union of the two and find the bounding box which we store in a record
called bb. We then find some statistics over the union (supressing the logger
output and storing the results in records). You can see that the number of
points found in the region reflects the union, not the bounding box of the
union.

Finally, we make an intersection of our two regions. Because regions r1 and r2
don't actually intersect, so the region returned is empty.

Automatic Extension
One general philosophy behind the regions software is that if you apply a
region to an image, but don't explicitly specify all axes of the image in that
region, then for the unspecified axes, the full image is taken. For example, imagine that from an RA/DEC optical image you made an RA/DEC region. You then applied it to an RA/DEC/Frequency radio spectral-line cube. For the frequency axis, which was not specified in the region, all frequency pixels would be selected. Another philosophy is that if, on application, a region extends beyond an image, the overhang is silently discarded.

```python
- ia.open('myimage') - ia.boundingbox(rg.box()) [blc=[1 1], trc=[155 178], inc=[1 1], bbShape=[155 178], regionShape=[155 178], imageShape=[155 178])

- r1 = rg.box(trc=[20]) - ia.boundingbox(r1) [blc=[1 1], trc=[20 178], inc=[1 1], bbShape=[20 178], regionShape=[20 178], imageShape=[155 178])

- r2 = rg.box(trc=[9000,20]) - ia.boundingbox(r2) [blc=[1 1], trc=[155 20], inc=[1 1], bbShape=[155 20], regionShape=[155 20], imageShape=[155 178])
```

In the first example, none of the axes are specified in the region, so it defaults to the full image on all axes. In the second example only the first axis of the trc is specified. In the third example, the trc overhang is silently discarded.

**Masks**

Some comment about the combination of masks and regions-of-interest is useful here. Consider a simple polygonal region. This region-of-interest is defined by a bounding box, the polygonal vertices, and a mask called a region mask. The region mask specifies whether a pixel within the bounding box is inside or outside the polygon. For a simple box region-of-interest, there is obviously no need for a region mask.

Now imagine that you wish to recover the mask of an image from a polygonal region-of-interest. Now, necessarily, the mask is returned to you in regular Boolean array. Thus, the array shape reflects the bounding-box of the polygonal region. If the actual pixel mask that you apply is all good, then the retrieved mask would be good inside of the polygonal region and bad outside of it. If the actual pixel mask had some bad values in it as well, the retrieved mask would be bad outside of the polygonal region. Inside the polygonal region it would be bad if the pixel mask was bad.

More simply put, the mask that you recover is just a logical “and” of the pixel mask and the region mask; if the pixel mask is T and the region mask is T then the retrieved mask is T (good), else it is F (bad).

**Vector Inputs**

Many of the functions of a Regionmanager tool take vectors (numeric or strings) as their arguments. You can assume that

- For numeric vectors you can enter as an actual numeric vector (e.g. [1.2, 2.5, 3]), a vector of strings (e.g. ”1.2 2.5 3”) or even a string with white space and/or comma delimiters (e.g. ’1.2 2.5 3’).

- For string vectors you can enter as an actual vector of strings (e.g. ”1.2 2.5 3”) or a string with white space or comma delimiters (e.g. ’1.2 2.5 3’).

**Default Values**
When specifying blc and trc vectors for some regions (usually boxes) it can often be that you wish to default one axis but not others. For example, you may like to specify a value for axis 2 but leave axis 1 at its defaults. This is trivial with the GUI interface (you just don’t fill in the ones you are not interested in), but is a little harder with the command line interface.

There is a function called def which provides a magic value which can be used with pixel boxes for this purpose. For example,

```
- ia.fromshape('x', [20,30]) - r = rg.box(trc=[rg.dflt(),10]) -
ia.boundingbox(region=r) [blc=[1 1] , trc=[20 10] , regionShape=[20 10] ,
imageShape=[20 30] ]
```

You can see that trc for axis 1 has defaulted to the shape of the image.

With world boxes, we have defined a magic unit called `default` instead. Thus

```
- ia.fromshape('x', [20,30]) - csys = ia.coordsys() -
rg.setcoordinates(csys.torecord()) - r = rg.wbox(trc="0dflt 10pix") -
ia.boundingbox(region=r) [blc=[1 1] , trc=[20 10] , regionShape=[20 10] ,
imageShape=[20 30] ]
```

**Error Checking**

Although some error checking is done when the region is created (in CASA), much of it does not occur until the region is applied to an image. In particular, when creating a compound region, it is not checked that the compound region contains only world-coordinate based regions. It is only when you apply the region to an image that these checks are made. Any errors that occur will cause exceptions to be generated. Thus, when writing scripts, you should always use try/except blocks as in this example.

```
- def getStats(imageobject, blc, trc, csys): try : r1 = rg.wbox(blc=blc,
trc=trc, csys=csys.torecord()) imageobject.statistics(r1) except Exception,
instance: print 'ERROR: ", instance return False
return True
```

This fairly silly function detects if the region creation failed (function given invalid arguments) and displays the exception if it did. If all is well statistics are then evaluated from the region, and True is returned.

**How a region is stored in CASA**

In CASA, a region is itself a tool. This means it can be transmitted about with Glish, saved to Tables and restored from Tables. A region is actually made with a generic container tool called an Itemcontainer. The regions that you make have no intelligence regarding their greater purpose in life (although of course an Itemcontainer does have some functions so they can be manipulated). All of the knowledge about regions lies with the Regionmanager. It should not be necessary for you to know anything about the insides of regions. You just need to know how to make them and how to use them.

THIS EXAMPLE IS NOT VALID ANYMORE

342
First we make a simple pixel-box region. We then try and print it out. You can see that this isn’t very enlightening. What you see are the Itemcontainer’s functions. To really see inside the region, you can use the torecord function of Itemcontainer. This converts the region to a record which you can view if you really must (the record is of no functional use when dealing with regions).

It’s perhaps also useful to know that once you have made a region you can’t break it. You can delete it, maybe overwrite it, but you can’t mistakenly mess up its contents (even with the Itemcontainer functions with which it was created). It’s user proof - there’s a challenge for you!

Methods

- regionmanager: Construct a regionmanager
- absreltype: Convert region type value to a string
- box: Create a pixel box region
- frombcs: Create a world coordinate region based on box-chan-stokes input
- complement: Create the complement of a world region
- concatenation: Concatenate world regions along a new axis
- deletefromtable: Delete regions from a Table
- difference: Create the difference of two world regions
- done: Destroy this regionmanager
- selectedchannels: Get an array of zero-based selected channel numbers from an input string specificaiton.
- fromtextfile: Create a region dictionary from a region text file.
- fromtext: Create a region dictionary from a region text string.
- fromfiletorecord: Create a region record(s) from a file(s).
- tofile: Create a region record file that can be read by from filetorecord.
- fromrecordtotable: Save regions stored in a record into a Table
- fromtabletorecord: Restore regions from a Table to a record
- intersection: Create the intersection of some world regions
- ispixelregion: Is this region a pixel region?
- isworldregion: Is this region a world region?
- namesintable: Find the names of the regions stored in a Table
- setcoordinates: Set new default Coordinate System
- makeunion: Create a union of world regions
- wbox: Create a world box region
- wpolygon: Create a world polygon region with quantities
Construct a regionmanager

Description

This is the only Regionmanager constructor. It should generally be unnecessary for you to make one as there is little state in a Regionmanager (you can set a Coordinate System with setcoordinates); the default Regionmanager `rg` should be all you need.

Arguments

Returns
casaregionmanager
Function

1.1.2 Convert region type value to a string

Description

This function is not intended for general user use. Regions may be specified with coordinates which are absolute or relative. This function converts the integer code defining the absolute/relative type of the coordinates (which is stored in the region) into a string (maybe for printing purposes).

The different types are:

- Integer String Description
  - 1 abs Absolute coordinate
  - 2 relref Relative reference pixel
  - 3 relcen Relative to center of image
  - 4 reldir Relative to some direction

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Region type value</th>
</tr>
</thead>
<tbody>
<tr>
<td>absrefvalue</td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
<tr>
<td></td>
<td>Integer - 1, 2, 3, 4</td>
</tr>
</tbody>
</table>

Returns

string

Example

```python
- r = rg.box(blc=[3,40], trc=[80,90])  # Create region
- v = r.get('arblc')                   # Get absrel value vector for blc
- for i in range(len(v)):
  + print rg.absreltype(v[i])         # Print string conversion for each axis
```
regionmanager.box.html

regionmanager.box - Function

1.1.2 Create a pixel box region

Description

This function creates a multi-dimensional pixel box region. The box is specified by a bottom-left corner, and top-right corner and an increment (or stride). Pixel coordinates are considered to run from 1 at the bottom left corner of the image to the image shape at the top-right corner of the image. You can specify whether the coordinates are given as pixel coordinates \((\text{frac}=F)\) or fractions of the image shape \((\text{frac}=T)\). Absolute fractions are in the range \([0,1]\).

You can also specify whether the coordinates are given as absolute coordinates \((\text{absrel}=\text{'abs'})\) or relative to the reference pixel \((\text{absrel}=\text{'relref'})\) or relative to the center of the image \((\text{absrel}=\text{'relcen'})\).

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>blc</td>
<td>doubleArray</td>
<td>0</td>
</tr>
<tr>
<td>trc</td>
<td>doubleArray</td>
<td>-1</td>
</tr>
<tr>
<td>inc</td>
<td>doubleArray</td>
<td>1</td>
</tr>
<tr>
<td>absrel</td>
<td>string</td>
<td>relref</td>
</tr>
<tr>
<td>frac</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>comment</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Shape</th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Returns | record |

| Example |

```plaintext
ia.open('myimage')
ia.shape()
[155 178 256]

r = rg.box()       # create region
- ia.boundingbox(r)
```
This region, on application to an image, selects the entire image.

Example

- ia.open('myimage')
- ia.shape()
  [155 178 256]
-
- r=rg.box(blc=[5,10])
- ia.boundingbox(r)
  [blc=[5 10 1] , trc=[155 178 256] , inc=[1 1 1] , bbShape=[151 169 256] ,
regionShape=[151 169 256] , imageShape=[155 178 256] ]

This region is only specified for the first two axes of the blc. Automatic extension rules apply for the other axis and the trc (defaults to the shape).

Example

- ia.open('myimage')
- ia.shape()
  [155 178 256]
-
- r=rg.box(blc=[10, 10, 10], trc=[20, 20, 20], inc=[2, 2, 2])
- ia.boundingbox(r)
regionShape=[6 6 6] , imageShape=[155 178 256] ]
-
stats=ia.statistics(region=r, list=False);
stats['npts'][0]
  216
This region picks out every other pixel in the 3D box. The ‘regionShape’ field of the bounding box record does reflect the increment whereas ‘bbShape’ does not. You can see that the number of points used in determining the statistics (216) reflects the increment as well.

Example

THIS EXAMPLE IS NOT VALID YET

- ia.open('myimage')
- ia.shape()
  [64 128]
- rmd = rg.dflt()
- r = rg.box([-5,-10], [rmd,20], absrel='relcen')
- ia.boundingbox(r)
  [blc=[28 55] , trc=[64 85] , inc=[1 1] , bbShape=[37 31] ,
  regionShape=[37 31] , imageShape=[64 128] ]

The region is specified in pixels relative to the center of the image. Note the use of the default value ({\cf rg.dflt()}) to default the first axis of the trc argument to the image shape without having to know the image shape.

Example

- ia.open('myimage')
- ia.shape()
  [155 178 256]
This example shows selection by relative to reference pixel fractional coordinates plus auto extension to unspecified axes.
regionmanager.frombcs.html

regionmanager.frombcs - Function

Create a world coordinate region based on box-chan-stokes input

Description

This function creates a multi-dimensional world coordinate region based on box, chans, stokes inputs familiar from image analysis tasks. It is being introduced as a temporary means of refactoring some python level task code into C++. However, if users find it to have value, its existence can be permanent.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Coordinate system record. Must be specified.</th>
</tr>
</thead>
<tbody>
<tr>
<td>csys</td>
<td>allowed: record</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>shape</td>
<td>shape of the image. Necessary for boundedness checks. Must have the same number of dimensions as the associated coordinate system. Default = []</td>
</tr>
<tr>
<td></td>
<td>allowed: intArray</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
<tr>
<td>box</td>
<td>Direction plane box specification as normally provided in image analysis tasks. &quot;&quot;&quot; means use entire directional plane as specified in shape. Default &quot;&quot;&quot;.</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>chans</td>
<td>Channel spec as normally provided to image analysis tasks. &quot;&quot;&quot; means use all channels, Default &quot;&quot;&quot;.</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>stokes</td>
<td>Stokes spec as normally provided to image analysis tasks. &quot;&quot;&quot; means use stokescontrol for setting stokes. Default &quot;&quot;&quot;.</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>stokescontrol</td>
<td>Polarization set to use if stokes parameter is not specified. Choices are &quot;a&quot; (use all stokes) and &quot;f&quot; (use first stokes). Default &quot;a&quot;.</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: a</td>
</tr>
<tr>
<td>region</td>
<td>Named region in the form imagename:regionname or region dictionary. Used only if box, chans, stokes not specified. Default &quot;&quot;&quot;.</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
</tbody>
</table>

**Returns**
record

**Example**
**regionmanager.complement - Function**

1.1.2 Create the complement of a world region

**Description**

This function (short-hand name *comp*) creates the complement of a world region(s).

The region parameter can be a single region record defining a simple or complex region or it can contain several region records in a Python dictionary. If multiple regions are given then the union of this set of regions is taken first, and the complement is found from the union.

NOTE: *ia.statistics()* is UNABLE to handle complement regions in CASA yet.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>region</td>
<td>The world region</td>
<td>any</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>variant</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td>Region tool</td>
</tr>
<tr>
<td>comment</td>
<td>A comment stored with the region</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td>String</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
- ia.open('hcn')
- csys = ia.coordsys()
- ia.shape()
[155 178]
```
As expected, the number of pixels in the complement is $(155 \times 178) - 126 = 27464$
regionmanager.concatenation.html

**regionmanager.concatenation - Function**

1.1.2 Concatenate world regions along a new axis

Description

This function (short-hand name **concat**) creates a region which is the concatenation along a new axis of the given world regions. This function is similar to the extension function. The **concatenation** function allows you to take many world regions, and concatenate them along one axis (rather than take one region and extend it along many axes which is what function **extension** does).

For example, you may have generated a different polygonal region for each spectral pixel of a spectral-line cube and you wish to concatenate them together to form the overall region for use in a deconvolution application. The axis to concatenate along is specified as a 1-dimensional world box. The shape of the 1D box must contain as many pixels (although you don’t have to specify it in pixels) as there are regions to concatenate.

Because this function is most likely to be used in a script, the interface takes a record containing **region** records, Python dictionaries, as there might be a lot of them.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>box</td>
<td>The axis to concatenate along</td>
</tr>
<tr>
<td>regions</td>
<td>World regions</td>
</tr>
<tr>
<td>comment</td>
<td>A comment stored with the region</td>
</tr>
</tbody>
</table>

| allowed: | any |
| Default: | variant |
| world box region |
| Record containing world regions |
| string |
| String |

Returns

record
Example

```python
- ia.open('cube')
- csys = ia.coordsys()
- rg.setcoordinates(csys.torecord(), verbose=False) # Don't tell us each time
  # private coordinates used
- box = rg.wbox(blc="20pix", trc="25pix", pixelaxes=[2])
- bb = ia.boundingbox(box)
-
- regs = {};
- local x, y;
- for i in bb.blc[3]:bb.trc[3]:
  + # Some code in function 'mypolygon' generates the
  + # x and y vectors for this spectral pixel, perhaps interactively
  +
  + mypolygon(x,y);
  +  regs["reg"+str(j)] = rg.wpolygon(x,y,[0,1])
- rc = rg.concatenation(box, regs)
-
- ia.statistics(region=rc, axes=[1,2])

<table>
<thead>
<tr>
<th>Plane</th>
<th>Freq</th>
<th>Npts</th>
<th>Sum</th>
<th>Mean</th>
<th>Rms</th>
<th>Sigma</th>
<th>Minimum</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>1.413724e+09</td>
<td>25</td>
<td>-4.778154e+00</td>
<td>-1.911262e-01</td>
<td>2.578399e-01</td>
<td>1.766359e-01</td>
<td>-4.252437e-01</td>
</tr>
<tr>
<td>21</td>
<td>1.413744e+09</td>
<td>40</td>
<td>-7.476902e+00</td>
<td>-2.990761e-01</td>
<td>3.692736e-01</td>
<td>2.210687e-01</td>
<td>-6.073643e-01</td>
</tr>
<tr>
<td>22</td>
<td>1.413763e+09</td>
<td>32</td>
<td>-2.696485e+00</td>
<td>-1.078594e-01</td>
<td>1.916686e-01</td>
<td>1.617070e-01</td>
<td>-3.295762e-01</td>
</tr>
<tr>
<td>23</td>
<td>1.413783e+09</td>
<td>77</td>
<td>4.889158e+00</td>
<td>1.955663e-01</td>
<td>3.148451e-02</td>
<td>2.518293e-02</td>
<td>-3.671944e-02</td>
</tr>
<tr>
<td>24</td>
<td>1.413803e+09</td>
<td>25</td>
<td>-1.337832e+00</td>
<td>-5.351327e-02</td>
<td>6.296212e-02</td>
<td>3.858393e-02</td>
<td>-1.232499e-02</td>
</tr>
<tr>
<td>25</td>
<td>1.413823e+09</td>
<td>15</td>
<td>1.091297e+00</td>
<td>4.365189e-02</td>
<td>7.252339e-02</td>
<td>5.910932e-02</td>
<td>-6.364562e-02</td>
</tr>
</tbody>
</table>
```

In this example, we create a 1D box and use it to concatenate 2D xy polygons along the z axis. We then ask for the statistics of each plane in the region. There is a different number of pixels per plane as each polygon is different.
regionmanager.deletefromtable.html

regionmanager.deletefromtable - Function

1.1.2 Delete regions from a Table

Description

This function deletes a region stored in an casa Table.
For the tablename argument, you have to give the name of an existing CASA table on disk (any kind of table).
You specify the name of the region with the regionname arguments. If you set regionname='' then nothing is done. The names of all the regions stored in a Table can be found with the function namesintable.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablename</td>
<td>The table</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>Image tool, table tool or String</td>
</tr>
<tr>
<td>regionname</td>
<td>Name(s) of the region(s) to delete</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>Vector of strings</td>
</tr>
</tbody>
</table>

Returns

bool

Example

- names = rg.namesintable(hcn)
- rg.deletefromtable(img, names[0])

In this example, we delete the first region that is reported to be in the Table \{tt 'hcn'}.
regionmanager.difference.html

**regionmanager.difference - Function**

Create the difference of two world regions

**Description**

This function (short-hand name `diff`) creates a region which is the difference of two world regions. The order of the regions is important. The difference consists of all pixels masked-on in the first region and not masked-on in the second region.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>region1</td>
<td>The first world region</td>
</tr>
<tr>
<td>allowed:</td>
<td>record</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>region2</td>
<td>The second world region</td>
</tr>
<tr>
<td>allowed:</td>
<td>record</td>
</tr>
<tr>
<td>Default:</td>
<td>Region tool</td>
</tr>
<tr>
<td>comment</td>
<td>A comment stored with the region</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>String</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

- ia.open('hcn')
- csyz = ia.coordsys()
- rg.setcoordinates(csys.torecord())
- blc = "10pix 10pix"
- trc = "60pix 60pix"
We use pixel units and boxes in this example to make it clear what is happening. The two regions overlap in the top right corner area of region \( r1 \) by an area of \( 11 \times 11 = 121 \) pixels. Therefore, the difference region \( r3 \) has \( 2601 - 121 = 2480 \) pixels in it. For difference region \( r4 \), the region of overlap is the bottom left corner area of region \( r2 \) and still contains 121 pixels. We expect \( 961 - 121 = 840 \) pixels in the difference region.
regionmanager.done.html

**regionmanager.done - Function**

Destroy this regionmanager

**Description**

This function destroys the contents of the `regionmanager` tool (including its GUI). The tool still exists as a Glish variable, but it is no longer a Regionmanager! You are unlikely to need this function.

**Arguments**

**Returns**

`bool`
regionmanager.selectedchannels.html

regionmanager.selectedchannels - Function

Get an array of zero-based selected channel numbers from an input string specification.

Description

This method returns all the selected zero-based channel numbers from the specified string within the image.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>specification</td>
<td>Valid channel specification. See help par.chans for examples.</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>shape</td>
<td>Image shape. Used to determine if the specification lies outside the image.</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>intArray</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Returns

intArray

Example

```python
ia.fromshape("", [20, 20, 20])
rg.setcoordinates(ia.coordsys().torecord())
selected_channels = rg.selectedchannels(specification="range=[40km/s,50km/s]", shape=ia.shape())
ia.done()
```
regionmanager.fromtextfile.html

regionmanager.fromtextfile - Function

1.1.2 Create a region dictionary from a region text file.

Description

This function reads a text file containing region descriptions and converts it to a python dictionary.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
</table>
| filename | List of text file containing the region description  
| allowed: string  
| Default: |  
| shape | Image shape.  
| allowed: intArray  
| Default: 0  
| csys | Coordinate system record. Defaults to coordinate system used in rg.setcoordinates()  
| allowed: record  
| Default: |  

Returns

record
regionmanager.fromtext.html

**regionmanager.fromtext - Function**

Create a region dictionary from a region text string.

### Description

This function reads a region region text descriptions and converts it to a python region dictionary.

### Arguments

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>region description</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>shape</td>
<td>Image shape, only used if first region is a difference.</td>
<td>intArray</td>
<td>1</td>
</tr>
<tr>
<td>csys</td>
<td>Coordinate system record. Defaults to coordinate system used in rg.setcoordinates()</td>
<td>record</td>
<td></td>
</tr>
</tbody>
</table>

### Returns

- record

### Example

```python
ia.open('test.image')
csys=ia.coordsys()
rg.setcoordinates(csys.torecord())
a=rg.fromtext("ellipse [[04h31m38.44139, 18d13m57.0861], [1.0arcsec, 1.0arcsec], 0.00000000deg]", shape=[1500, 1500, 1, 1])
ia.done()
```

In this example, we create a circular region of 1 arcsec radius centered on J2000 04h31m38.44139 18d13m57.0861.
regionmanager.fromfiletorecord.html

regionmanager.fromfiletorecord - Function

Create a region record(s) from a file(s).

Description

This function reads files containing ImageRegion objects and turns them into Region Records.
The intended use for this method is to read the file saved by the casa viewer and turn the files contents into regions that are usable by the image analysis tool.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>List of files containing the Image Regions</td>
<td>string</td>
<td>File name(s)</td>
</tr>
<tr>
<td>verbose</td>
<td>Report successful saves</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>regionname</td>
<td>Name(s) of the region(s) when saved in the table</td>
<td>string</td>
<td>Self naming</td>
</tr>
</tbody>
</table>

Returns

record

Example

- img = ia.open('hcn')
- rg.fromfiletorecord(T, "x1 x2", "file1, file2", r1, r2)
- ia.statistics( region=r1, verbose=True )
- ia.statistics( region=r2, verbose=True )
In this example, we create two regions called \texttt{r1} and \texttt{r2} from the files ??? The regions are renamed to ‘x1’ and ‘x2’ as they are stored.

Example

e
- img = ia.open('hcn')
- r1 = rg.box()
- r2 = rg.quarter()
- rg.fromglobaltotable(img, T, F, "", r1, r2)
- rg.namesintable(img)
  x1 x2

In this example, we save two regions called \texttt{r1} and \texttt{r2} to the table (previously containing no regions) referred to by the image tool \texttt{im}. The names for regions are made up for us as we don’t specify them. Note that because the regions are specified by the special \texttt{...} argument (it has no actual argument name), we must give the \texttt{regionname} argument explicitly as an empty vector of strings (else \texttt{glish} will take the empty string as a region).
regionmanager.tofile.html

**regionmanager.tofile** - Function

Create a region record file that can be read by from filetorecord.

**Description**

This function is to store a region created by the regionmanager in a disk file for future use.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>List of files containing the Image Regions</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>File name(s)</td>
</tr>
<tr>
<td>region</td>
<td>region record/dict to store in the file</td>
</tr>
<tr>
<td>allowed:</td>
<td>record</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
- img = ia.open('hcn')
- imcs=ia.coordsys()
- blc = ['16:28:25.50', '+040.49.05.61']
- trc = ['16:24:28.67', '+041.45.28.43']
- r1 = rg.wbox(blcl=blc, trc=trc, pixelaxes=[0,1], csys=imcs.torecord())
- rg.tofile('myboxregion', r1)
- r1readback=rg.fromfiletorecord('myboxregion')
```

In this example we create a box region using world coordinates for blc and trc. We save that region to a file called `myboxregion`. Then we read it back using the function `rg.fromfiletorecord` and store it in a variable `r1readback`.
identical.
regionmanager.fromrecordtotable - Function

Save regions stored in a record into a Table

Description

This function saves regions into an casa Table. For the `tablename` argument the user should be the name of an existing CASA Table on disk (any kind of table). If the parameter `asmask` is True then the table has to be an image table. A mask makes sense with an image only.

You can specify the name the region will have (regionname) when it is saved in the Table. If you don’t specify this, a digit based name is assigned to it or if specify a name that already exists a new one will be generated which is close but different. The function returns you the name the region is assigned.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablename</td>
<td>string</td>
<td>Image tool, table tool or String</td>
</tr>
<tr>
<td>regionname</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>regionrec</td>
<td>record</td>
<td>Record of region tool(s)</td>
</tr>
<tr>
<td>asmask</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>verbose</td>
<td>bool</td>
<td>true</td>
</tr>
</tbody>
</table>

Returns

string
Example

- ia.open('hcn')
- cs=ia.coordsys()
- blc = "16:28:25.50 +040.49.05.61"
- trc = "16:24:28.67 +041.45.28.43"
- r1 = rg.wbox(blc="10pix 20pix",trc="30pix 40pix",pixelaxes=[0,1],csys=cs.torecord())
- r2 = rg.wbox(blc=blc,trc=trc,pixelaxes=[0,1],csys=cs.torecord())
- rg.fromrecordtotable('hcn', "x", r1)
  x
- rg.fromrecordtotable('hcn', "x", r2)
  x0
- rg.namesintable('hcn')
  x x0

Example

2 CASA image files on disk 'hcn1' 'hcn2'

- names = rg.namesintable('hcn1')
- r = rg.fromtabletorecord('hcn1', names[0])
- rg.namesintable('hcn2')
- rg.fromrecordtotable('hcn2', names[0], r)

In this example, we recover a region into a record from one image, and then copy them to another.

Example
### In this example a region is saved as a mask
ia.open('myfancy.image')
csys=ia.coordsys()
ia.done()
## Lets make a world-box region
wbox=rg.wbox([‘10pix’, ‘10pix’, ‘0pix’, ‘0pix’], [‘20pix’, ‘20pix’, ‘0pix’, ‘0pix’], csys=csys.torecord())
### save that into the image as a mask rather than just a region and assign it the name mask1
rg.fromrecordtotable('myfancy.image', 'mask1', wbox, asmask=True)
### now let us set that as default mask
ia.open('myfancy.image')
ia.maskhandler(‘set’, ‘mask1’)
ia.done()
### and now let us view that image
viewer(‘myfancy.image’)
regionmanager.fromtabletorecord.html

**regionmanager.fromtabletorecord - Function**

1.1.2 Restore regions from a Table to a record

**Description**

This function restores a region from an CASA Table to the global name space. For the `tablename` argument, you can specify an image `tool`, a table `tool`, or a string. If you give a string, it should be the name of an existing CASA table on disk (any kind of table).

If `numberfields` is F, then the field names of the record are the same as they are in the Table. Otherwise, the regions are put into numbered fields (the field names could be anything).

You can use the function `namesintable` to find out the names of the regions in the Table.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tablename</code></td>
<td>The table</td>
<td>string</td>
<td>Image tool, table tool or String</td>
</tr>
<tr>
<td><code>regionname</code></td>
<td>Name of the region(s) to restore</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td><code>verbose</code></td>
<td>Report successful restores</td>
<td>bool</td>
<td>True</td>
</tr>
</tbody>
</table>

**Returns**

`record`

**Example**

373
- img = ia.open('hcn')
- rec = rg.fromtabletorecord(img, numberfields=T)
- print is_region(rec[0])

The record fields are numbered, not named.
regionmanager.intersection.html

**regionmanager.intersection - Function**

1.1.2 Create the intersection of some world regions

**Description**

This function (short-hand name int) creates a region which is the intersection of the given world regions. The input regions can themselves be compound regions (such as the union or intersection etc). The input regions must be provided as a Python dictionary of regions (see examples).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>World regions and comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>regions</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
<tr>
<td></td>
<td>Region tools or record of region tools, and String</td>
</tr>
<tr>
<td>comment</td>
<td>A comment stored with the region</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>String</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

- ia.open('hcn')
- csys = ia.coordsys()
- rg.setcoordinates(csys.torecord())
- b1c = "10pix 10pix 1pix"
- trc = "60pix 60pix 1pix"
- r1 = rg.wbox(blc=blc, trc=trc, pixelaxes=[0,1,2])
In this example, we use pixel coordinates so that it is clear what is happening. You can see that the number of pixels in the intersection (51) is less than the number in the bounding box of the intersection (121) because the intersection is actually polygonal and does not fill the bounding box.

Example

- ia.open('onno')
- csys = ia.coordsys()
- x = qa.quantity([3,6,9,6,5,5,3],'pix')
- y = qa.quantity([3,4,7,9,7,5,5],'pix')
- regions = {}
- regions['poly'] = rg.wpoly(x,y,[1,2],csys.torecord())
- blc = "17:42:29.303 -28.59.18.600"
- trc = "17:42:28.303 -28.59.10.600"
- regions['box'] = rg.wbox(blc,trc,[0,1],csys.torecord())
- r3 = rg.intersection(regions, 'The mysteries of CASA')

This example is the same as the previous one, except the regions are provided to the intersection function in a record, rather than directly in the call sequence.
regionmanager.ispixelregion.html

regionmanager.ispixelregion - Function

1.1.2 Is this region a pixel region?

Description

NOT IMPLEMENTED IN CASA
This function returns T if the region is a pixel region. For any other Glish variable it returns F.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>The region</th>
</tr>
</thead>
<tbody>
<tr>
<td>region</td>
<td>allowed: record</td>
</tr>
<tr>
<td></td>
<td>Default: Region tool</td>
</tr>
</tbody>
</table>

Returns

bool

Example

- ia.open(‘hcn’)
- csys = ia.coordsys()
- r1 = rg.box() # A pixel region
- r2 = rg.wbox(csys=csys.torecord()) # A world region
- rg.ispixelregion(r1) T
- rg.ispixelregion(r2) F
- x = [20,30]
- rg.ispixelregion(x) F
regionmanager.isworldregion.html

regionmanager.isworldregion - Function

1.1.2 Is this region a world region?

Description

NOT IMPLEMENTED IN CASA
This function returns T if the region is a world region. For any other Glish variable it returns F.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>The region allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>region</td>
<td>record</td>
<td>Region tool</td>
</tr>
</tbody>
</table>

Returns

bool

Example

- ia.open('hcn')
- csys = ia.coordsys()
- r1 = rg.box()  # A pixel region
- r2 = rg.wbox(csys=csys.torecord())  # A world region
- rg.isworldregion(r1)
  F
- rg.isworldregion(r2)
  T
- x = [20,30]
- rg.isworldregion(x)
  F
regionmanager.namesintable.html

**regionmanager.namesintable - Function**

Find the names of the regions stored in a Table

**Description**

This function returns the names of regions stored in an CASA Table. For the `tablename` argument, you can specify a string; it should be the name of an existing CASA table on disk (any kind of table).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tablename</td>
<td>The table</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>Default:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Image tool, table tool or String</td>
</tr>
</tbody>
</table>

**Returns**

stringArray

**Example**

- names=rg.namesintable('hcn')
- names
  r1 poly2 int0
**regionmanager.setcoordinates - Function**

**1.1.2 Set new default Coordinate System**

**Description**

This function allows you to (re)set the default Coordinate System used by the functions that make world regions. If you don’t specify a Coordinate System when you make the world region, the default Coordinate System, if there is one, is used. The Coordinate System is stored in a coordinates tool and is created with the coordsys tool function. Normally, the world region creating functions like wbox and wpolygon will issue a message each time the private Coordinate System is used. However, if you set verbose=F then this will not occur.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>csys</td>
<td>Default Coordinate System for use in world regions</td>
</tr>
<tr>
<td></td>
<td>allowed: record</td>
</tr>
<tr>
<td></td>
<td>Default: Coordinate tool</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

- ia.open('quiqui')
- csys = ia.coordsys()
- rg.setcoordinates(csys.torecord())
- r1 = rg.wbox()

Using private CoordinateSystem from image "quiqui"
**regionmanager.makeunion - Function**

Create a union of world regions

**Description**

This function takes a minimum of two world regions and creates a region which is the union of the given regions. The input regions can themselves be compound regions (such as the union or intersection etc). The input regions must be a Python dictionary of at least two regions (see examples).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>regions</td>
</tr>
<tr>
<td>allowed:</td>
</tr>
<tr>
<td>Default:</td>
</tr>
<tr>
<td>comment</td>
</tr>
<tr>
<td>allowed:</td>
</tr>
<tr>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```plaintext
- ia.open('onno')
- csys = ia.coordsys()
- x = qa.quantity([3,6,9,6,5,5,3],'pix')
- y = qa.quantity([3,4,7,9,7,5,5],'pix')
- r1 = rg.wpoly(x,y,[1,2],csys.torecord())
- blc = "17:42:29.303 -28.59.18.600"
```
When the polygon only is applied, it is auto extended along the third axis. The \texttt{statistics} function finds 6400 pixels in the region, which is $6400/256=25$ pixels per plane. Likewise, when the box only is applied, the \texttt{statistics} function finds 32256 pixels in the region, which is $32256/256=126$ pixels per plane. When the union is applied, the \texttt{statistics} function finds 38656 pixels in the region. First it finds the union of the polygon and box (which are specified only in the XY plane) and that union is extended. Thus we expect $(25+126)*256=38656$ pixels in the region of the union, as found.

Example
- ia.open('onno')
- csys = ia.coordsys()
- x = qa.quantity([3,6,9,6,5,5,3], 'pix')
- y = qa.quantity([3,4,7,9,7,5,5], 'pix')
- regions = {}
- regions['poly'] = rg.wpoly(x,y,[0,1],csys.torecord())
- blc = "17:42:29.303 -28.59.18.600"
- trc = "17:42:28.303 -28.59.10.600"
- regions['box'] = rg.wbox(blc,trc,[0,1],csys.torecord())
- r3 = rg.union(regions, 'The mysteries of CASA')

This example is the same as the previous one, except the regions are provided to the union function in a record, rather than directly in the call sequence.
regionmanager.wbox.html

regionmanager.wbox - Function

1.1.2 Create a world box region

Description

This function creates a multi-dimensional world box region; the corners of the box are specified in world coordinates. However, the box is not a true world volume in that its sides do not follow world contours. Its sides are parallel to the pixel axes. If you are in a region of high world coordinate contour non-linearity (e.g. near a pole), you are probably better off using a world polygon.

The box is specified by a bottom-left corner, and a top-right corner. The coordinates are given as quantities, and you can give a vector of quantities (e.g. blc = qa.quantity("1rad 20deg") or a quantity of a vector (e.g. blc = qa.quantity([10,30], 'rad')).

You can specify whether the coordinates are given as absolute coordinates (absrel='abs') or relative to the reference pixel (absrel='relref') or relative to the center of the image (absrel='relcen'). You can specify this for each axis (the same for the blc and trc) If you specify less values than the number of values in blc or trc then the last value you did specify is used as the default for all higher numbered axes (e.g. absrel='relref' means absrel="relref relref" for two axes).

You specify which pixel axes in the image the blc and trc vector refer to with the pixelaxes argument. If you don’t, it defaults to [0,1,2,...]. This specification is an important part of world regions.

You must also specify the Coordinate System with the csys argument. The Coordinate System is encapsulated in a coordinates tool and can be recovered from an image with the coordsys tool function. You can also set a default Coordinate System in the Regionmanager with the setcoordinates function.

In the Regionmanager we have defined units ‘pix’ and ‘frac’; these are then known to the quanta system. This means that you can effectively define a pixel box (except for the stride capability) as a world box with most of the advantages of world regions (can be used for compound regions). However, it is still not very portable to other images because the coordinates are pixel based, not world based.

Note that the need to deal with the pixelaxes and csys is hidden from you when using the gui interface of the Regionmanager.

Arguments
Inputs

- **blc**: blc of the box; a vector of quantities
  - allowed: any
  - Default: variant
  - Unity

- **trc**: trc of the box; a vector of quantities
  - allowed: any
  - Default: variant
  - Shape

- **pixelaxes**: Which pixel axes
  - allowed: intArray
  - Default: -1
  - [0,1,2,...]

- **csys**: Coordinate System
  - allowed: record
  - Default: Private Coordinate System

- **absrel**: Absolute or relative coordinates Vector of strings from 'abs', 'relref' and 'relen'
  - allowed: string
  - Default: 'abs'

- **comment**: A comment stored with the region
  - allowed: string
  - Default: 

Returns

- record

Example

```
- r = rg.wbox()
```

This region, on application to an image, will select the entire image.

Example

386
- ia.open('ada')
- csys = ia.coordsys()
- csys.summary()

<table>
<thead>
<tr>
<th>Name</th>
<th>Proj</th>
<th>Shape</th>
<th>Tile</th>
<th>Coord value at pixel</th>
<th>Coord incr</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>64</td>
<td>16</td>
<td>1.413350e+09</td>
<td>1.00</td>
<td>1.968717e+04 Hz</td>
<td></td>
</tr>
<tr>
<td>Velocity</td>
<td></td>
<td></td>
<td>1.378053e+02</td>
<td>1.00</td>
<td>-4.174021e+00 km/s</td>
<td></td>
</tr>
<tr>
<td>Declination</td>
<td>SIN</td>
<td>178</td>
<td>-28.59.18.600</td>
<td>90.00</td>
<td>1.000000e+00 arcsec</td>
<td></td>
</tr>
<tr>
<td>Right Ascension</td>
<td>SIN</td>
<td>155</td>
<td>17:42:29.303</td>
<td>90.00</td>
<td>-1.000000e+00 arcsec</td>
<td></td>
</tr>
</tbody>
</table>

- blc = "17:42:29.303 -28.59.18.600"
- trc = "17:42:28.303 -28.59.10.600"
- r1 = rg.wbox(blc=blc, trc=trc, pixelaxes=[0,1], csys=csys.torecord())
- ia.boundingbox(r1)

[blc=[1 90 90] , trc=[64 98 103] , regionShape=[64 9 14], imageShape=[64 178 155] ]

We have specified an RA and DEC for the blc and the trc (they should be quantities; for blc we do that explicitly, but for the trc we just give a vector of strings which is automatically converted for us to a vector of quantities).

From the \{stff summary\} listing you can see that RA and DEC correspond to pixel axes 3 and 2 respectively (don’t be confused by the dual listing for the spectral axis) so that is why the {stf pixelaxes} argument is set to [3,2]. If we had set blc/trc in DEC/RA order then we would have put {stf pixelaxes=[1,2]}. For the unspecified frequency axis, all pixels are selected.

Example

- ia.open('bork')
- csys = ia.coordsys()
- csys.summary()

<table>
<thead>
<tr>
<th>Name</th>
<th>Proj</th>
<th>Shape</th>
<th>Tile</th>
<th>Coord value at pixel</th>
<th>Coord incr</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Velocity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Declination</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Right Ascension</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

387
Right Ascension    SIN    155   31  17:42:29.303  90.00  -1.000000e+00 arcsec
Declination       SIN    178   89  -28.59.18.600  90.00  1.000000e+00 arcsec

- rg.setcoordinates(cs)
- blc = "-10pix -28.59.18.6"
- trc = "10pix -28.59.1.6"
- r1 = rg.wbox(blc=blc, trc=trc, absrel="relref abs")  # pixelaxes defaults to [0,1]
Using private CoordinateSystem from image "bork"
- ia.boundingbox(r1)
[blc=[80 90] , trc=[100 107] , regionShape=[21 18] , imageShape=[155 178] ]

In this example, we use pixel coordinates relative to the reference pixel for the RA axis and absolute world coordinates for the DEC axis. We also set the state of the \regionmanager\ with a Coordinate System to use when making world regions. You can see that when the region was made, a message was issued reminding you that the internal Coordinate System from the image \sff bork\ was being used.

Example

- ia.open('hcn')
- csys = ia.coordsys()
- csys.summary()

<table>
<thead>
<tr>
<th>Name</th>
<th>Proj</th>
<th>Shape</th>
<th>Tile</th>
<th>Coord value at pixel</th>
<th>Coord incr</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Right Ascension</td>
<td>SIN</td>
<td>155</td>
<td>31</td>
<td>17:42:29.303</td>
<td>90.00</td>
<td>-1.000000e+00 arcsec</td>
</tr>
<tr>
<td>Declination</td>
<td>SIN</td>
<td>178</td>
<td>89</td>
<td>-28.59.18.600</td>
<td>90.00</td>
<td>1.000000e+00 arcsec</td>
</tr>
<tr>
<td>Frequency</td>
<td></td>
<td>64</td>
<td>16</td>
<td>1.41350e+09</td>
<td>1.00</td>
<td>1.968717e+04 Hz</td>
</tr>
<tr>
<td>Velocity</td>
<td></td>
<td>1.378053e+02</td>
<td></td>
<td>1.00</td>
<td>-4.174021e+00 km/s</td>
<td></td>
</tr>
</tbody>
</table>

- blc = "1.414E9Hz"
- trc = "1.4145GHz"
- r = rg.wbox(blc=blc, trc=trc, pixelaxes=[2], csys=cs)
- ia.boundingbox(r)
[blc=[1 1 34] , trc=[155 178 59] , regionShape=[155 178 26] , imageShape=[155 178 64] ]

388
In this example we only specified a region for the frequency axis (note we used different units for the blc and trc). Therefore, on application, the region selected for the RA and DEC axes is automatically the full image.
regionmanager.wpolygon.html

regionmanager.wpolygon - Function

1.1.2 Create a world polygon region with quantities

Description

This function (short-hand name wpoly) creates a 2D world polygon region. The polygon is specified by an x and a y vector. These must be quantities of a vector (the world box function allows both quantities of vectors and vectors of quantities). This means that the units are common to all elements of each vector. Thus, qa.quantity([1,2,3],'rad') (a quantity of a vector) is different from qa.quantity("1rad 2rad 3rad") (a vector of quantities) although the information that they specify is the same.

You specify which pixel axes in the image the x and y vectors pertain to with the pixelaxes argument. If you don’t, it defaults to [0,1]. This specification is an important part of world regions.

You can specify whether the x and y vector coordinates are given as absolute coordinates (absrel='abs') or relative to the reference pixel (absrel='relref') or relative to the center of the image (absrel='relcen'). This argument applies to both the axes of the polygon.

You must also specify the Coordinate System with the csys argument. The Coordinate System is encapsulated in a coordinates tool and can be recovered from an image with the coordsys function. You can also set a default Coordinate System in the Regionmanager with the setcoordinates function.

In the Regionmanager we have defined units ‘pix’ and ‘frac’; these are then known to the quanta system. This means that you can effectively define a pixel box (except for the stride capability) as a world box with most of the advantages of world regions (can be used for compound regions). However, it is still not very portable to other images because the coordinates are pixel based, not world based.

Note that the need to deal with the pixelaxes and csys is hidden from you when using the gui interface of the Regionmanager.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>The x vector; a vector of quantities</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>y</td>
<td>The y vector; vector of quantities</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>pixelaxes</td>
<td>Quantity vector</td>
<td>intArray</td>
<td>-1, 1</td>
</tr>
<tr>
<td>csys</td>
<td>Coordinate System</td>
<td>record</td>
<td>Private Coordinate System</td>
</tr>
<tr>
<td>absrel</td>
<td>Absolute or relative coordinates; possibilities are 'abs', 'rel', 'relcen'</td>
<td>string</td>
<td>abs, 'abs'</td>
</tr>
<tr>
<td>comment</td>
<td>A comment stored with the region</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

| Returns    |                                      | record        |                  |

**Example**

```python
ia.open('myim.im')
csys = ia.coordsys()
x = ['3pix','6pix','9pix','6pix','5pix','5pix','3pix']
y = ['3pix','4pix','7pix','9pix','7pix','5pix','5pix']
r1 = rg.wpolygon(x=x, y=y, pixelaxes=[0,1], csys=csys.torecord())
stats = ia.statistics(region=r1)
ia.done()
```
We applied the 2D polygon, defined in the XY plane with absolute pixel coordinates, to a 3D image. Therefore, the third (Z) axis was automatically extended to the whole image.
1.1.3  coordsys - Tool

Operations on CoordinateSystems
Requires:

Synopsis

Description

Summary
A Coordsys tool is used to store and manipulate a Coordinate System (we will use the term ‘Coordinate System’ interchangeably with ‘Coordsys tool’). A Coordinate System is a collection of coordinates, such as a direction coordinate (E.g. RA/DEC), or a spectral coordinate (e.g. an LSRK frequency).
The main job of the Coordsys tool is to convert between absolute pixel and world (physical) coordinates. It also supports relative pixel and world coordinates (relative to reference location).
A Coordinate System is generally associated with an image (manipulated via an Image tool) but can also exist in its own right. An image is basically just a regular lattice of pixels plus a Coordinate System describing the mapping from pixel coordinate to world (or physical) coordinate.
Each coordinate is associated with a number of axes. For example, a direction coordinate has two coupled axes; a longitude and a latitude. A spectral coordinate has one axis. A linear coordinate can have an arbitrary number of axes, but they are all uncoupled. The Coordinate System actually maintains two kinds of axes; pixel axes and world axes.
As well as the coordinates, there is some extra information stored in the Coordinate System. This consists of the telescope, the epoch (date of observation), and the highly influential observer’s name. The telescope (i.e. position on earth) and epoch are important if you want to, say, change a spectral coordinate from LSRK to TOPO.
For general discussion about celestial coordinate systems, see the papers by Mark Calabretta and Eric Greisen. Background on the WCS system and relevant papers (including the papers published in


can be found here. Note that the actual system implemented originally in CASA was based on a 1996 draft of these papers. The final papers are being implemented while new version of the defining library become available.

Coordinate formatting
Many of the Coordsys tool functions use a world coordinate value as an argument. This world value can be formatted in many ways. Some functions (e.g. toworld) have a function argument called format which takes a string. This controls the format in which the coordinate is output and hence possibly input into some other function. Possibilities for format are:

- 'n' - means the world coordinate is given as a numeric vector (actually doubles). The units are implicitly those returned by function units.
- 'q' - means the world coordinate is given as a vector of quantities (value and unit) - see the quanta module. If there is only one axis (e.g. spectral coordinate), you will get a single quantum only.
- 'm' - means the world coordinate is given as a record of measures - see the measures module.
  
The record consists of fields named direction, spectral, stokes, linear, and tabular, depending upon which coordinate types are present in the Coordinate System.
  
The direction field holds a direction measure.
  
The spectral field holds further subfields frequency, radiovelocity, opticalvelocity, betavelocity. The frequency subfield holds a frequency measure. The radiovelocity subfield holds a doppler measure using the radio velocity definition. The opticalvelocity subfield holds a doppler measure using the optical velocity definition. The betavelocity subfield holds a doppler measure using the true or beta velocity definition.
  
The stokes field just holds a string giving the Stokes type (not a real measure).
  
The linear and tabular fields hold a vector of quanta (not a real measure).
- 's' - means the the world coordinate is given as a vector of formatted strings

You can give a combination of one or more of the allowed letters when using the format argument. The coordinate is given as a record, with possible fields 'numeric', 'quantity', 'measure' and 'string' where each of these fields is given as described above.

There are functions torel and toabs used to inter-convert between absolute and relative world and pixel coordinates. These functions have an argument isworld whereby you can specify whether the coordinate is a pixel coordinate or a world coordinate. In general, you should not need to use this argument because any coordinate variable generated by Coordsys tool functions 'knows' whether it is absolute or relative, world or pixel. However, you may be
inputting a coordinate variable which you have generated in some other way, and then you may need this.

**Stokes Coordinates**
Stokes axes don’t fit very well into our Coordinate model since they are not interpolatable. The alternative to having a Stokes Coordinate is having a Stokes pixel type (like double, complex). Both have their good and bad points. We have chosen to use a Stokes Coordinate. With the Stokes Coordinate, any absolute pixel coordinate value must be in the range of 1 to \( n_{\text{Stokes}} \), where \( n_{\text{Stokes}} \) is the number of Stokes types in the Coordinate. We define relative world coordinates for a Stokes axis to be the same as absolute world coordinates (it makes no sense to think of a relative value \( XY - XX \) say). You can use the specialized functions stokes and setstokes to recover and set new Stokes values in the Stokes Coordinate.

**World and Pixel axes**
The Coordinate System maintains what it calls pixel and world axes. The pixel axis is associated with, say, the axes of a lattice of pixels. The world axes describe notional world axes, generally in the same order as the pixel axes. However, they may be different. Imagine that a 3-D image is collapsed along one axis. The resultant image has 2 pixel axes. However, we can maintain the world coordinate for the collapsed axis (so we know the coordinate value still). Thus we have three world axes and two pixel axes. It is also possible for the C++ programmer to reorder these pixel and world axes. However, this is strongly discouraged, and you should never actually encounter a situation where the pixel and world axes are in different orders, but you may encounter cases where the number of world and pixela axes is different. For those of us (CASA programmers) writing robust scripts, we must account for these possibilities, although the user really shouldn’t bother. Thus, the pixel and world vectors return the pixel and world axes of the found coordinate.
The functions referencevalue, increment, units, and names return their vectors in world axis order. However, function referencepixel returns in pixel axis order (and the world vectors might have more values than the referencepixel vector).

**Overview of Coordsys tool functions**
- **Get/set** - Functions to get and set various items within the Coordinate System are
  - referencepixel - get the reference pixel
  - setreferencepixel - set the reference pixel
  - referencevalue - get the reference value
• Utility - There is a range of utility services available through the functions
  
  – axesmap - get mapping between pixel and world axes order
  – axiscoordinatetypes - get type of coordinate for each axis
  – coordinatetype - get type of coordinates
  – copy - make a copy of this tool
  – done - destroy this tool
• Coordinates

  – findaxis - find specified axis (by number) in coordinate system
  – findcoordinate - find specified (by number) coordinate
  – fromrecord - set Coordinate System from a casapy record
  – id - get the fundamental identifier of this tool
  – naxes - get number of axes
  – ncoordinates - get the number of coordinates
  – reorder - reorder coordinates
  – summary - summarize the Coordinate System
  – torecord - Convert a Coordinate SYstem to a casapy record
  – type - the type of this tool

• Coordinate conversion

  – convert - Convert one numeric coordinate with mixed input and output formats (abs/rel/world/pixel)
  – toabs - Convert a relative coordinate to an absolute coordinate
  – topixel - Convert from absolute world coordinate to absolute pixel coordinate
  – torel - Convert an absolute coordinate to a relative coordinate
  – toworld - Convert from an absolute pixel coordinate to an absolute world coordinate
  – convertmany - Convert many numeric coordinates with mixed input and output formats (abs/rel/world/pixel)
  – toabsmany - Convert many relative coordinates to absolute coordinates
  – topixelmany - Convert many absolute world coordinates to absolute pixel coordinates
  – torelmany - Convert many absolute coordinates to relative coordinates
  – toworldmany - Convert many absolute pixel coordinates to absolute world coordinates
  – frequencytovelocity - Convert from frequency to velocity
  – frequencytofrequency - Apply a relativistic Doppler shift to a list of frequencies
  – velocitytofrequency - Convert from velocity to frequency
  – setconversiontype - Set extra reference frame conversion layer
  – conversiontype - Recover extra reference frame conversion types

• Tests -

  397
- coordsystest - Run test suite for Coordsys tool

Methods
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>newcoordsys</code></td>
<td>Create a non-default coordsys tool</td>
</tr>
<tr>
<td><code>addcoordinate</code></td>
<td>Add default coordinates. (For assay testing only.)</td>
</tr>
<tr>
<td><code>axesmap</code></td>
<td>Find mapping between world and pixel axes</td>
</tr>
<tr>
<td><code>axiscoordinatetypes</code></td>
<td>Return types of coordinates for each axis</td>
</tr>
<tr>
<td><code>conversiontype</code></td>
<td>Get extra reference conversion layer</td>
</tr>
<tr>
<td><code>convert</code></td>
<td>Convert a numeric mixed coordinate</td>
</tr>
<tr>
<td><code>convertdirection</code></td>
<td>Convert the direction coordinate to the specified frame by rotating as necessary about the reference pixel.</td>
</tr>
<tr>
<td><code>convertmany</code></td>
<td>Convert many numeric mixed coordinates</td>
</tr>
<tr>
<td><code>coordinatetype</code></td>
<td>Return type of specified coordinate</td>
</tr>
<tr>
<td><code>copy</code></td>
<td>Copy this Coordsys tool</td>
</tr>
<tr>
<td><code>done</code></td>
<td>Destroy this Coordsys tool, restore default tool</td>
</tr>
<tr>
<td><code>epoch</code></td>
<td>Return the epoch</td>
</tr>
<tr>
<td><code>findaxis</code></td>
<td>Find specified axis in coordinate system</td>
</tr>
<tr>
<td><code>findaxisbyname</code></td>
<td>Find specified axis in coordinate system.</td>
</tr>
<tr>
<td><code>findcoordinate</code></td>
<td>Find axes of specified coordinate</td>
</tr>
<tr>
<td><code>frequencytofrequency</code></td>
<td>Apply relativistic Doppler shift to a list of frequencies</td>
</tr>
<tr>
<td><code>frequencytovelocity</code></td>
<td>Convert frequency to velocity</td>
</tr>
<tr>
<td><code>fromrecord</code></td>
<td>Fill Coordinate System from a record</td>
</tr>
<tr>
<td><code>increment</code></td>
<td>Recover the increments</td>
</tr>
<tr>
<td><code>lineartransform</code></td>
<td>Recover the linear transform matrix</td>
</tr>
<tr>
<td><code>names</code></td>
<td>Recover the names for each axis</td>
</tr>
<tr>
<td><code>naxes</code></td>
<td>Recover the number of axes</td>
</tr>
<tr>
<td><code>ncordinates</code></td>
<td>Recover the number of coordinates in the Coordinate System</td>
</tr>
<tr>
<td><code>observer</code></td>
<td>Return the name of the observer</td>
</tr>
<tr>
<td><code>projection</code></td>
<td>Recover the direction coordinate projection</td>
</tr>
<tr>
<td><code>referencecode</code></td>
<td>Return specified reference code</td>
</tr>
<tr>
<td><code>referencepixel</code></td>
<td>Recover the reference pixel</td>
</tr>
<tr>
<td><code>referencevalue</code></td>
<td>Recover the reference value</td>
</tr>
<tr>
<td><code>reorder</code></td>
<td>Reorder the coordinates</td>
</tr>
<tr>
<td><code>transpose</code></td>
<td>Transpose the axes</td>
</tr>
<tr>
<td><code>replace</code></td>
<td>Replace a coordinate</td>
</tr>
<tr>
<td><code>restfrequency</code></td>
<td>Recover the rest frequency</td>
</tr>
<tr>
<td><code>setconversiontype</code></td>
<td>Set extra reference conversion layer</td>
</tr>
<tr>
<td><code>getconversiontype</code></td>
<td>Get extra reference conversion layer (aka conversiontype).</td>
</tr>
<tr>
<td><code>setdirection</code></td>
<td>Set direction coordinate values</td>
</tr>
<tr>
<td><code>setepoch</code></td>
<td>Set a new epoch</td>
</tr>
<tr>
<td><code>setincrement</code></td>
<td>Set the increment</td>
</tr>
<tr>
<td><code>setlineartransform</code></td>
<td>Set the linear transform</td>
</tr>
<tr>
<td><code>setnames</code></td>
<td>Set the axis names</td>
</tr>
<tr>
<td><code>setobserver</code></td>
<td>Set a new observer</td>
</tr>
<tr>
<td><code>setprojection</code></td>
<td>Set the direction coordinate projection</td>
</tr>
<tr>
<td><code>setreferencecode</code></td>
<td>Set new reference code</td>
</tr>
<tr>
<td><code>setreferencelocation</code></td>
<td>Set reference pixel and value</td>
</tr>
<tr>
<td><code>setreferencepixel</code></td>
<td>Set the reference pixel</td>
</tr>
<tr>
<td><code>setreferencevalue</code></td>
<td>Set the reference value</td>
</tr>
<tr>
<td><code>setrestfrequency</code></td>
<td>Set the rest frequency</td>
</tr>
</tbody>
</table>
setspectral
setstokes
settabular
settelescope
setunits
stokes
summary
telescope
toabs
toabsmany
topixel
topixelmany	orecord
subimage	orel	orelmany	oworld	oworldmany	type	units
velocitytofrequency
parentname
setparentname

Set tabular values for the spectral coordinate
Set the Stokes types
Set tabular values for the tabular coordinate
Set a new telescope
Set the axis units
Recover the Stokes types
Summarize basic information about the Coordinate System
Return the telescope
Convert relative coordinate to absolute
Convert many numeric relative coordinates to absolute
Convert from absolute world to pixel coordinate
Convert many absolute numeric world coordinates to pixel
Convert Coordinate System to a record
delivers a coordinate origin re-referenced for a subimage
Convert absolute coordinate to relative
Convert many numeric absolute coordinates to relative
Convert from absolute pixel coordinate to world
Convert many absolute pixel coordinates to numeric world
Return the type of this tool
Recover the units for each axis
Convert velocity to frequency
Get parent image name.
Set the parent image name (normally not needed by end-users)
coordsys.newcoordsys.html

coordsys.newcoordsys - Function

1.1.3 Create a non-default coordsys tool

Description

By default, this constructor makes an empty Coordsys tool. You can ask it to include various sorts of coordinates through the arguments. Except for Stokes, you don’t have any control over the coordinate contents (e.g. reference value etc.) it does make for you on request. But you can edit the Coordinate System after creation if you wish.

If you wish to make a Stokes coordinate, then you assign stokes to a string (or a vector of strings) saying which Stokes you want. CASA allows rather a lot of potential Stokes types.

Probably most useful is some combination of the basic I, Q, U, V, XX, YY, XY, YX, RR, LL, RL, and LR.

However, a more esoteric choice is also possible: RX, RY, LX, LY, XR, XL, YR, YL (these are mixed linear and circular), PP, PQ, QP, QQ (general quasi-orthogonal correlation products) RCircular, LCircular, Linear (single dish polarization types).

You can also specify some polarization ‘Stokes’ types: Ptotal (Polarized intensity \((Q^2 + U^2 + V^2)^{1/2}\)), Plinear (Linearly Polarized intensity \((Q^2 + U^2)^{1/2}\)), PFtotal (Polarization Fraction (Ptotal/I)), PFlinear (Linear Polarization Fraction (Plinear/I)), and Pangle (Linear Polarization Angle \(0.5 \arctan(U/Q)\) in radians).

Probably you will find the more unusual types aren’t fully supported throughout the system.

You can make a LinearCoordinate with as many uncoupled axes as you like. Thus, linear=2 makes one LinearCoordinate with 2 axes (think of it like a DirectionCoordinate which also has 2 axes [but coupled in this case], a longitude and a latitude).

If you make a TabularCoordinate, it is linear to start with. You can change it to a non-linear one by providing a list of pixel and world values to function settabular.

Arguments
Inputs

direction Make a direction coordinate?
allowed: bool
Default: false

spectral Make a spectral coordinate?
allowed: bool
Default: false

stokes Make a Stokes coordinate with these Stokes
allowed: stringArray
Default: I Q U V XX YY XY YX RR LL RL LR

linear Make a linear coordinate with this many axes
allowed: int
Default: 0

tabular Make a tabular coordinate
allowed: bool
Default: false

Returns
cordsys

Example

"""
# print "	\t newcoordsys Ex 1 \t"
cs1=cs.newcoordsys()
print 'ncoordinates =',cs1.ncoordinates()
#0
cs1.done()
#True
cs2=cs.newcoordsys(direction=T, stokes=['I','V'])
print 'ncoordinates =',cs2.ncoordinates()
#2L
print cs2.coordinatetype()
#['Direction', 'Stokes']
cs2.summary()
#
"""
The second Coordinate System contains a direction coordinate and a Stokes coordinate. This means that there are three ‘axes’ associated with the 2 coordinates.
coordsys.addcoordinate.html

**coordsys.addcoordinate - Function**

1.1.3 Add default coordinates. (For assay testing only.)

**Description**

Add default coordinates of the specified types. This function allows multiple coordinates of the same type which are not well supported. Use only for assay tests.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>direction</td>
<td>Add a direction coordinate ?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>spectral</td>
<td>Add a spectral coordinate ?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>stokes</td>
<td>Add a Stokes coordinate with these Stokes</td>
<td>stringArray</td>
<td>I Q U V XX YY XY YX RR LL RL LR</td>
</tr>
<tr>
<td>linear</td>
<td>Add a linear coordinate with this many axes</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>tabular</td>
<td>Add a tabular coordinate</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**
print "\t----\t addcoordinate Ex 1 \t----"
mycs=cs.newcoordsys()
mycs.addcoordinate(direction=T)
mycs.done()
#
coordsys.axesmap.html

coordsys.axesmap - Function
1.1.3 Find mapping between world and pixel axes

Description

This function returns a vector describing the mapping from pixel to world or world to pixel axes. It is not for general user use. See the discussion about pixel and world axis ordering. Generally they will be in the same order.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Map from pixel to world</th>
</tr>
</thead>
<tbody>
<tr>
<td>toworld</td>
<td></td>
</tr>
<tr>
<td>allowed</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>true</td>
</tr>
</tbody>
</table>

Returns

intArray

Example

""
#
print "\t----\t axesmap Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
csys.axesmap(T);
#[1L, 2L, 3L]
csys.axesmap(F);
#[1L, 2L, 3L]
#
"""
**coordsys.axiscoordinatetypes.html**

**coordsys.axiscoordinatetypes - Function**

1.1.3 Return types of coordinates for each axis

**Description**

This function returns a vector string giving the coordinate type for each axis (world or pixel) in the Coordinate System. See the discussion about pixel and world axis ordering.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>world</td>
<td>World or pixel axes ?</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: true</td>
</tr>
</tbody>
</table>

**Returns**

stringArray

**Example**

```python
""
#
print "\t----\t axiscoordinatetypes Ex 1 \t----"
csys=cs.newcoordsys(direction=T,spectral=T)
csys.axiscoordinatetypes()
#['Direction', 'Direction', 'Spectral']
#
""
```

---

407
**coordsys.conversiontype** - Function

1.1.3 Get extra reference conversion layer

**Description**

Some coordinates contain a reference code. Examples of reference codes are B1950 and J2000 for direction coordinates, or LSRK and BARY for spectral coordinates. When you do conversions between pixel and world coordinate, the coordinates are in the reference frame corresponding to these codes. Function `setconversiontype` allows you to specify a different reference frame which is used when converting between world and pixel coordinate. This function allows you to recover those conversion types. If no extra conversion layer has been set, you get back the native reference types.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Coordinate type, direction, spectral</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>string</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>direction</td>
</tr>
</tbody>
</table>

**Returns**

string

**Example**

```python
#
print "\t----\t conversiontype Ex 1 \t----"

csys = cs.newcoordsys(direction=T, spectral=T)
print csys.conversiontype (type='direction'), ' ', csys.conversiontype (type='spectral')

#J2000 LSRK
csys.setconversiontype (direction='GALACTIC', spectral='BARY')
print csys.conversiontype (type='direction'), ' ', csys.conversiontype (type='spectral')
```

408
**coordsys.convert - Function**

Convert a numeric mixed coordinate

### Description

This function converts between mixed pixel/world/abs/rel numeric coordinates. The input and output coordinates are specified via a numeric vector giving coordinate values, a string vector giving units, a boolean vector specifying whether the coordinate is absolute or relative (to the reference pixel) and doppler strings specifying the doppler convention for velocities. The units string may include `pix` for pixel coordinates and velocity units (i.e. any unit consistent with m/s).

The allowed doppler strings and definition are described in function summary. The `shape` argument is optional. If your Coordinate System is from an image, then assign the image shape to this argument. It is used only when making mixed (pixel/world) conversions for Direction Coordinates to resolve ambiguity.

The example clarifies the use of this function.

### Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coordin</td>
<td>Input coordinate, as a numeric vector</td>
</tr>
<tr>
<td></td>
<td>allowed: doubleArray</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>absin</td>
<td>Are input coordinate elements absolute ?</td>
</tr>
<tr>
<td></td>
<td>allowed: boolArray</td>
</tr>
<tr>
<td></td>
<td>Default: true</td>
</tr>
<tr>
<td>dopplerin</td>
<td>Input doppler type for velocities</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: radio</td>
</tr>
<tr>
<td>unitsin</td>
<td>Input units, string vector</td>
</tr>
<tr>
<td></td>
<td>allowed: stringArray</td>
</tr>
<tr>
<td></td>
<td>Default: Native</td>
</tr>
<tr>
<td>absout</td>
<td>Are output coordinate elements absolute ?</td>
</tr>
<tr>
<td></td>
<td>allowed: boolArray</td>
</tr>
<tr>
<td></td>
<td>Default: true</td>
</tr>
<tr>
<td>dopplerout</td>
<td>Output doppler type for velocities</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: radio</td>
</tr>
<tr>
<td>unitsout</td>
<td>Output units</td>
</tr>
<tr>
<td></td>
<td>allowed: stringArray</td>
</tr>
<tr>
<td></td>
<td>Default: Native</td>
</tr>
<tr>
<td>shape</td>
<td>Image shape, integer vector</td>
</tr>
<tr>
<td></td>
<td>allowed: intArray</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
</tbody>
</table>

| Returns        | doubleArray                                                                 |

Example

In this example we convert from a vector of absolute pixels to a mixture of pixel/world and abs/rel.

```python
print "\t**** convert Ex 1 \t****"
csys=cs.newcoordsys(direction=T, spectral=T)  # 3 axes
cout=csys.convert(coordin=[10,20,30],absin=[T,T,T],
                   unitsin=["pix","pix","pix"],
                   absout=[T,F,T], dopplerout='optical',
```
unitsout=["pix","arcsec","km/s"])  
print cout  
#[10.0, 1140.0058038878046, 1139.1354056919731]  
#  
"""
coordsys.convertdirection - Function

1.1.3 Convert the direction coordinate to the specified frame by rotating as necessary about the reference pixel so the axes line up with the cardinal directions.

Description

Convert the direction coordinate in the coordinate system to the specified frame by rotating about the reference pixel so that the resulting coordinate axes are parallel to the cardinal directions. The resulting coordinate will not have a conversion layer, even if the input direction coordinate does. A conversion layer can be set after by running cs.setconversiontype(). Be aware that if you attach the resulting coordinate system to an image whose pixels have not been rotated around the reference pixel in the same manner, you will likely get an image for which the pixels do not match up to world coordinate values. This method should only be used by experienced users who know what they are doing. It was written originally to facilitate rotating the direction coordinate since the implementation of imregrid requires this in certain circumstances. The conversion is done in place; a new coordinate system tool is not created. The returned record represents an angular quantity through which the old direction coordinate was rotated to create the new coordinate.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>frame</td>
<td>Reference frame to convert to.</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
</tbody>
</table>

| Default: |

Returns

record

Example
coordsys.convertmany.html

coordsys.convertmany - Function

1.1.3 Convert many numeric mixed coordinates

Description

This function converts between many mixed pixel/world/abs/rel numeric coordinates. See function convert for more information. The only difference with that function is that you provide a matrix holding many coordinates to convert and a matrix of many converted coordinates is returned.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>coordin</td>
<td>Input coordinate, numeric matrix</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>absin</td>
<td>Are input coordinate elements absolute?</td>
<td>boolArray</td>
<td>true</td>
</tr>
<tr>
<td>dopplerin</td>
<td>Input doppler type for velocities</td>
<td>string</td>
<td>radio</td>
</tr>
<tr>
<td>unitsin</td>
<td>Input units, string vector</td>
<td>stringArray</td>
<td>Native</td>
</tr>
<tr>
<td>absout</td>
<td>Are output coordinate elements absolute?</td>
<td>boolArray</td>
<td>true</td>
</tr>
<tr>
<td>dopplerout</td>
<td>Output doppler type for velocities</td>
<td>string</td>
<td>radio</td>
</tr>
<tr>
<td>unitsout</td>
<td>Output units</td>
<td>stringArray</td>
<td>Native</td>
</tr>
<tr>
<td>shape</td>
<td>Image shape, integer array</td>
<td>intArray</td>
<td>-1</td>
</tr>
</tbody>
</table>
Returns

Example

""
#
# print "\t----\t convertmany Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T) # 3 axes
# absolute pixel coordinates; 10 conversions each of length 3; spectral
cin=[(15, 15, 15, 15, 15, 15, 15, 15, 15, 15), (20, 20, 20, 20, 20, 20, 20, 20, 20, 20),
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)]
cout = csys.convertmany (coordin=cin,
    absin=[T,T,T],
    unitsin="pix","pix","pix"],
    absout=[T,F,T],
    dopplerout='optical',
    unitsout="pix","deg","km/s")
print cout
#((15.0, 15.0, 15.0, 15.0, 15.0, 15.0, 15.0, 15.0, 15.0, 15.0),
# (0.316682788571637, 0.316682788571637, 0.316682788571637),
# 0.316682788571637, 0.316682788571637, 0.316682788571637, 0.316682788571637,
# 0.316682788571637, 0.316682788571637, 0.316682788571637, 0.316682788571637),
# (1145.3029083129913, 1145.3029083129913, 1145.3029083129913, 1144.877551885467, 1144.877551885467,
# (1144.877551885467, 1144.877551885467, 1144.877551885467, 1144.877551885467),
# 1144.877551885467, 1144.877551885467, 1144.877551885467, 1144.877551885467,
# 1144.877551885467, 1144.877551885467, 1144.877551885467, 1144.877551885467),
# 1144.877551885467, 1144.877551885467, 1144.877551885467, 1144.877551885467,
# 1144.877551885467, 1144.877551885467, 1144.877551885467, 1144.877551885467),
# (1143.3888287218554))
# """
coordsys.coordinatetype - Function

1.1.3 Return type of specified coordinate

Description

This function returns a string describing the type of the specified coordinate. If `which=unset` the types for all coordinates are returned. Possible output values are 'Direction', 'Spectral', 'Stokes', 'Linear', and 'Tabular'.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>Which coordinate ? (0-rel)</td>
<td>int</td>
</tr>
<tr>
<td>allowed:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
<td>-1</td>
</tr>
</tbody>
</table>

Returns

`stringArray`

Example

```
# #
print "\t----\t coordinatetype Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
csys.coordinatetype(0)
  #'Direction'
csys.coordinatetype()
  ['#Direction', 'Spectral']
# #
```
coordsys.copy.html

coordsys.copy - Function

1.1.3 Copy this Coordsys tool

Description

This function returns a copy, not a reference, of the Coordsys tool. It is your responsibility to call the done function on the new tool.

Arguments

Returns

coordsys

Example

""
#
print "\t----\t copy Ex 1 \t----"
cs1 = cs.newcoordsys(direction=T, spectral=T)
cs2 = cs1 # Reference
print cs1, cs2
cs1.summary()
cs2.summary()
cs1.done() # done invokes default coordsys tool
cs1.summary()
cs2.summary() # cs2 gets doned when cs1 does
cs1 = cs.newcoordsys(direction=T, spectral=T)
cs2 = cs1.copy() # Copy
cs1.done()
cs1.summary() # cs1 is default coordsys tool
cs2.summary() # cs2 is still viable
""
cs2.done()
cs2.summary()  # Now it's done (done just invokes default constructor)
#
"""
coordsys.done.html

**coordsys.done - Function**

1. Destroy this Coordsys tool, restore default tool

**Description**

If you no longer need to use a Coordsys tool calling this function will free up its resources and restore the default coordsys tool.

**Arguments**

**Returns**

bool

**Example**

```
"
#
print "\t----\t done Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
csys.done()
print csys.torecord() # default tool
#
"
```
coordsys.epoch.html

**coordsys.epoch - Function**

1.1.3 Return the epoch

**Description**

This function returns the epoch of the observation as a Measure.

**Arguments**

**Returns**

record

**Example**

```python
#
# print "\t----\t epoch Ex 1 \t----"
csys = cs.newcoordsys()
ep = csys.epoch()
print ep
#{'type': 'epoch', 'm0': {'value': 54151.96481085648, 'unit': 'd'}, 'refer': 'UTC'}
time = me.getvalue(ep) # Extract time with measures
print time
#{'m0': {'value': 54151.96481085648, 'unit': 'd'}}
qa.time(time) # Format with quanta
#'23:09:20'
#
```
coordsys.findaxis.html

**coordsys.findaxis - Function**

1.1.3 Find specified axis in coordinate system

**Description**

This function finds the specified axis in the Coordinate System. If the axis does not exist, it throws an exception.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>world</td>
<td>is axis a world or pixel axis?</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>axis</td>
<td>Axis in coordinate system</td>
<td>int</td>
<td>0</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
##
# print "\t----\t findaxis Ex 1 \t----"
csys=cs.newcoordsys(direction=T, linear=2) # RA/DEC/Lin1/Lin2
rtn=csys.findaxis(T,1) # DEC
rtn
#{'axisincoordinate': 1L, 'coordinate': 0L}
```

```python
rtn = csys.findaxis(T,2) # Lin1
rtn
#{'axisincoordinate': 0L, 'coordinate': 1L}
```

#
In these examples, the Coordinate System has 2 coordinates and 4 axes (0-rel, both world and pixel the same). The first example finds the DEC axis (coordinate system axis 1) to be the second axis of the Direction Coordinate (coordinate 0). The second example finds the first linear axis (coordinate system axis 2) to be the first axis of the Linear Coordinate (coordinate 1).
Find specified axis in coordinate system.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axisname</td>
<td>Name of axis to find. Minimal match supported</td>
</tr>
<tr>
<td>allowed: string</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>allowfriendlyname</td>
<td>Support friendly naming. Eg &quot;spectral&quot; will match &quot;frequency&quot; or &quot;velocity&quot;</td>
</tr>
<tr>
<td>allowed: bool</td>
<td></td>
</tr>
<tr>
<td>Default: true</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

`int`

**Example**

```python
# Find the declination axis
ia.open("myimage")
csys = ia.coordsys()
ia.done()
try:
```
dec_axis_number = csys.findaxisbyname("dec", False)
except Exception:
    print "Declination axis not found"

# find the spectral axis
try:
    spec_axis_number = csys.findaxisbyname("spectral", True)
except Exception:
    print "Spectral axis could not be found."
Description

This function finds the axes in the Coordinate System for the specified coordinate (minimum match is active for argument type). By default it finds the first coordinate, but if there is more than one (can happen for linear coordinates), you can specify which. It returns a dictionary with 'return', 'pixel', and 'world' as keys. The associated value of 'return' is a boolean indicating if the specified coordinate was found. The values of 'pixel' and 'world' are arrays indicating the indices of the associated pixel and world axes, respectively, of the specified coordinate. If the coordinate does not exist, these arrays will be empty.

See also the function axesmap which returns the mapping between pixel and world axes.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Type of coordinate to find: direction, stokes, spectral, linear, or tabular</td>
<td>allowed: string</td>
</tr>
<tr>
<td>which</td>
<td>Which coordinate if more than one</td>
<td>allowed: int</td>
</tr>
</tbody>
</table>

Returns

record

Example

```
print "\t----\t findcoordinate Ex 1 \t----"
csys=cs.newcoordsys(direction=T)
rtn=cs.findcoordinate('direction')
print rtn
#{'world': [0L, 1L], 'pixel': [0L, 1L]}
print 'pixel, world axes =', rtn['pixel'], rtn['world']
# pixel, world axes = [0 1] [0 1]
1.1.3 Apply relativistic Doppler shift to a list of frequencies

Description

This function converts frequencies to frequencies by applying a relativistic Doppler shift: \( \text{fout} = \text{fin} \times \sqrt{(1.-v/c)/(1.+v/c)} \).

The input frequencies are specified via a vector of numeric values and a specified unit (frequnit). If you don’t give a frequency unit, it is assumed that the units are those given by function coordsys units() for the spectral coordinate.

This function does not make any frame conversions (e.g. LSR to BARY).

This function fails if there is no spectral coordinate in the Coordinate System. See also function frequencytovelocity.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Frequencies to convert</td>
<td>allowed: doubleArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>frequnit</td>
<td>Unit of input frequencies. Default is unit of the spectral coordinate.</td>
<td>allowed: string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>velocity</td>
<td>Velocity</td>
<td>allowed: any</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Returns

doubleArray

Example
```python
ia.open('M100line.image')
mycs = ia.coordsys()
ia.close()

mycs.frequencytofrequency(value=[115271201800.0], frequnit='Hz', velocity='1000km/s')
results in
array([114887337607.0])

Let's see if this is correct
    print 115271201800.0*sqrt((1.-1000000./299792458.0)/(1.+1000000./299792458.0))
Result: 1.14887337607e+11
```
Description

This function converts frequencies to velocities. The input frequencies are specified via a vector of numeric values and a specified unit (frequnit). If you don’t give a frequency unit, it is assumed that the units are those given by function coordsys units() for the spectral coordinate.

This function does not make any frame conversions (e.g. LSR to BARY) but you can specify the velocity doppler definition via the doppler argument (see image summary() for possible values).

The velocities are returned in a vector for which you specify the units (velunit - default is km/s).

This function will return a fail if there is no spectral coordinate in the Coordinate System. See also function velocitytofrequency.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Frequency to convert</td>
<td>doubleArray</td>
<td></td>
<td></td>
</tr>
<tr>
<td>frequnit</td>
<td>Unit of input frequencies. Default is unit of the spectral coordinate.</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doppler</td>
<td>Velocity doppler definition</td>
<td>string</td>
<td></td>
<td>radio</td>
</tr>
<tr>
<td>velunit</td>
<td>Unit of output velocities</td>
<td>string</td>
<td></td>
<td>km/s</td>
</tr>
</tbody>
</table>

Returns
doubleArray
Example

```python
# print "\t----\t frequencytovelocity Ex 1 \t----" 
im = ia.fromshape(shape=[10,10,10])
csys = ia.coordsys()
rt = csys.findcoordinate('spectral')  # Find spectral axis
pa = rt['pixel']
wa = rt['world']
pixel = csys.referencepixel();  # Use reference pixel for non-spectral
nFreq = ia.shape()[pa];  # Length of spectral axis
freq = [];
for i in range(nFreq):
    pixel[pa] = i  # Assign value for spectral axis of pixel coordinate
    w = csys.toworld(value=pixel, format='n')  # Convert pixel to world
    freq.append(w['numeric'][wa]);  # Fish out frequency
print "freq=", freq
vel = csys.frequencytovelocity(value=freq, doppler='optical', velunit='km/s')
print "vel=", vel
```

In this example, we find the optical velocity in km/s of every pixel along the spectral axis of our image. First we obtain the Coordinate System from the image. Then we find which axis of the Coordinate System (image) pertain to the spectral coordinate. Then we loop over each pixel of the spectral axis, and convert a pixel coordinate (one for each axis of the image) to world. We obtain the value for the spectral axis from that world vector, and add it to the vector of frequencies. Then we convert that vector of frequencies to velocity.
coordsys.fromrecord.html

**coordsys.fromrecord - Function**

1.1.3 Fill Coordinate System from a record

**Description**

You can convert a Coordinate System to a record (torecord). This function (fromrecord) allows you to set the contents of an existing Coordinate System from such a record. In doing so, you overwrite its current contents.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>record</td>
<td>Record containing Coordinate System</td>
</tr>
<tr>
<td>allowed:</td>
<td>record</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

```
bool
```

**Example**

```python
""
#
print "	----	 fromrecord Ex 1 	----"
csys = cs.newcoordsys(direction=T, stokes="I Q")
print csys.ncoordinates()
#2
r = csys.torecord()
cs2 = cs.newcoordsys()
print cs2.ncoordinates()
#0
cs2.fromrecord(r)
print cs2.ncoordinates()
#2
```

432
#
##
Description

Each axis associated with the Coordinate System has a reference value, reference pixel and an increment (per pixel). These are used in the mapping from pixel to world coordinate.

This function returns the increment (in world axis order). You can recover the increments either for all coordinates (leave type unset) or for a specific coordinate type (minimum match of the allowed types will do). If you ask for a non-existent coordinate an exception is generated.

See the discussion regarding the formatting possibilities available via argument format.

You can set the increment with function setincrement.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>Format string from combination of &quot;n&quot;, &quot;q&quot;, &quot;s&quot;, &quot;m&quot;</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>n</td>
</tr>
<tr>
<td>type</td>
<td>Coordinate type: &quot;direction&quot;, &quot;stokes&quot;, &quot;spectral&quot;, &quot;linear&quot;, &quot;tabular&quot;. Leave empty for all.</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

Returns

record

Example

""QUI" #
print "\t----\t increment Ex 1 \t----"
csys=cs.newcoordsys(direction=T,spectral=T)
print csys.increment(format=’q’)
#{’quantity’: {’*1’: {’unit’: "’", ’value’: -1.0},
#
’*2’: {’unit’: "’", ’value’: 1.0},
#
’*3’: {’unit’: ’Hz’, ’value’: 1000.0}}}
print csys.increment(format=’n’)
#{’numeric’: [-1.0, 1.0, 1000.0]}
print csys.increment(format=’n’, type=’spectral’)
#{’numeric’: [1000.0]}
#
"""

435


Recover the linear transform matrix

**Description**

Recover the linear transform component for the specified coordinate type. You can set the linear transform with function setlineartransform.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Coordinate type: &quot;direction&quot;, &quot;stokes&quot;, &quot;spectral&quot;, &quot;linear&quot;, &quot;tabular&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>string allowed: string Default:</td>
</tr>
</tbody>
</table>

**Returns**

variant

**Example**

```python

#"
# print "\t----\t lineartransform Ex 1 \t----"
csys=cs.newcoordsys(direction=T,linear=3)
csys.lineartransform('dir') # 2 x 2
# [(1.0, 0.0), (0.0, 1.0)]
csys.lineartransform('lin') # 3 x 3
# [(1.0, 0.0, 0.0), (0.0, 1.0, 0.0), (0.0, 0.0, 1.0)]
# ""
```

436
coordsys.names.html

coordsys.names - Function

[1.1.3] Recover the names for each axis

Description

Each axis associated with the Coordinate System has a name (they don’t mean anything fundamental). This function returns those names in world axis order. You can recover the names either for all coordinates (leave type unset) or for a specific coordinate type (minimum match of the allowed types will do). If you ask for a non-existent coordinate an exception is generated. You can set the names with function setnames.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
</tr>
<tr>
<td>allowed:</td>
</tr>
<tr>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

stringArray

Example

""
#
print "\t----\t names Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
n = csys.names()
print n[0]
#Right Ascension
print n[1]
#Declination
""
print n[2]
#Frequency
print cs.names('spec')
#Frequency
#
"""
coordsys.naxes.html

coordsys.naxes - Function

Recover the number of axes

Description

Find the number of axes in the Coordinate System. You may find the number of world or pixel axes; these are generally the same and general users can ignore the distinction. See the discussion about pixel and world axis ordering.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>world</td>
<td>Find number of world or pixel axes ?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>true</td>
</tr>
</tbody>
</table>

Returns

int

Example

```
# print "\t----\t naxes Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
n = csys.naxes(T)
print n
#3 # 2 direction axes, 1 spectral
n = csys.naxes(F)
print n
#3
# """
```

439
coordsys.ncoordinates.html

**coordsys.ncoordinates - Function**

1.1.3 Recover the number of coordinates in the Coordinate System

**Description**

This function recovers the number of coordinates in the Coordinate System.

**Arguments**

**Returns**

int

**Example**

```python
""
#
print "\t----\t ncoordinates Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
print csys.ncoordinates()
#2
cs2 = cs.newcoordsys(linear=4)
print cs2.ncoordinates()
#1
#
""
```
coordsys.observer.html

coordsys.observer - Function

1.1.3 Return the name of the observer

Description

This function returns the name of the observer. You can set it with the function setobserver.

Arguments

Returns

string

Example

""
#
print "\t----\t observer Ex 1 \t----"
csys = cs.newcoordsys()
print csys.observer()
#Karl Jansky
#
""
coordsys.projection.html

**coordsys.projection** - Function

[1.1.3] Recover the direction coordinate projection

**Description**

If the Coordinate System contains a direction coordinate, this function can be used to recover information about the projection. For discussion about celestial coordinate systems, including projections, see the papers by Mark Calabretta and Eric Greisen. The initial draft from 1996 (implemented in CASA. Background information can be found [here](#)).

What this function returns depends upon the value you assign to *type*.

- **type=unset**. In this case (the default), the actual projection type and projection parameters are returned in a record with fields *type* and *parameters*, respectively.
- **type='all'**. In this case, a vector of strings containing all of the possible projection codes is returned.
- **type=code**. If you specify a valid projection type code (see list by setting *type*='all') then what is returned is the number of parameters required to describe that projection (useful in function setprojection).

You can change the projection with setprojection.

If the Coordinate System does not contain a direction coordinate, an exception is generated.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
</table>
| type   | Type of projection. Defaults to current projection.  
| allowed: string |
| Default: |

**Returns**

- record

**Example**

443
We first recover the projection type and parameters from the direction coordinate. Then we find the list of all possible projection types. Finally, we recover the number of parameters required to describe the 'ZPN' projection.
coordsys.referencecode.html

**coordsys.referencecode - Function**

1.1.3 Return specified reference code

**Description**

This function returns the reference code for all, or the specified coordinate type. Examples of the reference code are B1950 and J2000 for direction coordinates, or LSRK and BARY for spectral coordinates. If `type` is left unset, then a vector of strings is returned, one code for each coordinate type in the Coordinate System. If you specify `type` then select from 'direction', 'spectral', 'stokes', and 'linear' (the first two letters will do). However, only the first two coordinate types will return a non-empty string. If the Coordinate System does not contain a coordinate of the type you specify, an exception is generated. The argument `list` is ignored unless you specify a specific `type`. If `list=T`, then this function returns the list of all possible reference codes for the specified coordinate type. Otherwise, it just returns the actual code current set in the Coordinate System. The list of all possible types is returned as a record (it is actually generated by the listcodes function in the measures system). This record has two fields. These are called 'normal' (containing all normal codes) and 'extra' (maybe empty, with all extra codes like planets). You can set the reference code with setreferencecode.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Coordinate type: &quot;direction&quot;, &quot;stokes&quot;, &quot;spectral&quot;, &quot;linear&quot;, &quot;tabular&quot;. Leave empty for all. allowed: string Default:</td>
</tr>
<tr>
<td>list</td>
<td>List all possibilities? allowed: bool Default: false</td>
</tr>
</tbody>
</table>

**Returns**

stringArray
Example

In this example we first get the list of all possible reference codes for a direction coordinate. Then we get the actual reference code for the direction coordinate in our Coordinate System.
Recover the reference pixel

Description

Each axis associated with the Coordinate System has a reference value, reference pixel and an increment (per pixel). These are used in the mapping from pixel to world coordinate.

This function returns the reference pixel (in pixel axis order). You can recover the reference pixel either for all coordinates (leave type unset) or for a specific coordinate type (minimum match of the allowed types will do). If you ask for a non-existent coordinate an exception is generated.

You can set the reference pixel with function setreferencepixel.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Coordinate type: &quot;direction&quot;, &quot;stokes&quot;, &quot;spectral&quot;, &quot;linear&quot;, &quot;tabular&quot;. Leave empty for all.</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
</tbody>
</table>

Returns record

Example

```python
print "\t----\t referencepixel Ex 1 \t----"
csys = cs.newcoordsys(spectral=T, linear=2)
csys.setreferencepixel([1.0, 2.0, 3.0])
print csys.referencepixel()
#{'ar_type': 'absolute', 'pw_type': 'pixel', 'numeric': array([ 1., 2., 3.])}
```
print csys.referencepixel('lin')
#{'ar_type': 'absolute', 'pw_type': 'pixel', 'numeric': array([ 2., 3.])}
#
"""
coordsys.referencevalue.html

**coordsys.referencevalue - Function**

1.1.3 Recover the reference value

**Description**

Each axis associated with the Coordinate System has a reference value, reference pixel and an increment (per pixel). These are used in the mapping from pixel to world coordinate.

This function returns the reference value (in world axis order). You can recover the reference value either for all coordinates (leave type unset) or for a specific coordinate type (minimum match of the allowed types will do). If you ask for a non-existent coordinate an exception is generated.

See the discussion regarding the formatting possibilities available via argument format.

You can set the reference value with function setreferencevalue.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>Format string. Combination of &quot;n&quot;, &quot;q&quot;, &quot;s&quot;, &quot;m&quot;</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: n</td>
</tr>
<tr>
<td>type</td>
<td>Coordinate type: &quot;direction&quot;, &quot;stokes&quot;, &quot;spectral&quot;, &quot;linear&quot;, &quot;tabular&quot;. Leave empty for all.</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```

```
print "\t----\t referencevalue Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
print csys.referencevalue(format='q')
#
# 'ar_type': 'absolute',
# 'pw_type': 'world',
# 'quantity': {'*1': {'unit': '', 'value': 0.0},
#              '*2': {'unit': '', 'value': 0.0},
#              '*3': {'unit': 'Hz', 'value': 1415000000.0}}
print csys.referencevalue(format='n')
#
# 'ar_type': 'absolute',
# 'numeric': array([ 0.00000000e+00, 0.00000000e+00, 1.41500000e+09]),
# 'pw_type': 'world'}
print csys.referencevalue(format='n', type='spec')
#
# 'ar_type': 'absolute',
# 'numeric': array([ 1.41500000e+09]),
# 'pw_type': 'world'}
#
"
coordsys.reorder.html

**coordsys.reorder - Function**

Reorder the coordinates

### Description

This function reorders the coordinates in the Coordinate System. You specify the new order of the coordinates in terms of their old order.

### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>order</td>
<td>New coordinate order</td>
</tr>
<tr>
<td>allowed:</td>
<td>intArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

### Returns

bool

### Example

```python
"
#
print "\t----\t reorder Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T, linear=2)
print csys.coordinatetype()
#['Direction', 'Spectral', 'Linear']
csys.reorder([1,2,0]);
print csys.coordinatetype()
#['Spectral', 'Linear', 'Direction']
#
""
```
**Description**

This method transposes the axes (both world and pixel) in the coordinate system. You specify the new order of the axes in terms of their old order, so eg order=[1,0,3,2] means reorder the axes so that the zeroth axis becomes the first axis, the first axis becomes the zeroth axis, the second axis becomes the third axis, and the third axis becomes the second axis.

**Arguments**

Inputs

<table>
<thead>
<tr>
<th>Order</th>
<th>New axis order</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>intArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
csys = cstool()

# Create a coordinate system with axes, RA, Dec, Stokes, and Frequency
csys.newcoordsys(direction=T, spectral=T, stokes=['I','Q'])

# transpose the axes so that the order is RA, Dec, Frequency, and Stokes
csys.transpose(order=[0, 1, 3, 2])
```
coordsys.replace.html

coordsys.replace - Function

L.1.3 Replace a coordinate

Description

This function replaces one coordinate in the current Coordinate System by one coordinate in the given Coordinate System. The specified coordinates must have the same number of axes.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>csys</td>
<td>Coordinate System to replace from. Use coordsys' torecord() to generate required record.</td>
<td>record</td>
<td></td>
</tr>
<tr>
<td>whichin</td>
<td>Index of input coordinate (0-rel)</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>whichout</td>
<td>Index of output coordinate</td>
<td>int</td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool

Example

"""
#
print "	\t replace Ex 1 \t""
cs1 = cs.newcoordsys(direction=T, linear=1)
print cs1.coordinatetype()
#['Direction', 'Linear']
"""
cs2 = cs.newcoordsys(spectral=T)
cs1.replace (cs2.torecord(),0,1)
print cs1.coordinatetype()
#['Direction', 'Spectral']
#"""
Recover the rest frequency

Description

If the Coordinate System contains a spectral coordinate, then it has a rest frequency. In fact, the spectral coordinate can hold several rest frequencies (to handle for example, an observation where the band covers many lines), although only one is active (for velocity conversions) at a time. This function recovers the rest frequencies as a quantity vector. The first frequency is the active one.

You can change the rest frequencies with setrestfrequency.

If the Coordinate System does not contain a frequency coordinate, an exception is generated.

Arguments

Returns

Example

```python
# print "\t----\t restfrequency Ex 1 \t----"
csys = cs.newcoordsys(spectral=T)
print csys.restfrequency()
#{'value': [1420405751.7860003], 'unit': 'Hz'}
csys.setrestfrequency (value=qa.quantity([1.2e9, 1.3e9],'Hz'), which=1, append=F)
print csys.restfrequency()
#{'value': [1300000000.0, 1200000000.0], 'unit': 'Hz'}
```
In the example, the initial spectral coordinate has 1 rest frequency. Then we set it with two, nominating the second as the active rest frequency, and recover them.
Description

Some coordinates contain a reference code. Examples of reference codes are B1950 and J2000 for direction coordinates, or LSRK and BARY for spectral coordinates. When you do conversions between pixel and world coordinate, the coordinates are in the reference frame corresponding to these codes. This function allows you to specify a different reference frame which is used when converting between world and pixel coordinate (see function conversiontype to recover the conversion types). If it returns F, it means that although the conversion machines were successfully created, a trial conversion failed. This usually means the REST frame was involved which requires a radial velocity (not yet implemented). If this happens, the conversion type will be left as it was. The function fails if more blatant things are wrong like a missing coordinate, or an incorrect reference code. The list of possible reference codes can be obtained via function referencecode. With this function, you specify the desired reference code. Then, when a conversion between pixel and world is requested, an extra conversion is done to (toWorld) or from (toPixel) the specified reference frame. The summary function shows the extra conversion reference system to the right of the native reference system (if it is different) and in parentheses. Note that to convert between different spectral reference frames, you need a position, epoch and direction. The position (telescope) and epoch (date of observation), if not in your coordinate system can be set with functions settelescope and setepoch. The direction is the reference direction of the required direction coordinate in the coordinate system.

As an example, let us say you are working with a spectral coordinate which was constructed with the LSRK reference frame. You want to convert some pixel coordinates to barycentric velocities (reference code BARY).

```python
#
print "\t----\t setconversiontype Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T); # Create coordinate system
rt=csys.findcoordinate('spectral') # Find spectral coordinate
wa=rt['world']
pa=rt['pixel']
```
u = csys.units()[wa]  # Spectral unit
print csys.referencecode(type='spectral')  # Which is in LSRK reference frame
#LSRK
p = [10,20,30]  # Convert a pixel to LSRK world
w = csys.toworld(p, format='n')  # Convert a pixel to LSRK world
print 'pixel, world = ', p, w['numeric']
#pixel, world = [10, 20, 30] [21589.99816660376, 20.000112822985134, 1415030000.0]
p2 = csys.topixel(w)  # and back to pixel
print 'world, pixel = ', w['numeric'], p2
#world, pixel = [21589.99816660376, 20.000112822985134, 1415031369.0081882]
# Convert LSRK frequency to LSRK velocity
v = csys.frequencytovelocity(value=w['numeric'][wa], frequnit=u, doppler='RADIO', velunit='m/s');
print 'pixel, frequency, velocity = ', p[pa], w['numeric'][wa], v
#pixel, frequency, velocity = 30 1415030000.0 1134612.30321

You must also be aware of when this extra layer is active and when it is not.
It’s a bit nasty.

- Whenever you use toWorld, toPixel toWorldMany, or toPixelMany the layer is active.
- Whenever you use convert or convertMany the layer may be active.

Here are the rules!
It is only relevant to spectral and direction coordinates.
For the direction coordinate part of your conversion, if you request a pure world or pixel conversion it is active. Any pixel/world mix will not invoke it (because it is ill defined).

For the spectral coordinate part it is always active (only one axis so must be pixel or world).

458
• This layer is irrelevant to all functions converting between frequency and velocity, and absolute and relative. The values are in whatever frame you are working with.

The summary function lists the reference frame for direction and spectral coordinates. If you have also set a conversion reference code it also lists that (to the right in parentheses).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Reference code</th>
</tr>
</thead>
<tbody>
<tr>
<td>direction</td>
<td>allowed: string</td>
</tr>
<tr>
<td>spectral</td>
<td>Reference code</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
</tbody>
</table>

**Returns**

bool
coordsys.getconversiontype.html

coordsys.getconversiontype - Function

1.1.3 Get extra reference conversion layer (aka conversiontype).

Description

See conversiontype for more complete description.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Conversion type</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Conversion type</td>
</tr>
<tr>
<td>showconversion</td>
<td>Show the conversion layer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>allowed</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>true</td>
</tr>
</tbody>
</table>

Returns

string
coordsys.setdirection.html

**coordsys.setdirection - Function**

[1.1.3] Set direction coordinate values

**Description**

When you construct a Coordsys tool, if you include a Direction Coordinate, it will have some default parameters. This function simply allows you to replace the values of the Direction Coordinate.

You can also change almost all of those parameters (such as projection, reference value etc.) via the individual functions setreferencecode, setprojection, setreferencepixel, setreferencevalue, setincrement, and setlineartransform provided by the Coordsys tool. See those functions for more details about the formatting of the above function arguments.

Bear in mind, that if your Coordinate System came from a real image, then the reference pixel is special and you should not change it.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th><strong>refcode</strong></th>
<th>Reference code. Default is no change.</th>
<th>allowed: string</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>proj</strong></td>
<td>Projection type. Default is no change.</td>
<td>allowed: string</td>
<td>Default:</td>
</tr>
<tr>
<td></td>
<td><strong>projpar</strong></td>
<td>Projection parameters. Default is no change.</td>
<td>allowed: doubleArray</td>
<td>Default: -1</td>
</tr>
<tr>
<td></td>
<td><strong>refpix</strong></td>
<td>Reference pixel. Default is no change.</td>
<td>allowed: doubleArray</td>
<td>Default: -1</td>
</tr>
<tr>
<td></td>
<td><strong>refval</strong></td>
<td>Reference value. Default is no change.</td>
<td>allowed: variant</td>
<td>Default:</td>
</tr>
<tr>
<td></td>
<td><strong>incr</strong></td>
<td>Increment. Default is no change.</td>
<td>allowed: variant</td>
<td>Default:</td>
</tr>
<tr>
<td></td>
<td><strong>xform</strong></td>
<td>Linear transform. Default is no change.</td>
<td>allowed: variant</td>
<td>Default:</td>
</tr>
<tr>
<td></td>
<td><strong>poles</strong></td>
<td>Native poles. Default is no change.</td>
<td>allowed: variant</td>
<td>Default:</td>
</tr>
</tbody>
</table>

| Returns | **bool** |

| Example |

```python
print "\t----\t setdirection Ex 1 \t----"
csys = cs.newcoordsys(direction=T);
csys.setdirection (refcode='GALACTIC', proj='SIN', projpar=[0,0],
                     refpix=[-10,20], refval="10deg -20deg");
print csys.projection()
#{'type': 'SIN', 'parameters': array([ 0., 0.])}
print csys.referencepixel()
```
# {'ar_type': 'absolute', 'pw_type': 'pixel', 'numeric': array([-10.,  20.]})
print csys.referencevalue(format='s')
# {'ar_type': 'absolute', 'pw_type': 'world',
#  'string': array(['10.00000000 deg', '-20.00000000 deg'],
#                   dtype='|S17'))
#
# ***
coordsys.setepoch.html

coordsys.setepoch - Function

1.1.3 Set a new epoch

Description

This function sets a new epoch (supplied as an epoch measure) of the observation. You can get the current epoch with function epoch.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>New epoch measure</td>
</tr>
<tr>
<td>allowed:</td>
<td>record</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool

Example

```python
""
#
print "\t----\t setepoch Ex 1 \t----"
csys = cs.newcoordsys()
ep = csys.epoch()
print ep
#{'type': 'epoch', 'm0': {'value': 54161.766782997685, 'unit': 'd'}, 'refer': 'UTC'}
ep = me.epoch('UTC', 'today')
csys.setepoch(ep)
print csys.epoch()
#{'type': 'epoch', 'm0': {'value': 54161.766782997685, 'unit': 'd'}, 'refer': 'UTC'}
#
""
```
coordsys.setincrement.html

coordsys.setincrement - Function

Set the increment

Description

Each axis associated with the Coordinate System has a reference value, reference pixel and an increment (per pixel). These are used in the mapping from pixel to world coordinate.

This function allows you to set a new increment. You should not do this on "stokes" axes unless you are an adept or a big risk taker.

You can set the increments either for all axes (type=unset) or for just the axes associated with a particular coordinate type.

You may supply the increments in all of the formats described in the formatting discussion.

In addition, you can also supply the increments as a quantity of vector of doubles. For example qa.quantity([-1,2],’arcsec’).

You can recover the current increments with function increment.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Increments</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>any</td>
</tr>
<tr>
<td></td>
<td>variant</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>type</th>
<th>Coordinate type: &quot;direction&quot;, &quot;stokes&quot;, &quot;spectral&quot;, &quot;linear&quot;, &quot;tabular&quot;. Leave empty for all</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

bool

Example

466
In the example we first recover the increments as a vector of quantities. We then create a quantity for a new value for the spectral coordinate increment. Note we use units of kHz whereas the spectral coordinate is currently expressed in units of Hz. We then set the increment for the spectral coordinate. We then recover the increment again; you can see 4kHz has been converted to 4000Hz. We also show how to set the increment using a string interface.
coordsys.setlineartransform.html

coordsys.setlineartransform - Function

1.1.3 Set the linear transform

Description

This function set the linear transform component. For Stokes Coordinates this function will return T but do nothing. You can recover the current linear transform with function lineartransform.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Coordinate type: &quot;direction&quot;, &quot;stokes&quot;, &quot;spectral&quot;, &quot;linear&quot;, &quot;tabular&quot;. Leave empty for all.</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>string</td>
</tr>
<tr>
<td>value</td>
<td>Linear transform</td>
</tr>
</tbody>
</table>

Returns

bool

Example

""
#
print "\t----\t setlineartransform Ex 1 \t----"
csys = cs.newcoordsys(spectral=T, linear=3)
xf = csys.lineartransform('lin')
print xf
#[(1.0, 0.0, 0.0), (0.0, 1.0, 0.0), (0.0, 0.0, 1.0)]
xf[0]=list(xf[0])
xf[0][1]=0.01
#xf[0]=tuple(xf[0])
csys.setlineartransform('lin',xf)
print csys.lineartransform('lin')
#[(1.0, 0.01, 0.0), (0.0, 1.0, 0.0), (0.0, 0.0, 1.0)]
"""
coordsys.setnames - Function

Set the axis names

Description

Each axis associated with the Coordinate System has a name. It isn’t used in any fundamental way.
This function allows you to set new axis names.
You can set the names either for all axes (type=unset) or for just the axes associated with a particular coordinate type.
You can recover the current axis names with function names.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Names</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Names</td>
<td>stringArray</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>Coordinate type: &quot;direction&quot;, &quot;stokes&quot;, &quot;spectral&quot;, &quot;linear&quot;, &quot;tabular&quot; or leave empty for all</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool

Example

```
```
#['a', 'b', 'c']
csys.setnames("flying fish", 'lin')
print csys.names()
#['a', 'flying', 'fish']
#
**coordsys.setobserver - Function**

### 1.1.3 Set a new observer

**Description**

If you want to grab all the glory, or transfer the blame, this function sets a new observer of the observation. You can get the current observer with function `observer`. The observer’s name is not fundamental to the Coordinate System!

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>New observer</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>string</td>
</tr>
<tr>
<td>allowed</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
#
print "\t----\t setobserver Ex 1 \t-----"
csys = cs.newcoordsys()
print csys.observer()
#Karl Jansky
csys.setobserver('Ronald Biggs')
print csys.observer()
#Ronald Biggs
#
""""""'
```

---

473
coordsys.setprojection.html

 coordsys.setprojection - Function

1.1.3 Set the direction coordinate projection

Description

If the Coordinate System contains a direction coordinate, this function can be used to set the projection. For discussion about celestial coordinate systems, including projections, see the papers by Mark Calabretta and Eric Greisen. The initial draft from 1996 (implemented in CASA) can be found here.

You can use the function projection to find out all the possible types of projection. You can also use it to find out how many parameters you need to describe a particular projection. See Calabretta and Greisen for details about those parameters (see section 4 of their paper); in FITS terms these parameters are what are labelled as PROJP.

Some brief help here on the more common projections in astronomy.

- SIN has either 0 parameters or 2. For coplanar arrays like East-West arrays, one can use what is widely termed the NCP projection. This is actually a SIN projection where the parameters are 0 and $1/tan(\delta_0)$ where $\delta_0$ is the reference declination. Images made from the ATNF’s Compact Array with CASA will have such a projection. Otherwise, the SIN projection requires no parameters (but you can give it two each of which is zero if you wish).

- TAN is used widely in optical astronomy. It requires 0 parameters.

- ZEA (zenithal equal area) is used widely in survey work. It requires 0 parameters.

If the Coordinate System does not contain a direction coordinate, an exception is generated.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Type of projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>string</td>
</tr>
<tr>
<td>parameters</td>
<td>doubleArray</td>
</tr>
</tbody>
</table>

474
Returns
bool

Example

```
# print "\t----\t Ex setprojection 1 \t----"
im = ia.maketestimage('cena', overwrite=true)
csys = ia.coordsys()
print csys.projection()
#{'type': 'SIN', 'parameters': array([ 0., 0.])}
print csys.projection('ZEA')
#{'nparameters': 0}
csys.setprojection('ZEA')
im2 = ia.regrid('cena.zea', csys=csys.torecord(), overwrite=true)
#
```

We change the projection of an image from SIN to ZEA (which requires no parameters).
coordsys.setreferencecode.html

coordsys.setreferencecode - Function

**1.1.3 Set new reference code**

**Description**

This function sets the reference code for the specified coordinate type. Examples of reference codes are B1950 and J2000 for direction coordinates, or LSRK and BARY for spectral coordinates.

You must specify `type`, selecting from 'direction', or 'spectral' (the first two letters will do). If the Coordinate System does not contain a coordinate of the type you specify, an exception is generated.

Specify the new code with argument `value`. To see the list of possible codes, use the function referencecode (see example).

If `adjust` is T, then the reference value is recomputed. This is invariably the correct thing to do. If `adjust` is F, then the reference code is simply overwritten; do this very carefully.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Reference code</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>Coordinate type: direction or spectral</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>direction</td>
</tr>
<tr>
<td>adjust</td>
<td>Adjust reference value ?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>true</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**
In this example we first get the list of all possible reference codes for a direction coordinate. Then we set the actual reference code for the direction coordinate in our Coordinate System.

Example

```python
#
print "\t----\t Ex setreferencecode 1 \t----"
csys = cs.newcoordsys(direction=T)
clist = csys.referencecode('dir', T)  # See possibilities
print clist
#['J2000', 'JMEAN', 'JTRUE', 'APP', 'B1950', 'BMEAN', 'BTRUE', 'GALACTIC',
# 'HADEC', 'AZEL', 'AZELSW', 'AZELNE', 'AZELGEO', 'AZELSWGEO', 'AZELNEGEO',
# 'JNAT', 'ECLIPTIC', 'MECLIPTIC', 'TECLIPTIC', 'SUPERGAL', 'ITRF', 'TOPO',
# 'ICRS', 'MERCURY', 'VENUS', 'MARS', 'JUPITER', 'SATURN', 'URANUS',
# 'NEPTUNE', 'PLUTO', 'SUN', 'MOON', 'COMET']
print csys.referencecode('dir')
#J2000
csys.setreferencecode('B1950', 'dir', T)
#
#"
```

```python
In this example we first get the list of all possible reference codes for a direction coordinate. Then we set the actual reference code for the direction coordinate in our Coordinate System.
```
In this example we show how to regrid an image from J2000 to B1950. First we recover the Coordinate System into the Coordsys tool called \stf cs\'. We then set a new direction reference code, making sure we recompute the reference value. Then the new Coordinate System is supplied in the regridding process (done with an Image tool).
Description

This function sets the reference pixel and reference value to the specified values. The world coordinate can be specified in any of the formats that the output world coordinate is returned in by the toworld function. You can specify a mask (argument `mask`) indicating which pixel axes are set (T) and which are left unchanged (F). This function will refuse to change the reference location of a Stokes axis (gets you into trouble otherwise). This function can be rather useful when regridding images. It allows you to keep easily a particular feature centered in the regridded image.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixel</td>
<td>New reference pixel. Defaults to old reference pixel.</td>
<td>intArray</td>
<td>-1</td>
</tr>
<tr>
<td>world</td>
<td>New reference value. Defaults to old reference value.</td>
<td>any</td>
<td>variant -1</td>
</tr>
<tr>
<td>mask</td>
<td>Indicates which axes to center. Defaults to all.</td>
<td>boolArray</td>
<td>false</td>
</tr>
</tbody>
</table>

Returns

bool

Example

```python
#
```
print "\t----\t setreferencelocation Ex 1 \t----"
csys = cs.newcoordsys(linear=2)
print csys.referencepixel()
# [0.0, 0.0]
print csys.referencevalue()
#{'numeric': array([ 0., 0.])}
w = csys.toworld([19, 19], format='n')
shp = [128, 128]
p = [64, 64]
csys.setreferencelocation (pixel=p, world=w)
print csys.referencepixel()
# [64.0, 64.0]
print csys.referencevalue()
#{'numeric': array([ 19., 19.])}
# """
coordsys.setreferencepixel.html

coordsys.setreferencepixel - Function

Set the reference pixel

Description

Each axis associated with the Coordinate System has a reference value, reference pixel and an increment (per pixel). These are used in the mapping from pixel to world coordinate.

This function allows you to set a new reference pixel. You should not do this on "stokes" axes unless you are an adept or a big risk taker.

You can set the reference pixel either for all axes (type=unset) or for just the axes associated with a particular coordinate type.

Bear in mind, that if your Coordinate System came from a real image, then the reference pixel is special and you should not change it for Direction Coordinates.

You can recover the current reference pixel with function referencepixel.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Reference pixel</td>
</tr>
<tr>
<td>allowed:</td>
<td>doubleArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reference pixel</td>
</tr>
<tr>
<td></td>
<td>doubleArray</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| type       | Coordinate type: "direction", "stokes", "spectral", "linear", "tabular" or leave unset for all |
| allowed:   | string                                |
| Default:   |                                       |

Returns

bool

Example

"""
#
print "\t----\t setreferencepixel Ex 1 \t----"
csys = cs.newcoordsys(spectral=T, linear=2)
csys.setreferencepixel(value=[1.0, 2.0, 3.0])
print csys.referencepixel()
# [1.0, 2.0, 3.0]
csys.setreferencepixel([-1, -1], 'lin')
print csys.referencepixel()
# [1.0, -1.0, -1.0]
#
###
coordsys.setreferencevalue - Function

L1.3 Set the reference value

Description

Each axis associated with the Coordinate System has a reference value, reference pixel and an increment (per pixel). These are used in the mapping from pixel to world coordinate.

This function allows you to set a new reference value. You should not do this on "stokes" axes unless you are an adept or a big risk taker.

You may supply the reference value in all of the formats described in the formatting discussion.

You can recover the current reference value with function referencevalue.

Note that the value argument should be one of the specified possibilitioes. Especially a measure will be accepted, but will have a null effect, due to the interpretation as a generic record.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Reference value</td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: variant</td>
</tr>
<tr>
<td>type</td>
<td>Coordinate type: &quot;direction&quot;, &quot;stokes&quot;, &quot;spectral&quot;, &quot;linear&quot;, &quot;tabular&quot; or leave empty for all.</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

bool

Example

" "
#
print "\t----\t setreferencevalue Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
rv = csys.referencevalue(format='q')
print rv
# {'quantity': {'*1': {'value': 0.0, 'unit': ''},
# '*2': {'value': 0.0, 'unit': ''}, '*3': {'value': 1415000000.0, 'unit': 'Hz'}}}
rv2 = rv['quantity']['*3']
rv2['value'] = 2.0e9
print rv2
#{'value': 2000000000.0, 'unit': 'Hz'}
csys.setreferencevalue(type='spec', value=rv2)
print csys.referencevalue(format='n')
#{'numeric': array([ 0.00000000e+00, 0.00000000e+00, 2.00000000e+09])}
#
# To set a new direction reference value, the easiest way, given a
direction measure dr would be:
dr = me.direction('j2000', '30deg', '40deg')
# SHOULD BE SIMPLIFIED!!!
newrv=csys.referencevalue(format='q')
newrv['quantity']['*1']=dr['m0']
newrv['quantity']['*2']=dr['m1']
csys.setreferencevalue(value=newrv)
print csys.referencevalue(format='q')
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'quantity': {'*1': {'value': 1800.0, 'unit': ''},
# '*2': {'value': 2399.9999999999995, 'unit': ''},
# '*3': {'value': 1415000000.0, 'unit': 'Hz'}}
#
"""
Description

If the Coordinate System contains a spectral coordinate, then it has a rest frequency. In fact, the spectral coordinate can hold several rest frequencies (to handle for example, an observation where the band covers many lines), although only one is active (for velocity conversions) at a time. This function allows you to set new rest frequencies. You can provide the rest frequency as a quantity, or as a quantity string, or a double (units of current rest frequency assumed).

You specify whether the list of frequencies will be appended to the current list or whether it will replace that list. You must select which of the frequencies will become the active one. By default its the first in the list. The index refers to the final list (either appended or replaced).

You can recover the current rest frequencies with restfrequency. If the Coordinate System does not contain a frequency coordinate, an exception is generated.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>New rest frequencies</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>which</td>
<td>Which is the active rest frequency</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>append</td>
<td>Append this list or overwrite?</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

Returns

bool

Example
print "\t----\t setrestfrequency Ex 1 \t----"
csys = cs.newcoordsys(spectral=T)
print csys.restfrequency()  #{'value': array([ 1.42040575e+09]), 'unit': 'Hz'}
csys.setrestfrequency(qa.quantity('1.4GHz'))
print csys.restfrequency()  #{'value': array([ 1.40000000e+09]), 'unit': 'Hz'}
csys.setrestfrequency(1.3e9)
print csys.restfrequency()  #{'value': array([ 1.30000000e+09]), 'unit': 'Hz'}
csys.setrestfrequency (value=[1.2e9, 1.3e9], which=1)
print csys.restfrequency()  #{'value': array([ 1.30000000e+09, 1.20000000e+09]), 'unit': 'Hz'}
csys.setrestfrequency (qa.quantity([1,2],'GHz'), which=3, append=T)
print csys.restfrequency()  #{'value': array([ 2.00000000e+09, 1.20000000e+09, 1.30000000e+09, 1.00000000e+09]), 'unit': 'Hz'}
csys.setrestfrequency("1.4E9Hz 1667MHz")
print csys.restfrequency()  #{'value': array([ 1.40000000e+09, 1.66700000e+09]), 'unit': 'Hz'}
coordsys.setspectral.html

coordsys.setspectral - Function

1.1.3 Set tabular values for the spectral coordinate

Description

When you construct a Coordsys tool, if you include a Spectral Coordinate, it will be linear in frequency. This function allows you to replace the Spectral Coordinate by a finite table of values. Coordinate conversions between pixel and world are then done by interpolation.

You may specify either a vector of frequencies or velocities. If you specify frequencies, you can optionally specify a (new) reference code (see function setreferencecode for more details) and rest frequency (else the existing ones will be used).

If you specify velocities, you can optionally specify a (new) reference code and rest frequency (else the existing ones will be used). You must also give the doppler type (see function summary for more details). The velocities are then converted to frequency for creation of the Spectral Coordinate (which is fundamentally described by frequency).

You may specify the rest frequency as a Quantum or a double (native units of Spectral Coordinate used).

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>refcode</td>
<td>Reference code. Leave unset for no change.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>restfreq</td>
<td>Rest frequency. Leave unset for no change.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>frequencies</td>
<td>Vector of frequencies. Leave unset for no change.</td>
<td>any</td>
<td>variant 1GHz</td>
</tr>
<tr>
<td>doppler</td>
<td>Doppler type. Leave unset for no change.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>velocities</td>
<td>Vector of velocities types. Leave unset for no change.</td>
<td>any</td>
<td>variant 1km/s</td>
</tr>
</tbody>
</table>
Returns

bool

Example

print "\t----\t setspectral Ex 1 \t----"
csys = cs.newcoordsys(spectral=T);
f1 = [1,1.01,1.03,1.4]
fq = qa.quantity(f1, 'GHz')
csys.setspectral(frequencies=fq)
v = csys.frequencytovelocity(f1, 'GHz', 'radio', 'km/s')
print 'v=', v  #v= [88731.317461076716, 86620.706055687479, 82399.483244909003, 4306.8612455073862]
vq = qa.quantity(v, 'km/s')
csys.setspectral(velocities=vq, doppler='radio')
f2 = csys.velocitytofrequency(v, 'GHz', 'radio', 'km/s')
print 'f1 = ', f1  #f1 = [1, 1.01, 1.03, 1.3999999999999999]
print 'f2 = ', f2  #f2 = [1.0, 1.01, 1.03, 1.3999999999999999]

We make a linear Spectral Coordinate. Then overwrite it with a list of frequencies. Convert those values to velocity, then overwrite the coordinate starting with a list of velocities. Then convert the velocities to frequency and show we get the original result.
coordsys.setstokes - Function

Set the Stokes types

Description

If the Coordinate System contains a Stokes Coordinate, this function allows you to change the Stokes types defining it. If there is no Stokes Coordinate, an exception is generated.
See the coordsys constructor to see the possible Stokes types you can set.
You can set the Stokes types with function setstokes.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>stokes</td>
<td>Stokes types</td>
</tr>
<tr>
<td></td>
<td>allowed:  stringArray</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

bool

Example

```python
###
# print "\t--\t setstokes Ex 1 \t--"
csys = cs.newcoordsys(stokes="I V")
print csys.stokes()
#['I', 'V']
csys.setstokes("XX RL")
print csys.stokes()
#['XX', 'RL']
#
###
```
coordsys.settabular.html

coordsys.settabular - Function

1.1.3 Set tabular values for the tabular coordinate

Description

When you construct a Coordsys tool, if you include a Tabular Coordinate, it will be linear. This function allows you to replace the Tabular Coordinate by a finite table of values. Coordinate conversions between pixel and world are then done by interpolation (or extrapolation beyond the end). The table of values must be at least of length 2 or an exception will occur.

You may specify a vector of pixel and world values (in the current units of the Tabular Coordinate). These vectors must be the same length. If you leave one of them unset, then the old values are used, but again, ultimately, the pixel and world vectors must be the same length.

The new reference pixel will be the first pixel value. The new reference value will be the first world value.

Presently, there is no way for you to recover the lookup table once you have set it.

If you have more than one Tabular Coordinate, use argument which to specify which one you want to modify.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Vector of (0-rel) pixel values. Default is no change.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixel</td>
<td>allowed: doubleArray</td>
</tr>
<tr>
<td>world</td>
<td>Default: -1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>which</th>
<th>Which Tabular coordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed: int</td>
<td></td>
</tr>
<tr>
<td>Default: 0</td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool
Example

```python
# print "\t----\t settabular Ex 1 \t----"

csys = cs.newcoordsys(tabular=T);
print csys.settabular (pixel=[1,10,15,20,100], world=[10,20,50,100,500])
#True
#
#
```

We make a linear Tabular Coordinate. Then overwrite it with a non-linear list of pixel and world values.
coordsys.settelescope.html

**coordsys.settelescope - Function**

1.1.3 Set a new telescope

**Description**

This function sets a new telescope of the observation. The telescope position may be needed for reference code conversions; this is why it is maintained in the Coordinate System. So it is fundamental to the Coordinate System and should be correct.

You can find a list of the observatory names known to CASA with the Measures obslist function.

You can get the current telescope with function telescope.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>New telescope</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
###
# print "\t----\t settelescope Ex 1 \t----"
csys = cs.newcoordsys()
print csys.telescope()
#ATCA
csys.settelescope('VLA')
print csys.telescope()
#VLA
```

493
csys.settelescope('The One In My Backyard')
#Tue Mar 6 21:41:24 2007   WARN coordsys::settelescope:
#This telescope is not known to the casapy system
#You can request that it be added
print me.obslist()
#ALMA ARECIBO ATCA BIMA CLRO DRAO DWL GB GBT GMRT IRAM PDB IRAM_PDB
# JCMT MOPRA MOST NRAO12M NRAO_GBT PKS SAO SMA VLA VLBA WSRT
#
"""
**coordsys.setunits - Function**

Set the axis units

**Description**

Each axis associated with the Coordinate System has a unit. This function allows you to set new axis units.

You can set the units either for all axes (type=unset) or for just the axes associated with a particular coordinate type.

In general, the units must be consistent with the old units. When you change the units, the increment and reference value will be adjusted appropriately.

However, for a linear or tabular coordinate, and only when you specify type='linear' or type='tabular' (i.e. you supply units only for the specified linear of tabular coordinate), and if you set overwrite=T, you can just overwrite the units with no further adjustments. Otherwise, the overwrite argument will be silently ignored. Use argument which to specify which coordinate if you have more than one of the specified type.

You can recover the current axis units with function units.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Units</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Units</td>
<td>stringArray</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>Coordinate type: ”direction”, ”stokes”, ”spectral”, ”linear”, ”tabular” or leave unset for all.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite linear or tabular coordinate units?</td>
<td>bool</td>
<td></td>
</tr>
<tr>
<td>which</td>
<td>Which coordinate if more than one of same type. Default is first.</td>
<td>int</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

495
Example

```python
print "\t----\t setunits Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
csys.summary()
csys.setunits(value="deg rad mHz");
csys.summary()
```

coordsys.stokes - Function

1.1.3 Recover the Stokes types

Description

If the Coordinate System contains a Stokes Coordinate, this function recovers the Stokes types defining it. If there is no Stokes Coordinate, an exception is generated.
You can set the Stokes types with function setstokes.

Arguments

Returns

stringArray

Example

    """
    #
    print "\t----\t stokes Ex 1 \t----"
    csys = cs.newcoordsys(stokes=['I','V'])
    print csys.stokes()
    ['#I', 'V']
    csys = cs.newcoordsys(stokes='Q U')
    print csys.stokes()
    ['#Q', 'U']
    #
    """
coordsys.summary.html

coordsys.summary - Function

1.1.3 Summarize basic information about the Coordinate System

Description

This function summarizes the information contained in the Coordinate System. For spectral coordinates, the information is listed as a velocity as well as a frequency. The argument doppler allows you to specify what doppler convention it is listed in. You can choose from radio, optical and beta. Alternative names are z for optical, and relativistic for beta. The default is radio. The definitions are

- radio: $1 - F$
- optical: $-1 + 1/F$
- beta: $(1 - F^2)/(1 + F^2)$

where $F = \nu/\nu_0$ and $\nu_0$ is the rest frequency. If the rest frequency has not been set in your image, you can set it with the function setrestfrequency. These velocity definitions are provided by the measures system via the Doppler measure (see example).

If you set list=F, then the summary will not be written to the global logger. However, the return value will be a vector of strings holding the summary information, one string per line of the summary.

For direction and spectral coordinates, the reference frame (e.g. J2000 or LSRK) is also listed. Along side this, in parentheses, will be the conversion reference frame as well (if it is different from the native reference frame). See function setconversion to see what this means.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th>List velocity information with this doppler definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>doppler</td>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td>RADIO</td>
</tr>
<tr>
<td>list</td>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td>true</td>
</tr>
</tbody>
</table>
Returns
stringArray

Example

""
#
print "\t\t----\t summary Ex 1 \t----"
d = me.doppler('beta')
print me.listcodes(d)
# [normal=RADIO Z RATIO BETA GAMMA OPTICAL TRUE RELATIVISTIC, extra=]
csys = cs.newcoordsys(direction=T, spectral=T)
print csys.summary(list=F)
#
#Direction reference : J2000
#Spectral reference : LSRK
#Velocity type : RADIO
#Rest frequency : 1.42041e+09 Hz
#Telescope : ATCA
#Observer : Karl Jansky
#Date observation : 2007/07/14/04:49:31
#
#Axis Coord Type Name Proj Coord value at pixel Coord incr Units
#-----------------------------------------------
#0 0 Direction Right Ascension SIN 00:00:00.000 0.00 -6.000000e+01 arcsec
#1 0 Direction Declination SIN +00.00.00.000 0.00 6.000000e+01 arcsec
#2 1 Spectral Frequency 1.415e+09 0.00 1.000000e+03 Hz
#    Velocity 1140.94 0.00 -2.110611e-01 km/s
#
#""
coordsys.telescope.html

**coordsys.telescope - Function**

1.1.3 Return the telescope

**Description**

This function returns the telescope contained in the Coordinate System as a simple string.

The telescope position may be needed for reference code conversions; this is why it is maintained in the Coordinate System.

The conversion from string to position is done with Measures observatory. The example shows how.

**Arguments**

**Returns**

string

**Example**

```python
""
#
print "\t----\t telescope Ex 1 \t----"
csys = cs.newcoordsys()
print csys.telescope()
#ATCA
print me.observatory(csys.telescope())
#{'type': 'position', 'refer': 'ITRF',
 # 'm1': {'value': -0.5261379196128062, 'unit': 'rad'},
 # 'm0': {'value': 2.6101423190348916, 'unit': 'rad'},
 # 'm2': {'value': 6372960.2577234386, 'unit': 'm'}}
#
```

500
We get the telescope as a string.
The Measures system is used to convert from
the simple name to a position Measure.
coordsys.toabs.html

**coordsys.toabs - Function**

1.1.3 Convert relative coordinate to absolute

**Description**

This function converts a relative coordinate to an absolute coordinate. The coordinate may be a pixel coordinate or a world coordinate. If the coordinate is a pixel coordinate, it is supplied as a numeric vector. If the coordinate is a world coordinate, you may give it in all of the formats described in the [formatting discussion](#). If the coordinate value is supplied by a Coordsys tool function (e.g. toworld) then the coordinate 'knows' whether it is world or pixel (and absolute or relative). However, you might supply the value from some other source as a numeric vector (which could be world or pixel) in which case you must specify whether it is a world or pixel coordinate via the **isworld** argument.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Relative coordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
<tr>
<td>isworld</td>
<td>Is coordinate world or pixel? Default is unset.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

````
#
print "\t----\t toabs Ex 1 \t----"
```
This example uses world coordinates.

Example

---

csys = cs.newcoordsys(direction=T, spectral=T)
aw = csys.toworld([100,100,24], 's')
rw = csys.torel(aw)
aw2 = csys.toabs(rw)
print aw
#{'ar_type': 'absolute', 'pw_type': 'world',
  # 'string': array([23:53:19.77415678, '+01.40.00.84648186',
  #                   '1.41502400e+09 Hz'], dtype='|S19'))
print rw
#{'ar_type': 'relative', 'pw_type': 'world',
  # 'string': array([-6.00084720e+03 arcsec, 6.00084648e+03 arcsec',
  #                   '2.40000000e+04 Hz'], dtype='|S23'))
print aw2
#{'ar_type': 'absolute', 'pw_type': 'world',
  # 'string': array([23:53:19.77415672, '+01.40.00.84648000',
  #                   '1.41502400e+09 Hz'], dtype='|S19'))
# ***

This example uses world coordinates.

Example
This example uses pixel coordinates.
**coordsys.toabsmany.html**

**coordsys.toabsmany - Function**

1.1.3 Convert many numeric relative coordinates to absolute

**Description**

This function converts many relative coordinates to absolute. It exists so you can efficiently make many conversions (which would be rather slow if you did them all with toabs). Because speed is the object, the interface is purely in terms of numeric matrices, rather than being able to accept strings and quanta etc. like toabs can.

When dealing with world coordinates, the units of the numeric values must be the native units, given by function units.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Relative coordinates</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>isworld</td>
<td>Is coordinate world or pixel? Default is unset.</td>
<td>int</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

````
# print "\t---\t toabsmany Ex 1 \t---"
csys = cs.newcoordsys(direction=T, spectral=T)  # 3 axes
rv = csys.referencevalue();  # reference value
w = csys.torel(rv)  # make relative
```
inc = csys.increment(); # increment
off=[]
for idx in range(100):
    off.append(inc['numeric'][2]*idx) # offset for third axis
wrel = ia.makearray(0,[3,100]) # 100 conversions each of length 3
for i in range(3):
    for j in range(100):
        wrel[i][j]=w['numeric'][i]
for j in range(100):
    wrel[2][j] += off[j] # Make spectral axis values change
wabs = csys.toabsmany (wrel, T)['numeric'] # Convert
print wabs[0][0],wabs[1][0],wabs[2,0] # First absolute coordinate
# 0.0 0.0 1415000000.0
print wabs[0][99],wabs[1][99],wabs[2][99] # 100th absolute coordinate
# 0.0 0.0 1415099000.0
#
"""

This example uses world coordinates.
Convert from absolute world to pixel coordinate

Description

This function converts between world (physical) coordinate and absolute pixel coordinate (0-rel).
The world coordinate can be provided in one of four formats via the argument `world`. These match the output formats of function `toworld`.
If you supply fewer world values than there are axes in the Coordinate System, your coordinate vector will be padded out with the reference value for the missing axes. Excess values will be silently ignored.
You may supply the world coordinate in all of the formats described in the formatting discussion.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Absolute world coordinate</td>
</tr>
<tr>
<td>allowed</td>
<td>any</td>
</tr>
<tr>
<td>Default</td>
<td>variant</td>
</tr>
</tbody>
</table>

Returns

record

Example

```
#
print "\t----\t topixel Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T, stokes="I V", linear=2)
w = csys.toworld([-2,2,1,2,23,24], 'n')
print csys.topixel(w)
#{'ar_type': 'absolute', 'pw_type': 'pixel',}
```
Example

***

# print "\t----\t topixel Ex 2 \t----"
#
# csys = cs.newcoordsys (stokes="I V", linear=2)
#
# csys.toworld([0,1,2], 's')
#
# {'ar_type': 'absolute', 'pw_type': 'world',
# 'string': array(['I', '1.00000000e+00 km', '2.000000000e+00 km'],
# dtype='|S18')}
#
# csys.toworld([0,1,2], 'm')
#
# {'ar_type': 'absolute', 'pw_type': 'world',
# 'measure': {'stokes': 'I', 'linear': {'*1': {'value': 1.0, 'unit': 'km'},
# '*2': {'value': 2.0, 'unit': 'km'}}}}
#
# csys.toworld([0,1,2], 'q')
#
# {'ar_type': 'absolute', 'pw_type': 'world',
# 'quantity': {'*1': {'value': 1.0, 'unit': ''},
# '*2': {'value': 2.0, 'unit': 'km'}, '*3': {'value': 2.0, 'unit': 'km'}}
#
# ***
Example

""
#
print "\t----\t topixel Ex 3 \t----"
csys = cs.newcoordsys (spectral=T, linear=1)
print csys.toworld([0,1,2], 'q')
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'quantity': {'*1': {'value': 1415000000.0, 'unit': 'Hz'},
# '*2': {'value': 1.0, 'unit': 'km'}}}
#
"""
**Description**

This function converts many absolute world coordinates to pixel coordinates. It exists so you can efficiently make many conversions (which would be rather slow if you did them all with topixel). Because speed is the object, the interface is purely in terms of numeric matrices, rather than being able to accept strings and quanta etc. like topixel can. The units of the numeric values must be the native units, given by function units.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Absolute world coordinates</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
#
print "\t----\t topixelmany Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T) # 3 axes
r
v = csys.referencevalue(); # reference value
inc = csys.increment(); # increment
off = []
for idx in range(100):
    off.append(inc['numeric'][2] * idx) # offset for third axis
```

510
wabs = ia.makearray(0, [3,100])  # 100 conversions each of length 3
for i in range(3):
    for j in range(100):
        wabs[i][j]=rv['numeric'][i]
for j in range(100):
    wabs[2][j] += off[j]  # Make spectral axis values change
pabs = csys.topixelmany (wabs)['numeric'];  # Convert
print pabs[0][0], pabs[1][0], pabs[1][2]  # First absolute pixel coordinate
# 0.0 0.0 0.0
print pabs[0][99], pabs[1][99], pabs[2][99]  # 100th absolute pixel coordinate
# 0.0 0.0 99.0
#
Convert Coordinate System to a record

You can convert a Coordinate System to a record with this function. There is also fromrecord to set a Coordinate System from a record. These functions allow Coordsys tools to be used as parameters in the methods of other tools.

Arguments

Returns
record

Example

```python
#
print "\t----\t torecord Ex 1 \t----"
csys = cs.newcoordsys(direction=T, stokes="I Q")
rec = csys.torecord();
rec = cs2 = cs.newcoordsys();
print cs2.ncoordinates()
#0
cs2.fromrecord(rec);
print csys.ncoordinates(), cs2.ncoordinates()
#2 2
#
```
coordsys.subimage.html

**coordsys.subimage - Function**

1.1.3 delivers a coordinate origin re-referenced for a subimage

### Description

You can convert a Coordinate System to another coordinatesystem applicable to a subImage. The newshape does not matter as this is the coordinatesystem not the image except for Stokes axis; therefore you can ignore `newshape` except when your sub-image you are considering has only a section of your original Stokes axis.

### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>originshift</td>
<td>The shift value from original reference (vector of values in pixels)</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
<tr>
<td>newshape</td>
<td>The new shape of the image it will applicable to (pixel shape)</td>
</tr>
<tr>
<td>allowed:</td>
<td>intArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

### Returns

record

### Example

```
#
print "\t----\t subimage Ex 1 \t----"
ia.open('original.image')
csys = ia.coordsys()imshape=ia.shape()```
# want to make an empty sub image of the 11th channel
# keeping other reference pixel as is
refshft=[0,0,0,10]
subcoordsysrec=csys.subimage(neworigin=refshft)
imshape[3]=1
ia.fromshape(outfile='Eleventh_chan_template.image', shape=imshape, csys=subcoordsysrec)

"""
Function

1.1.3 Convert absolute coordinate to relative

Description

This function converts an absolute coordinate to a relative coordinate. The coordinate may be a pixel coordinate or a world coordinate. Relative coordinates are relative to the reference pixel (pixel coordinates) or the reference value (world coordinates) in the sense $relative = absolute - reference$.

If the coordinate is a pixel coordinate, it is supplied as a numeric vector. If the coordinate is a world coordinate, you may give it in all of the formats described in the formatting discussion.

If the coordinate value is supplied by a Coordsys tool function (e.g. toworld) then the coordinate 'knows' whether it is world or pixel (and absolute or relative). However, you might supply the value from some other source as a numeric vector (which could be world or pixel) in which case you must specify whether it is a world or pixel coordinate via the isworld argument.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Absolute coordinate</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
<tr>
<td>isworld</td>
<td>Is coordinate world or pixel? Default is unset.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
</tbody>
</table>

Returns

record

Example
This example uses world coordinates.

This example uses pixel coordinates.
coordsys.torelmany.html

_coordsys.torelmany - Function_

Convert many numeric absolute coordinates to relative

**Description**

This function converts many absolute coordinates to relative. It exists so you can efficiently make many conversions (which would be rather slow if you did them all with torel). Because speed is the object, the interface is purely in terms of numeric matrices, rather than being able to accept strings and quanta etc. like torel can.

When dealing with world coordinates, the units of the numeric values must be the native units, given by function units.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Absolute coordinates</td>
<td>any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
<td></td>
</tr>
<tr>
<td>isworld</td>
<td>Is coordinate world or pixel? Default is unset.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```
#
print "\t----\t torelmany Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)    # 3 axes
w = csys.referencevalue();                      # reference value
inc = csys.increment();                        # increment
```
off = []
for idx in range(100):
    off.append(inc['numeric'][2] * idx)  # offset for third axis
wabs = ia.makearray(0, [3,100])  # 100 conversions each of length 3
for i in range(3):
    for j in range(100):
        wabs[i][j] = w['numeric'][i]
for j in range(100):
    wabs[2][j] += off[j]  # Make spectral axis values change
wrel = cs.torelmany (wabs, T['numeric'])  # Convert
print wrel[0][0], wrel[1][0], wrel[2][0]  # First relative coordinate
# 0.0 0.0 0.0
print wrel[0][99], wrel[1][99], wrel[2][99]  # 100th relative coordinate
# 0.0 0.0 99000.0
#
""

This example uses world coordinates.
coordsys.toworld.html

coordsys.toworld - Function

This function converts between absolute pixel coordinate (0-rel) and absolute world (physical coordinate).

If you supply fewer pixel values than there are axes in the Coordinate System, your coordinate vector will be padded out with the reference pixel for the missing axes. Excess values will be silently ignored.

You may ask for the world coordinate in all of the formats described in the discussion regarding the formatting possibilities available via argument format.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Absolute pixel coordinate. Default is reference pixel.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>format</td>
<td>Format string: combination of &quot;n&quot;, &quot;q&quot;, &quot;s&quot;, &quot;m&quot;</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: n</td>
</tr>
</tbody>
</table>

Returns

record

Example

""
#
print "\t----\t toworld Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)
Example

print cs.toworld([-3,1,1], 'n')
#{'ar_type': 'absolute', 'pw_type': 'world',
 # 'numeric': array([ 3.00000051e+00, 1.0000001e+00, 1.41500100e+09])}
print cs.toworld([-3,1,1], 'q')
#{'ar_type': 'absolute', 'pw_type': 'world',
 # 'quantity': {'*1': {'value': 3.0000005076962117, 'unit': ''},
 # '*2': {'value': 1.0000000141027674, 'unit': ''},
 # '*3': {'value': 1415001000.0, 'unit': 'Hz'}}}
print cs.toworld([-3,1,1], 'm')
#{'ar_type': 'absolute', 'pw_type': 'world', 'measure':
 # {'spectral': {'radiovelocity': {'type': 'doppler', 'm0': {'value': 1140733.0762829871, 'unit': 'm/s'}, 'refer': 'RADIO'},
 # 'opticalvelocity': {'type': 'doppler', 'm0': {'value': 1145090.2316004676, 'unit': 'm/s'}, 'refer': 'OPTICAL'},
 # 'frequency': {'type': 'frequency', 'm0': {'value': 1415001000.0, 'unit': 'Hz'}, 'refer': 'LSRK'},
 # 'betavelocity': {'type': 'doppler', 'm0': {'value': 1142903.3485169839, 'unit': 'm/s'}, 'refer': 'TRUE'}},
 # 'direction': {'type': 'direction', 'm1': {'value': 0.0002908882127680503, 'unit': 'rad'},
 # 'm0': {'value': 0.00087266477368000634, 'unit': 'rad'}, 'refer': 'J2000'}}}
print cs.toworld([-3,1,1], 's')
#{'ar_type': 'absolute', 'pw_type': 'world',
 # 'string': array(['00:00:12.00000203', '+00.01.00.00000085', '1.41500100e+09 Hz'], dtype='|S19')}

Example

print "\t----\t toworld Ex 2 \t----"
csys = cs.newcoordsys (stokes="I V", linear=2)
print csys.toworld([0,1,2], 's')
#{'ar_type': 'absolute', 'pw_type': 'world',
 # 'string': array(['I', '1.00000000e+00 km', '2.00000000e+00 km'],
 # dtype='|S18')}
print csys.toworld([0,1,2], 'm')
#{'ar_type': 'absolute', 'pw_type': 'world',
 # 'measure': {'stokes': 'I', 'linear': {'*1': {'value': 1.0, 'unit': 'km'},
 # '*2': {'value': 1.0, 'unit': 'km'}}}
print csys.toworld([0,1,2], 'q')
#{'ar_type': 'absolute', 'pw_type': 'world',
 # 'quantity': {'*1': {'value': 1.0, 'unit': ''},
 # '*2': {'value': 1.0, 'unit': 'km'}}

521
Example

print "\t----\t toworld Ex 3 \t----"
csys = cs.newcoordsys (spectral=T, linear=1)
print cs.toworld([0,1,2], 'q')
#{'ar_type': 'absolute', 'pw_type': 'world',
# 'quantity': {'*1': {'value': 1415000000.0, 'unit': 'Hz'},
#               '*2': {'value': 1.0, 'unit': 'km'}}}
coordsys.toworldmany.html

coordsys.toworldmany - Function

1.1.3 Convert many absolute pixel coordinates to numeric world

Description

This function converts many absolute pixel coordinates to world coordinates. It exists so you can efficiently make many conversions (which would be rather slow if you did them all with toworld). Because speed is the object, the interface is purely in terms of numeric matrices, rather than being able to produce strings and quanta etc. like toworld can. The units of the output world values are the native units given by function units.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Absolute pixel coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>allowed: variant</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

record

Example

""
#
print "\t----\t toworldmany Ex 1 \t----"
csys = cs.newcoordsys(direction=T, spectral=T)  # 3 axes
rp = csys.referencepixel()['numeric'];        # reference pixel
pabs = ia.makearray(0,[3,100])                # 100 conversions each of length 3
for i in range(3):
    for j in range(100):
        pabs[i][j] = rp[i]
"""
for ioff in range(100):  # offset for third axis
    pabs[2][ioff] += ioff;  # Make spectral axis values change
wabs = csys.toworldmany (pabs)['numeric'];  # Convert
print wabs[0][0], wabs[1][0], wabs[2][0]  # First absolute pixel coordinate
# 0.0 0.0 1415000000.0
print wabs[0][99], wabs[1][99], wabs[2][99]  # 100th absolute pixel coordinate
# 0.0 0.0 1415099000.0
#
coordsys.type.html

**coordsys.type - Function**

1.1.3 Return the type of this tool

**Description**

This function returns the string ‘coordsys’.

**Arguments**

**Returns**

string
coordsys.units - Function

1.1.3 Recover the units for each axis

Description

Each axis associated with the Coordinate System has a unit. This function returns those units (in world axis order).
You can recover the units either for all coordinates (leave type unset) or for a specific coordinate type (minimum match of the allowed types will do). If you ask for a non-existent coordinate an exception is generated.
You can set the units with function setunits.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Coordinate type: 'direction', 'stokes', 'spectral', 'linear' or leave unset for all allowed: string</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>stringArray</td>
</tr>
</tbody>
</table>

Example

```python
# print "\t---\t units Ex 1 \t---"
csys = cs.newcoordsys(direction=T, spectral=T)
print csys.units()
#['', '', 'Hz']
print csys.units('spec')
#Hz
```
coordsys.velocitytofrequency.html

coordsys.velocitytofrequency - Function

1.1.3 Convert velocity to frequency

Description

This function converts velocities to frequencies. The input velocities are specified via a vector of numeric values, a specified unit (velunit), and a velocity doppler definition (doppler). The frequencies are returned in a vector for which you specify the units (frequnit). If you don’t give the unit, it is assumed that the units are those given by function units for the spectral coordinate. This function will return a fail if there is no spectral coordinate in the Coordinate System. See also the function frequencytovelocity.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Velocity to convert</td>
</tr>
<tr>
<td>frequnit</td>
<td>Unit of output frequencies. Default is intrinsic units.</td>
</tr>
<tr>
<td>doppler</td>
<td>Velocity doppler definition</td>
</tr>
<tr>
<td>velunit</td>
<td>Unit of input velocities</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Allowable</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>doubleArray</td>
</tr>
<tr>
<td>frequnit</td>
<td>string</td>
</tr>
<tr>
<td>doppler</td>
<td>string</td>
</tr>
<tr>
<td>velunit</td>
<td>km/s</td>
</tr>
</tbody>
</table>

Returns
doubleArray

Example

528
In this example, we find the optical velocity in km/s of every pixel along the spectral axis of our image. First we obtain the Coordinate System from the image. Then we find which axis of the Coordinate System (image) pertain to the spectral coordinate. Then we loop over each pixel of the spectral axis, and convert a pixel coordinate (one for each axis of the image) to world. We obtain the value for the spectral axis from that world vector, and add it to the vector of frequencies. Then we convert that vector of frequencies to velocity. Then we convert it back to frequency. They better agree.
coordsys.parentname.html

coordsys.parentname - Function

1.1.3 Get parent image name.

Description

This function returns the parent image name for ‘coordsys’.

Arguments

Returns

string
coordsys.setparentname.html

**coordsys.setparentname - Function**

1.1.3 Set the parent image name (normally not needed by end-users)

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>imagename</td>
<td>String named parent image</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

bool
1.1.4 imagepol - Tool

Polarimetric analysis of images
Requires: image

Synopsis

Description

Summary
An Imagepol tool provides specialized polarimetric analysis of images. Some of these things could be done with the Lattice Expression Language (LEL; see note223) but are more conveniently offered separately.

General
Before it can be used, the Imagepol tool must be attached to an image (CASA, FITS, and Miriad formats are supported) with a Stokes coordinate axis. What you can then do with your Imagepol tool depends on exactly which Stokes parameters you have in the image. You must have some combination of Stokes I, Q, U and V on the Stokes axis. These refer to total intensity, two components of linear polarization, and circular polarization, respectively. Therefore, if you ask for linear polarization and the image only has Stokes I and V, you will get an error.

The Imagepol tool functions generally return, by default, an on-the-fly Image tool as their output. In most cases, this is a “virtual” image. There are a range of different sorts of “virtual” images in CASA (see Image). But the Imagepol tool functions generally return reference Image tools. That is, these reference different pieces of the original image attached to the Imagepol tool, either directly, or as mathematical expressions (e.g. the polarized intensity). If you delete the attached image, you render your Imagepol tool and its outputs useless. If you wish, rather than return a virtual image tool, you can fill in the outfile argument of most Imagepol tool functions and write the image out to disk, associating the Image tool with the disk file.

In some of the functions, the standard deviation of the thermal noise is needed. This is for debiasing polarized intensity images or working out statistical error images. By default it is worked out for you from the attached image with outliers from the mean discarded. Since it is the thermal noise it is trying to find, it is worked out from the V, Q & U, and finally I data in that order of precedence. This is because Stokes V is much less likely to contain source signal than Stokes I. You can supply the noise level if you know it better. For example, for small images or images with few signal-free pixels, the theoretical estimate may be better.
Analysis and Display
Traditionally, when generating secondary and tertiary images (e.g. position angle, fractional polarization, rotation measure etc), one masks the output image according to some statistical test. For example, if the error in the output image is greater than some value, or the errors in the input images are greater than some value. Imagepol tools do not offer this kind of masking. It does provide you with the error images for the derived images. By using LEL when you analyze your images, you can mask the images however you want when you use them. That is, we defer the error interpretation as long as possible. Here is an example.

```python
print "\t----\t Tool level Ex 1 \t----"
potool = casac.homefinder.find_home_by_name('imagepolHome')
po = potool.create()
po.imagepoltestimage(outfile='stokes.image') # Create test image
po.close() # Close so we can illustrate opening an image
po.open('stokes.image') # Open image with Imagepol tool
lpa = po.linpolposang() # Linearly polarized position angle image
lpaerr = po.sigmalinpolposang() # Error in linearly polarized position angle image
lpa.statistics(); # Get statistics on position angle image
#viewer(mask=lpaerr.name()+'<5') # Display when p.a. error < 5 degrees
```

Display is handled via the Viewer tool. It can display and overlay combinations of raster, contour and vector representations of your data. It is common to display linear polarization data via vectors where the position angle of the vector is the position angle of the linear polarized radiation, and the amplitude of the vector is proportional to either the total polarized intensity or fractional polarized intensity. The data source of a vector display is either a Complex or a Float image. If it is a Complex image (e.g. the complex linear polarization \(Q + iU\)) then both the amplitude and the phase (position angle) are available. If it is just a Float image, then it is assumed to be the position angle and an amplitude of unity will be provided. The angular units are given by image brightness units which you can set with function setbrightnessunits. If the units are not recognized as angular, degrees are assumed. The position angle is measured positive North through East when you display a plane holding a celestial coordinate (the usual astronomical convention). For other axis/coordinate combinations, a positive position angle is measured from +x to +y in the absolute world coordinate frame. The Imagepol tool can create Complex disk images for you via functions complexlinpol (complex linear polarization), complexfraclinpol (complex
fractional linear polarization) and makecomplex (takes amp/phase or real/imag). As well as these Complex images, you can also make Float images of the linearly polarized intensity, linearly polarized position angle, and the fractional linearly polarized intensity (see below). 

Now, the Image tool cannot yet deal with Complex images (it will in the future). This means that you cannot currently do

```
#
print "\t----\t Tool level Ex 2 \t----"
po.open('stokes.image') # Open image with Imagepol tool
po.complexlinpol('clp') # Make complex image of linear polarization disk file
try:
    print "Expect SEVERE error and Exception here"
    ia.open('clp') # Error
except Exception, e:
    print "Expected exception occurred!"
po.close()
#
```

which is a bit annoying presently. However, the Viewer tool is able to read Complex images so that you are able to display them ok.

```
#
print "\t----\t Tool level Ex 3 \t----"
po.open('stokes.image') # Open image with Imagepol tool
po.complexlinpol('clp2') # Make complex image of linear polarization disk file
#viewer() # Start viewer to give access to Complex image
#
```

If you wanted to make a vector map display you would select the appropriate image in the Viewer’s data manager GUI, click ‘Vector Map’ on the right hand side and it would appear in the display. Note that the Viewer’s Vector map display capability also offers you amplitude noise debiasing and the On-The-Fly mask.

**Overview of Imagepol tool functions**

- Access to the different Stokes subimages is via functions stokes, stokesi, stokesq, stokesu, and stokesv.

- Create the standard secondary and tertiary polarization images with
  - complexfraclinpol - complex fractional linear polarization
− complexlinpol - complex linear polarization
− makecomplex - make complex image from amp/phase or real/imag
− pol - polarized quantities as specified
− linpolint - linearly polarized intensity
− linpolposang - linearly polarized position angle
− totpolint - total polarized intensity
− fraclinpol - fractional linear polarization
− fractotpol - fractional total polarization
− depolratio - linear depolarization ratio

• Create the standard secondary and tertiary polarization error images
  (simple propagation of Gaussian errors) with

  − sigma - best guess at thermal noise
  − sigmastokes - specified Stokes parameter
  − sigmastokesi - Stokes I
  − sigmastokesq - Stokes Q
  − sigmastokesu - Stokes U
  − sigmastokesv - Stokes V
  − signalinpolint - linearly polarized intensity
  − signalinpolposang - linearly polarized position angle
  − sigmatotpolint - total polarized intensity
  − sigmalignpolpol - fractional linear polarization
  − sigmalignpolint - fractional total polarization
  − sigmadepolratio - linear depolarization ratio

• You can compute Rotation Measure images via either the traditional
  techniques involving a number of different frequencies (regularly or
  irregularly spaced) with function rotationmeasure or via a new Fourier
  Technique for many regularly-spaced frequencies with function
  fourierrotationmeasure.

Methods

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>imagepoltestimage</td>
<td>Attach the Imagepol tool to a test image file</td>
</tr>
<tr>
<td>complexlinpol</td>
<td>Complex linear polarization</td>
</tr>
<tr>
<td>complexfraclinpol</td>
<td>Complex fractional linear polarization</td>
</tr>
<tr>
<td>depolratio</td>
<td>Linear depolarization ratio</td>
</tr>
<tr>
<td>close</td>
<td>Close the image tool</td>
</tr>
<tr>
<td>done</td>
<td>Close this Imagepol tool</td>
</tr>
</tbody>
</table>
Find Rotation Measure (Fourier approach)
Fractional linear polarization
Fractional total polarization
Linearly polarized intensity
Linearly polarized position angle
Make a Complex image
Open a new image with this imagepol tool
Polarized quantities
Find Rotation Measure (traditional approach)
Find best guess at thermal noise
Error in linear depolarization ratio
Error in fractional linear polarization
Error in fractional total polarization
Error in linearly polarized intensity
Error in linearly polarized position angle
Find standard deviation of specified Stokes data
Find standard deviation of Stokes I data
Find standard deviation of Stokes Q data
Find standard deviation of Stokes U data
Find standard deviation of Stokes V data
Error in total polarized intensity
Stokes
Stokes I
Stokes Q
Stokes U
Stokes V
Summarise Imagepol tool
Total polarized intensity
imagepol.imagepoltestimage - Function

1.1.4 Attach the Imagepol tool to a test image file

Description

This function can be used to generate a test image and then attach the Imagepol tool to it. The test image is 4-dimensional (RA, DEC, Stokes and Frequency). The Stokes axis holds I,Q,U and V. The source is just a constant I (if you don’t add noise all spatial pixels will be identical) and V. Q and U vary with frequency according to the specified Rotation Measure components (no attempt to handle bandwidth smearing within channels is made). The actual values of I,Q,U, and V are chosen arbitrarily otherwise (could be added as arguments if desired).

You can use this image, in particular, to explore the Rotation Measure algorithms in functions rotationmeasure and fourierrotationmeasure.

If you don’t specify the Rotation Measure, then it is chosen for you so that there is no position angle ambiguity between adjacent channels (the value will be sent to the Logger).

The noise added to the image is specified as a fraction of the total intensity (constant). Gaussian noise with a standard deviation of $\sigma \times I_{\text{max}}$ is then added to the image.

Arguments
### Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name</td>
<td>string</td>
<td><code>imagepol.iqv</code></td>
</tr>
<tr>
<td>rm</td>
<td>Rotation Measure (rad/m/m). Default is auto no-ambiguity determine.</td>
<td>doubleArray</td>
<td>0.0</td>
</tr>
<tr>
<td>pa0</td>
<td>Position angle (degrees) at zero wavelength</td>
<td>double</td>
<td>0.0</td>
</tr>
<tr>
<td>sigma</td>
<td>Fractional noise level</td>
<td>double</td>
<td>0.01</td>
</tr>
<tr>
<td>nx</td>
<td>Shape of image in x direction</td>
<td>int</td>
<td>32</td>
</tr>
<tr>
<td>ny</td>
<td>Shape of image in y direction</td>
<td>int</td>
<td>32</td>
</tr>
<tr>
<td>nf</td>
<td>Shape of image in frequency direction</td>
<td>int</td>
<td>32</td>
</tr>
<tr>
<td>f0</td>
<td>Reference frequency (Hz)</td>
<td>double</td>
<td>1.4e9</td>
</tr>
<tr>
<td>bw</td>
<td>Bandwidth (Hz)</td>
<td>double</td>
<td>128.0e6</td>
</tr>
</tbody>
</table>

### Returns

bool

### Example

```
#
print "\t imagepoltestimage Ex 1 \t"
po.imagepoltestimage(outfile='imagepoltestimage', rm=200)
po.rotationmeasure(rm='rm.out', rmmax=250)
```
ia.open('rm.out')
ia.statistics()
#viewer()
#
"""
imagepol.complexlinpol.html

imagepol.complexlinpol - Function

1.1.4 Complex linear polarization

Description

This function produces the complex linear polarization; $Q + iU$ and writes it to a disk image file.
The Image tool cannot yet handle Complex images. You must therefore write the Complex image to disk. The Viewer can display Complex images. Also the Table tool can access Complex images.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output image file name</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

bool

Example

```python
# print "\t----\t complexlinpol Ex 1 \t----"
p.open('stokes.image')
p.complexlinpol('cplx')
tb.open('cplx')
#tb.browse()
#
```
imagepol.complexfraclinpol.html

imagepol.complexfraclinpol - Function

1.1.4 Complex fractional linear polarization

Description

This function produces the complex fractional linear polarization; \((Q + iU)/I\) and writes it to a disk image file. The Image tool cannot yet handle Complex images. You must therefore write the Complex image to disk. The Viewer can display Complex images. Also the Table tool can access Complex images.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output image file name</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>allowed: string Default:</td>
</tr>
</tbody>
</table>

Returns

bool

Example

```python
#
print "\t----\t complexfraclinpol Ex 1 \t----"
p.open('stokes.image')
p.complexfraclinpol('cplx2')
tb.open('cplx2')
#tb.browse()
#```

541
imagepol.depolratio.html

**imagepol.depolratio - Function**

1.1.4 Linear depolarization ratio

**Description**

This function returns the linear depolarization ratio computed from two frequencies; this is the ratio of the fractional linear polarization at the two frequencies. Generally this is done when you have generated two images, each at a different frequency (continuum work). Thus if the fractional linear polarization images are $m_{\nu_1}$ and $m_{\nu_2}$ then the depolarization ratio is $m_{\nu_1}/m_{\nu_2}$.

This function operates with two images; the first (at frequency $\nu_1$) is the one attached to your Imagepol tool. The second (at frequency $\nu_2$) is supplied via the argument `infile`, which is a String holding the name of the image file.

In generating the depolarization ratio, you may optionally debias the linearly polarized intensity. This requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

You can get the depolarization ratio error image with function `sigmadepolratio`.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>infile</code></td>
<td>Other image</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><code>debias</code></td>
<td>Debias the linearly polarized intensity?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td><code>clip</code></td>
<td>Clip level for auto-sigma determination</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>10.0</td>
</tr>
<tr>
<td><code>sigma</code></td>
<td>Standard deviation of thermal noise. Default is auto determined.</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>-1</td>
</tr>
<tr>
<td><code>outfile</code></td>
<td>Output image file name. Default is unset.</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>
Returns
image

Example

""
#
# print "\t----\t depolratio Ex 1 \t----"
#po.open('stokes.4800')
#dpr = po.depolratio(infile='stokes.8300') # m_4800 / m_8300
#edpr = po.sigmadepolratio(infile='stokes.8300');
#dpr.done()
#edpr.done()
#
"""
imagepol.close.html

**imagepol.close - Function**

1.1.4 Close the image tool

**Description**

This function closes the imagepol tool. This means that it detaches the tool from its image file (flushing all the changes first). The imagepol tool is “null” after this change (it is not destroyed) and calling any tool function other than open will result in an error.

**Arguments**

**Returns**

bool

**Example**

```python
""
#
print "\t----\t close Ex 1 \t----"
# First create image and attach it to imagepol tool
po.imagepoltestimage('myimagepol')
po.close() # Detaches image from Imagepol tool
print "!!!EXPECT ERROR HERE!!!"
po.summary() # No image so this results in an error.
po.open('myimagepol') # Image is reattached
po.summary() # No error
po.close() #
""
```
imagepol.done - Function

1.1.4 Close this Imagepol tool

Description

This function is the same as close().

Arguments

Returns

bool

Example

""
#
print "\t----\t done Ex 1 \t----"
po.open('myimagepol')
po.done()  # Detaches image from Imagepol tool
print "!!!EXPECT ERROR HERE!!"
p.summary()  # No image so this results in an error.
po.open('myimagepol')  # Image is reattached
po.summary()  # No error
po.done()
#
"""
imagepol.fourierrotationmeasure.html

imagepol.fourierrotationmeasure - Function

1.1.4 Find Rotation Measure (Fourier approach)

Description

This function will only work if you attach the Imagepol tool (using open) to an image containing Stokes Q and U, and a regular frequency axis. It Fourier transforms the complex linear polarization (Q+iU) over the spectral axis to the rotation measure axis. Thus, if your input image contained RA/DEC/Stokes/Frequency, the output image would be RA/DEC/RotationMeasure. The Rotation Measure axis has as many pixels as the spectral axis.

This method enables you to see the polarization as a function of Rotation Measure. Its main use is when searching for large RMs. See Killeen, Fluke, Zhao and Ekers (1999, preprint) for a description of this method (or http://www.atnf.csiro.au/~nkilleen/rm.ps) and its advantages over the traditional method (rotationmeasure) of extracting the Rotation Measure.

Although you can write out the complex polarization image with the argument complex, you can’t do much with it because Image tools cannot handle complex images. Hence you can also write out the complex linear polarization image in any or all of the other forms.

The argument zerolag0 allows you to force the zero lag (or central bin) of the Rotation Measure spectrum to zero (effectively by subtracting the mean of Q and U from the Q and U images). This may avoid Gibbs phenomena from any strong low Rotation Measure signal which would normally fall into the central bin.

Arguments
Inputs

complex
Output complex linear polarization image file name. Default is unset.
allowed: string
Default:

amp
Output linear polarization amplitude image file name Default is unset.
allowed: string
Default:

pa
Output linear polarization position angle (degrees) image file name Default is unset.
allowed: string
Default:

real
Output linear polarization real image file name Default is unset.
allowed: string
Default:

imag
Output linear polarization imaginary angle image file name Default is unset.
allowed: string
Default:

zerolag0
Force zero lag to 0?
allowed: bool
Default: false

Returns
bool

Example

""
# print "\t----\t fourierrotationmeasure Ex 1 \t----"
po.imagepoltestimage(outfile='iquv.im', rm=[5.0e5,1e6], nx=8, ny=8, nf=512,
f0=1.4e9, bw=8e6)
po.fourierrotationmeasure(amp='amp')
ia.open('amp')
ia.statistics()
#viewer() # And reorder to put RM along X-axis
#
imagepol.fraclinpol.html

**imagepol.fraclinpol - Function**

### 1.1.4 Fractional linear polarization

**Description**

This function returns the fractional linear polarization: $\sqrt{Q^2 + U^2} / I$.
You may optionally debias the polarized intensity. This requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>debias</td>
<td>Debias the linearly polarized intensity ?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>clip</td>
<td>Clip level for auto-sigma determination</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>10.0</td>
</tr>
<tr>
<td>sigma</td>
<td>Standard deviation of thermal noise. Default is auto determined.</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>-1</td>
</tr>
<tr>
<td>outfile</td>
<td>Output image file name Default is unset.</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

image

**Example**

``````
print "\t---\t fraclinpol Ex 1 \t---"
po.open('stokes.image')
flp = po.fraclinpol()
flp.summary()
flp.done()
#
###
imagepol.fractotpol.html

imagepol.fractotpol - Function

1.1.4 Fractional total polarization

Description

This function returns the fractional linear polarization; \( \sqrt{Q^2 + U^2 + V^2}/I \). You may optionally debias the polarized intensity. This requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

If your image contains only Q and U, or only V, then just those components contribute to the total polarized intensity.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>debias</td>
<td>Debias the total polarized intensity ?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>clip</td>
<td>Clip level for auto-sigma determination</td>
<td>double</td>
<td>10.0</td>
</tr>
<tr>
<td>sigma</td>
<td>Standard deviation of thermal noise. Default is auto determined.</td>
<td>double</td>
<td>-1</td>
</tr>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

Returns

image

Example
print "\t----\t fractotpol Ex 1 \t----"
po.open('stokes.image')
ftp = po.fractotpol()
ftp.statistics()
ftp.done()
#
**imagepol.linpolint - Function**

1.4 Linearly polarized intensity

**Description**

This function returns the linearly polarized intensity; $\sqrt{Q^2 + U^2}$. You may optionally debias the polarized intensity. This requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>debias</td>
<td>Debias the linearly polarized intensity ?</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>clip</td>
<td>Clip level for auto-sigma determination</td>
<td>double</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>sigma</td>
<td>Standard deviation of thermal noise. Default is auto determined.</td>
<td>double</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

image

**Example**

```
print "\t----\t linpolint Ex 1 \t----"
po.open('stokes.image')
lpi = po.linpolint()
lpi.statistics()
lpi.done()
#
""
imagepol.linpolposang.html

**imagepol.linpolposang - Function**

1.1.4 Linearly polarized position angle

**Description**

This function returns the linearly polarized position angle image 
\(0.5 \tan^{-1}(U/Q)\) in degrees.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
</tr>
<tr>
<td>allowed: string</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

image

**Example**

```python
"
# print "\t----\t linpolposang Ex 1 \t----"
po.open('stokes.image')
lppa = po.linpolposang()
lppa.statistics()
lppa.done()
#
"```

556
imagepol.makecomplex.html

imagepol.makecomplex - Function

1.4 Make a Complex image

Description

This function generates a Complex image file from either a real and imaginary, or an amplitude and phase pair of images. If you give a linear position angle image for the phase, it will be multiplied by two before the real and imaginary parts are formed.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>complex</td>
<td>Output complex image file name. Must be specified.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>real</td>
<td>Input real image file name. Default is unset.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>imag</td>
<td>Input imaginary image file name. Default is unset.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>amp</td>
<td>Input amplitude image file name. Default is unset.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>phase</td>
<td>Input phase image file name. Default is unset.</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool

Example
In this example we make two complex linear polarization images which should be identical.
imagepol.open.html

**imagepol.open - Function**

1.1.4 Open a new image with this imagepol tool

**Description**

Before polarimetric analysis can commence, an image file must be attached to the imagepol tool using the open function. Also, use this function when you are finished analyzing the current image file and want to attach to another one. This function detaches the image tool from the current image file, if one exists, and reattaches it (opens) to the new image file.

The input image file may be in native CASA, FITS, or Miriad format. Look here for more information on foreign images.

The input image must have a Stokes axis. The exact collection of Stokes that the image has, determines what the Imagepol tool can compute. Stokes I, Q, U, and V refer to total intensity, two components of linear polarization, and circular polarization, respectively. Therefore, if you ask for linear polarization and the image only has Stokes I and V, you will get an error.

The input image may contain data at many frequencies. For example, the image may be a 4D image with axes RA, DEC, Stokes and Frequency (order not important) where the Frequency axis is regularly sampled. However, the image may also contain many frequencies at irregular intervals. Such an image may be created with the Image tool function imageconcat. It enables you to concatenate images along an axis, and it allows irregular coordinate values along that axis.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>image file name or image record (generated by ia.torecord())</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
</tbody>
</table>

**Returns**

bool
The \{\texttt{open}\} function first closes the old \texttt{imagefile}\ if one exists.
imagepol.pol.html

**imagepol.pol - Function**

1.1.4 Polarized quantities

**Description**

This function just packages the other specific polarization functions into one where you specify an operation with the argument `which` (can be useful for scripts). This argument can take the values:

- ‘lpi’ - linearly polarized intensity (function linpolint)
- ‘tpi’ - total polarized intensity (function totpolint)
- ‘lppa’ - linearly polarized position angle (function linpolposang)
- ‘flp’ - fractional linear polarization (function fraclinpol)
- ‘ftp’ - fractional total polarized intensity (function fractotpol)

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>Specify operation. One of ‘lpi’, ‘tpi’, ‘lppa’, ‘flp’, ‘ftp’ (case insensitive)</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>debias</td>
<td>Debias the polarized intensity ?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>clip</td>
<td>Clip level for auto-sigma determination</td>
<td>double</td>
<td>10.0</td>
</tr>
<tr>
<td>sigma</td>
<td>Standard deviation of thermal noise. Default is auto determined.</td>
<td>double</td>
<td>-1</td>
</tr>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>
Returns
image

Example

""
#
print "\t----\t pol Ex 1 \t----"
po.open('stokes.image')
lpi = po.pol('lpi')
lpi.statistics()
lpi.done()
#
""
imagepol.rotationmeasure.html

imagepol.rotationmeasure - Function

1.1.4 Find Rotation Measure (traditional approach)

Description

This function generates the rotation measure image from a collection of different frequencies. It will only work if the Imagepol tool is attached to an image containing Stokes $Q$ and $U$, and a frequency axis (regular or irregular) with at least 2 pixels. It will work out the position angle images for you. See also the fourierrotationmeasure function for a new Fourier-based approach.

Rotation Measure algorithms that work robustly are not common. The main problem is in trying to account for the $n - \pi$ ambiguity (see Leahy et al., Astronomy & Astrophysics, 156, 234 or Killeen et al.; http://www.atnf.csiro.au/~nkilleen/rm.ps).

The algorithm that this function uses is that of Leahy et al. (see Appendix A.1). But as in all these algorithms, the basic process is that for each spatial pixel, a vector of position angles (i.e. at the different frequencies) is fit to determine the rotation measure and the position angle at zero wavelength (and their errors). An image containing the number of $n - \pi$ turns that were added to the data at each spatial pixel and for which the best fit was found can be written. The reduced chi-squared image for the fits can also be written. Note that no assessment of curvature (i.e. deviation from the simple linear position angle - $\lambda^2$ functional form) is made.

Any combination of output images can be written.

The argument sigma gives the thermal noise in Stokes $Q$ and $U$. By default it is determined automatically using the image data. But if it proves to be inaccurate (maybe not many signal-free pixels), it may be specified. This is used for calculating the error in the position angles (propagation of Gaussian errors).

The argument maxpaerr specifies the maximum allowable error in the position angle that is acceptable. The default is an infinite value. From the standard propagation of errors, the error in the linearly polarized position angle is determined from the Stokes $Q$ and $U$ images (at each spatial pixel for each frequency). At each spatial pixel we do a fit to the position angle vector (i.e. at the different frequencies) to determine the rotation measure. If the position angle error for any pixel in the vector exceeds the specified value, it is dropped from the fit. The process generates an error for the fit and this is used to compute the errors in the output images.

Note that maxpaerr is not used to specify that any pixel for which the output position angle error exceeds this value should be masked out.
The argument `rmfg` is used to specify a foreground RM value. For example, you may know the mean RM in some direction out of the Galaxy, then including this can aid the algorithm by reducing ambiguity.

The argument `rmmax` specifies the maximum absolute RM that should be solved for. This quite an important parameter. If you leave it at the default, zero, no ambiguity handling will be used. So some apriori information should be supplied: this is the basic problem with rotation measure algorithms.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>rm</td>
<td>Output Rotation Measure image file name. Default is unset.</td>
<td>unset</td>
</tr>
<tr>
<td>rmerr</td>
<td>Output Rotation Measure error image file name. Default is unset.</td>
<td>unset</td>
</tr>
<tr>
<td>pa0</td>
<td>Output position angle (degrees) at zero wavelength image file name. Default is unset.</td>
<td>unset</td>
</tr>
<tr>
<td>pa0err</td>
<td>Output position angle (degrees) at zero wavelength error image file name. Default is unset.</td>
<td>unset</td>
</tr>
<tr>
<td>nturns</td>
<td>Output number of turns image file name. Default is unset.</td>
<td>unset</td>
</tr>
<tr>
<td>chisq</td>
<td>Output reduced chi squared image file name. Default is unset.</td>
<td>unset</td>
</tr>
<tr>
<td>sigma</td>
<td>Estimate of the thermal noise. Default is auto estimate.</td>
<td>auto estimate</td>
</tr>
<tr>
<td>rmfg</td>
<td>Foreground Rotation Measure (rad/m/m) to subtract.</td>
<td>0.0</td>
</tr>
<tr>
<td>rmmax</td>
<td>Maximum rotation measure (rad/m/m) to solve for. IMPORTANT TO SPECIFY.</td>
<td>0.0</td>
</tr>
<tr>
<td>maxpaerr</td>
<td>Maximum input position angle error (degrees) to allow</td>
<td>1e30</td>
</tr>
<tr>
<td>plotter</td>
<td>Name of plotter. Default is none.</td>
<td>none</td>
</tr>
<tr>
<td>nx</td>
<td>Number of plots in x direction</td>
<td>5</td>
</tr>
<tr>
<td>ny</td>
<td>Number of plots in y direction</td>
<td>5</td>
</tr>
</tbody>
</table>
Returns
bool

Example

"
#
print "\t----\t rotationmeasure Ex 1 \t----"
#im = ia.imageconcat(outfile='stokes.image',
#    infiles="im.f1 im.f2 im.f3 im.f4 im.f5", axis=4)
po.open('stokes.image')
ok = po.rotationmeasure(rm='rm', rmerr='rmerr', rmmax=800, maxpaerr=10)
#
"

Say we have 5 images, each with axes RA, DEC, Stokes, and Frequency in that order. We use the Image \tool\ to concatenate these images along the frequency axis - you have ordered them in increasing or decreasing frequency order. We then compute the Rotation Measure and Rotation Measure error images with the traditional method and write them out to disk.
imagepol.sigma.html

**imagepol.sigma - Function**

1.1.4 Find best guess at thermal noise

**Description**

This function returns the standard deviation from V, Q&U or I in that order of precedence. It is attempting to give you the best estimate of the thermal noise it can from the data. Outliers from the mean are clipped at the specified level.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Clip level for auto-sigma determination</th>
</tr>
</thead>
<tbody>
<tr>
<td>clip</td>
<td>Clip level for auto-sigma determination</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>10.0</td>
</tr>
</tbody>
</table>

**Returns**

double

**Example**

```python
""
#
print "\t----\t sigma Ex 1 \t----"
po.open('stokes.image')
sigma = po.sigma()
print "sigma=", sigma
#
""
```

567
imagepol.sigmadepolratio.html

**imagepol.sigmadepolratio - Function**

1.1.4 Error in linear depolarization ratio

**Description**

This function returns the error in the linear depolarization ratio computed from two frequencies; this is the ratio of the fractional linear polarization at the two frequencies. Generally this is done when you have generated two images, each at a different frequency (continuum work). Thus if the fractional linear polarization images are $m_1$ and $m_2$ then the depolarization ratio is $m_1/m_2$.

This function operates with two images; the first is attached to the Imagepol tool. The second is supplied via the argument `infile`, which is a String holding the name of the **image file**.

In generating the depolarization ratio, and hence its error, you may optionally debias the linearly polarized intensity. This requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

You can get the depolarization ratio image with function `depolratio`.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Unpacking</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>infile</td>
<td><code>Other image. Required input.</code></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>allowed: <code>string</code></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Default: <code>string</code></td>
<td>-</td>
</tr>
<tr>
<td>debias</td>
<td><code>Debias the linearly polarized intensity ?</code></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>allowed: <code>bool</code></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Default: <code>false</code></td>
<td>-</td>
</tr>
<tr>
<td>clip</td>
<td><code>Clip level for auto-sigma determination</code></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>allowed: <code>double</code></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Default: <code>10.0</code></td>
<td>-</td>
</tr>
<tr>
<td>sigma</td>
<td><code>Standard deviation of thermal noise. Default is auto determined.</code></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>allowed: <code>double</code></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Default: <code>-1</code></td>
<td>-</td>
</tr>
<tr>
<td>outfile</td>
<td><code>Output image file name. Default is unset.</code></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>allowed: <code>string</code></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Default: <code>string</code></td>
<td>-</td>
</tr>
</tbody>
</table>

568
Returns
image

Example

""
#
#print "\t\t\tsigm depol ratio Ex 1 \t\t\t"
#po.open('stokes.4800')
#dpr = po.depolratio('stokes.8300')
#edpr = po.sigm depol ratio('stokes.8300');
#dpr.done()
#edpr.done()
#"""
Error in fractional linear polarization

Description

This function returns the error (standard deviation) of the fractional linear polarization. This result comes from standard propagation of errors. The result is an on-the-fly Image tool as the error is signal-to-noise ratio dependent. This function requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>clip</td>
<td>Clip level for auto-sigma determination</td>
<td>double</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>Default: 10.0</td>
</tr>
<tr>
<td>sigma</td>
<td>Standard deviation of thermal noise. Default is auto determined.</td>
<td>double</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>Default: -1</td>
</tr>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

image

Example

```python
#
```
print "\t----\t sigmafracalinpol Ex 1 \t----"
po.open('stokes.image')
sigflp = po.sigmafracalinpol()
sigflp.statistics()
sigflp.done()          # free up resources
#
"""
This function returns the error (standard deviation) of the fractional total polarization. This result comes from standard propagation of errors. The result is an on-the-fly Image tool as the error is signal-to-noise ratio dependent. This function requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>clip</td>
<td>Clip level for auto-sigma determination</td>
<td>double</td>
<td>10.0</td>
</tr>
<tr>
<td>sigma</td>
<td>Standard deviation of thermal noise. Default is auto determined.</td>
<td>double</td>
<td>-1</td>
</tr>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

### Returns

image

### Example

```bash
$ $ $ $ #
```
print "\t---\t sigmafractotpol Ex 1 \t---" 
po.open('stokes.image')
sigftp = po.sigmafractotpol()
sigftp.statistics()
sigftp.done()
# 
""
imagepol.sigmaLinpoltint.html

imagepol.sigmaLinpoltint - Function

1.1.4 Error in linearly polarized intensity

Description

This function returns the error (standard deviation) of the linearly polarized intensity; $\sqrt{(Q^2 + U^2)}$. This result comes from standard propagation of statistical errors. The result is a float as the error is not signal-to-noise ratio dependent.

This function requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>clip</td>
<td>Clip level for auto-sigma determination</td>
<td>double</td>
<td>10.0</td>
</tr>
<tr>
<td>sigma</td>
<td>Standard deviation of thermal noise. Default is auto determined.</td>
<td>double</td>
<td>-1</td>
</tr>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

Returns

double

Example

"""
#
print "\t----\t sigmalinpolint Ex 1 \t----"
op.open('stokes.image')
siglpi = po.sigmalinpolint()
print "siglpi=", siglpi
#
""
imagepol.signalinpolposang.html

imagepol.signalinpolposang - Function

1.4 Error in linearly polarized position angle

Description

This function returns the error (standard deviation) of the linearly polarized position angle \(0.5 \tan^{-1}(U/Q) \sqrt{(Q^2 + U^2)}\) in degrees. This result comes from standard propagation of errors. The result is an on-the-fly Image tool as the error is signal-to-noise ratio dependent.

This function requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clip</td>
<td>Clip level for auto-sigma determination</td>
</tr>
<tr>
<td>sigma</td>
<td>Standard deviation of thermal noise. Default is auto determined.</td>
</tr>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
</tr>
</tbody>
</table>

Returns

image

Example

```
print "\t----\t sigmalinpolposang Ex 1 \t----"
po.open('stokes.image')
siglppa = po.sigmalinpolposang()
siglppa.statistics()
siglppa.done()
#
""
imagepol.sigmastokes.html

imagepol.sigmastokes - Function

1.1.4 Find standard deviation of specified Stokes data

Description

This function returns the standard deviation of the noise for the specified
Stokes. Outliers from the mean are clipped at the specified level.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>Must specify Stokes parameter. One of 'I', 'Q', 'U', 'V' (case insensitive)</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>clip</td>
<td>Clip level for auto-sigma determination</td>
<td>allowed: double</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns
double

Example

```python

#
# print "\t----\t sigmastokes Ex 1 \t----"
po.open('stokes.image')
sigq = po.sigmastokes('q', 10.0)
print "sigq=", sigq
#
```

578
imagepol.sigmastokesi - Function

1.1.4 Find standard deviation of Stokes I data

Description

This function returns the standard deviation of the noise for the Stokes I data. Outliers from the mean are clipped at the specified level.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>clip</td>
<td>Clip level for auto-sigma determination</td>
<td>double</td>
<td>10.0</td>
</tr>
</tbody>
</table>

Returns

double

Example

```python
# print "\t----\t sigmastokesi Ex 1 \t----"
p.o.open('stokes.image')
sgi = p.o.sigmastokesi(10.0)
print "sigi=", sigi
# 
```

579
imagepol.sigmastokesq - Function

1.1.4 Find standard deviation of Stokes Q data

Description

This function returns the standard deviation of the noise for the Stokes Q data. Outliers from the mean are clipped at the specified level.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>clip</td>
<td>Clip level for auto-sigma determination</td>
<td>double</td>
<td>10.0</td>
</tr>
</tbody>
</table>

Returns
double

Example

"""
#
print "\t--\t sigmastokesq Ex 1 \t--"
popen('stokes.image')
sigq = po.sigmastokesq(10.0)
print "sigq=", sigq
#
"""
imagepol.sigmastokesu.html

**imagepol.sigmastokesu - Function**

1.1.4 Find standard deviation of Stokes U data

**Description**

This function returns the standard deviation of the noise for the Stokes U data. Outliers from the mean are clipped at the specified level.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clip</td>
<td>Clip level for auto-sigma determination</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>10.0</td>
</tr>
</tbody>
</table>

**Returns**

double

**Example**

```python
###
# print "\t\t sigmastokesu Ex 1 \t----"
p0.open('stokes.image')
sigu = p0.sigmastokesu(10.0)
print "sigu=", sigu
#
###
```

581
imagepol.sigmastokesv - Function

Find standard deviation of Stokes V data

Description

This function returns the standard deviation of the noise for the Stokes V data. Outliers from the mean are clipped at the specified level.

Arguments

Inputs

<table>
<thead>
<tr>
<th>clip</th>
<th>Clip level for auto-sigma determination</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>10.0</td>
</tr>
</tbody>
</table>

Returns
double

Example

```python

""
#
print "\t----\t sigmastokesv Ex 1 \t----"
po.open('stokes.image')
sigv = po.sigmastokesv(10.0)
print "sigv="", sigv
#
""
```

582
Error in total polarized intensity

Description

This function returns the error (standard deviation) of the total polarized intensity; $\sqrt{Q^2 + U^2 + V^2}$. This result comes from standard propagation of statistical errors. The result is a float as the error is not signal-to-noise ratio dependent.

This function requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>clip</td>
<td>Clip level for auto-sigma determination</td>
<td>double</td>
<td>10.0</td>
</tr>
<tr>
<td>sigma</td>
<td>Standard deviation of thermal noise. Default is auto determined.</td>
<td>double</td>
<td>-1</td>
</tr>
</tbody>
</table>

Returns

double

Example

```python
#
print "\t----\t sigmastotpolint Ex 1 \t----"
po.open('stokes.image')
sigtspi = po.sigmastotpolint()
```
print "sigtpi=", sigtpi
#
"""
imagepol.stokes.html

**imagepol.stokes - Function**

1.1.4 Stokes

**Description**

This function returns an on-the-fly image tool containing the specified Stokes only. This interface can be useful for scripts.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>Must specify Stokes. One of 'I', 'Q', 'U', 'V' (case insensitive)</td>
</tr>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
</tr>
</tbody>
</table>

**Returns**

image

**Example**

```python
#
print "\t----\t stokes Ex 1 \t----"
po.open('stokes.image')
q = po.stokes('q')
q.statistics()
q.done()
#
```

585
imagepol.stokesi.html

imagepol.stokesi - Function

1.1.4 Stokes I

Description

This function returns an on-the-fly image tool containing Stokes I only.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

Returns

image

Example

```python
# print "\t----\t stokesi Ex 1 \t----"
po.open('stokes.image')
i = po.stokesi()
i.statistics()
i.done()
#
```
imagepol.stokesq.html

**imagepol.stokesq - Function**

### 1.1.4 Stokes Q

**Description**

This function returns an on-the-fly image tool containing Stokes Q only.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
</tr>
<tr>
<td>allowed: string</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

image

**Example**

```

""
#
print '"t----"t stokesq Ex 1 \t"""t----"
po.open('stokes.image')
qu = po.stokesq()
qu.statistics()
qu.statistics()
qu.done()
#
""
```

---

588
imagepol.stokesu.html

**imagepol.stokesu - Function**

### 1.1.4 Stokes U

**Description**

This function returns an on-the-fly image tool containing Stokes U only.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

image

**Example**

```python

###
#
print "\t----\t stokesu Ex 1 \t----"
p.open('stokes.image')
u = po.stokesu()
u.statistics()
u.done()
#
###
```
imagepol.stokesv.html

**imagepol.stokesv - Function**

### 1.1.4 Stokes V

**Description**

This function returns an on-the-fly image tool containing Stokes V only.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

image

**Example**

```python
###
#
print "\t----\t stokesv Ex 1 \t----"
po.open('stokes.image')
v = po.stokesv()
v.statistics()
v.done()
#
###
```
imagepol.summary.html

imagepol.summary - Function

1.1.4 Summarise Imagepol tool

Description

This function just lists a summary of the Imagepol tool to the logger. Currently it just summarizes the image to which the tool is attached.

Arguments

Returns

bool

Example

""
#
print "\t----\t summary Ex 1 \t----"
p0.open('stokes.image')
p0.summary()
#
#Image name : stokes.image
#Object name :
#Image type : PagedImage
#Image quantity : Intensity
#Pixel mask(s) : None
#Region(s) : None
#
#Direction reference : J2000
#Spectral reference : TOPO
#Velocity type : RADIO
""
<table>
<thead>
<tr>
<th>#Axis</th>
<th>Coord</th>
<th>Type</th>
<th>Name</th>
<th>Proj</th>
<th>Shape</th>
<th>Tile</th>
<th>Coord value at pixel</th>
<th>Coord incr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Direction</td>
<td>Right Ascension</td>
<td>SIN</td>
<td>32</td>
<td>32</td>
<td>00:00:00.000</td>
<td>16.00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Direction</td>
<td>Declination</td>
<td>SIN</td>
<td>32</td>
<td>32</td>
<td>+00.00.00.000</td>
<td>16.00</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Stokes</td>
<td>Stokes</td>
<td></td>
<td>4</td>
<td>4</td>
<td>I Q U V</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Spectral</td>
<td>Frequency</td>
<td>32</td>
<td>32</td>
<td></td>
<td>1.4e+09</td>
<td>16.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Velocity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-8.565499e+02</td>
</tr>
</tbody>
</table>
imagepol.totpolint.html

**imagepol.totpolint - Function**

1.1.4 Total polarized intensity

**Description**

This function returns the total polarized intensity; $\sqrt{Q^2 + U^2 + V^2}$. If your image contains only $Q$ and $U$, or only $V$, then just those components contribute to the total polarized intensity.

You may optionally debias the polarized intensity. This requires the standard deviation of the thermal noise. You can either supply it if you know it, or it will be worked out for you with outliers from the mean clipped at the specified level.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>debias</td>
<td>Debias the total polarized intensity ?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>clip</td>
<td>Clip level for auto-sigma determination</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>10.0</td>
</tr>
<tr>
<td>sigma</td>
<td>Standard deviation of thermal noise. Default is auto determined.</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>-1</td>
</tr>
<tr>
<td>outfile</td>
<td>Output image file name. Default is unset.</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

image

**Example**

593
#
print "\t----\t totpolint Ex 1 \t----"
op.open('stokes.image')
tpi = po.totpolint()
tpi.statistics()
tpi.done()
#
""

---

components-Module.html

1.2 components - Module

Access and manipulate components

Description

This module contains functionality to manipulate model components
The available tools in this module are

- [Componentlist] - a tool for manipulating groups of components

Example

""
#
print "\t----\t Module Ex 1 \t----"
pathname=os.environ.get("CASAPATH")
pathname=pathname.split()[0]
datapath=pathname+"/data/demo/Images/imagetestimage.fits"
ia.fromfits(outfile='testimage.im', infile=datapath, overwrite=T) # 1
hdr = ia.summary() # 2
ia.statistics() # 3
ia.close() # 4
print "Last example! Exiting..."
exit()
#
"""
1.2.1 componentlist - Tool

A tool for the manipulation of groups of components.

Requires:

Synopsis

Description

A componentlist is a tool that contains functions that manipulate components. A component is a functional representation of the sky brightness and is described in more detail in the overview of this module.

The simplest way to make a componentlist tool is to use the addcomponent Thies creates a componentlist tool with a specified component but not associated with any file.

The alternative is to use the open. This creates a componentlist tool by reading the data form an CASA table.

The simplest way to add components to a list is to use the simulate function. The components that are added to the list can then be edited using the component editor, or any of the manipulate functions like setshape or setrefdir.

Methods

- open: Construct an componentlist from the data in an CASA table.
- asciitocomponentlist: Create a componentlist from an ascii file (Not implemented yet).
- concatenate: Append components from another componentlist.
- fromrecord: Make a componentlist tool from a record.
- torecord: Convert componentlist to a record.
- remove: Remove a component from the list.
- purge: Permanently delete removed components.
- recover: Obtain removed components.
- length: Find the number of components in the list.
- indices: Return a vector of indices. (Not implemented yet)
- sort: Sort the components in a list.
- isphysical: Check if a component is physically plausible.
- sample: Sample the flux of the list in a specified direction. (Not implemented yet)
- rename: Give the list a name so it can save itself. Use close to save to disk.
- simulate: Add some simulated components to the list.
- addcomponent: Add a component to the list.
- close: Save the componentlist to disk and reset its state.
edit
start up the component editor gui (Not implemented yet)
done
delete the componentlist tool
select
mark components in the list
deselect
unmark components in the list
selected
determine which components are selected
getlabel
get the label of the specified component
setlabel
set the label of the specified components
getfluxvalue
get the flux value of the specified component
getfluxunit
get the flux unit of the specified component
getfluxpol
get the polarization representation for the flux of the specified component (Not implemented yet)
getfluxerror
get the error in the flux of the specified component
setflux
set the flux of the specified components
convertfluxunit
change (convert) the flux units of the specified components
convertfluxpol
change (convert) the polarization representation of the specified components
getrefdir
return the reference direction
getrefdirra
get the RA of the reference direction. (Not implemented yet)
getrefdirdec
get the declination of the reference direction. (Not implemented yet)
setrefdir
set the reference direction
setrefdirframe
set the reference frame for the direction
convertrefdir
convert the reference direction to a new frame
shape
return the shape type of the component
getshape
return the shape parameters the component
setshape
change the shape of the component
convertshape
change the units of the shape parameters (Not implemented yet)
spectrum
return the spectral shape of the component
setspectrum
return the spectral parameters the component
getfreq
get the reference frequency (Not implemented yet)
getfreqvalue
get the reference frequency value (Not implemented yet)
setfreq
set the reference frequency
setfreqframe
set the reference frame for the frequency
convertfrequnit
convert the reference frequency to a new unit
getcomponent
extract a component from the list.
add
add a component to the list.
replace
replace components in the list. (Not implemented yet)
summarize
summarize the specified component to the logger
iscomponentlist
is the argument a componentlist tool? (Not implemented yet)
componentlist.open - Function

1.2.1 Construct a componentlist from the data in a CASA table

Description

Use this constructor to construct a componentlist tool by reading the data from a CASA table. To ensure that this table contains all the necessary columns and to allow the table format to be enhanced in the future, it is highly recommended that the table be created using a componentlist tool. The table that contains the componentlist may be opened read-only by setting the readonly flag to True. When this is done, some of the functions in the componentlist tool cannot be used. These include the “set”, “convert”, “remove”, “replace”, “purge”, “recover”, and “sort” functions.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>The filename of the table</td>
</tr>
<tr>
<td>nomodify</td>
<td>Should the table be opened read only</td>
</tr>
<tr>
<td>log</td>
<td>Send a message to the logger</td>
</tr>
</tbody>
</table>

| allowed: | string  |
| Default: |         |
| allowed: | boolean |
| Default: | false   |
| allowed: | boolean |
| Default: | true    |

Returns

bool

Example

1To be more precise the table should have been created using the functions in the ComponentList class (C++).

597
cl.open('crux.cl');
The text file contains one line per star and each line has up to 7 logical columns containing, in order:
1. X position (Right Ascension 00 00 00.00)
2. Y position (Declination +/-00 00 00.0)
3. Major axis (Full width in arc seconds on sky)
4. Minor axis (Full width in arc seconds on sky)
5. Position Angle (E of N, degrees)
6. Star Type (-1 to 20, integer)
7. Star label (up to 24 character string)
If X and Y are not RA-DEC or DEC-RA, then the logical columns are also 7 actual columns and the units are in AIPS standard units (e.g. degrees, m/s etc.). In this case the position angle should be given as 0.0, the major axis is the width in the Y coordinate and the minor axis is the width in the X coordinate. For RA and DEC positions, the sexagesimal notation is used (e.g. HH MM SS.SSS -DD MM SS.S) for the positions and arc seconds on the sky are used for the Deltas.
The last 5 columns are not required. If the last 5 columns are not given, a value of 1 cell is assumed for the deltas.
If the position angle is not included, the default is 0 degrees.
If the star type is not included, the default type is a cross.
The default is no label string.
There are currently 22 different types of star marks.
< 0: No Mark, only the star label is printed
0: Cross
1: Ellipse
10: Five pointed star
11: Star of David
2: Box  12: Seven pointed star
3: Triangle  13: Eight pointed star
4: Diamond  14: Nine pointed star
5: Pentagon  15: Ten pointed star
6: Hexagon  16: Eleven pointed star
7: Septagon  17: Twelve pointed star
8: Octagon  18: Thirteen pointed star
9: Nine-gon  19: Fourteen pointed star
20: Cross with gap  >20: Ellipse

The Box (type=2) is different from the diamond in that the star size is the half height and width of the box dimensions. The Box and the Null (<0) are labeled at RA and Dec plus Delta RA and Delta Dec. The other marks are labeled at the right edge of the of the Rotated RA axis. The CROSS WITH GAP (type=20) has the inner third of the cross removed so the marked object is not over written.

For more information, see the AIPS help file for STARS.

Caveats:

- In AIPS, the following are supported: 1: 1900; 2: B1950; 3: J2000; 4: Galactic; 5: OHLSSON Gal.; 6: VAN TULDER Galactic; 7: Super Galactic; if > 1000 then year assumed. Currently, CASA can support 2 (refer='b1950'), 3 (refer='j2000'), 4 (refer='gal'), 7 (refer='supergal'). If you need any of the others, please contact us.

Caltech The Caltech package uses a format for specifying positions relative to an undefined position. In the documentation for `modelfit`, the format is described as follows:

Model files are text files that can be typed or printed directly; they can be modified or created using the standard text editors. A model file consists of one line for each component of the model, with up to 7 numbers on each line (in free format):

1. Component flux density (Jy)
2. Distance of center of component from origin (milliarcsec), "radius"
3. Position angle of center of component with respect to the origin (degrees, North through East), "theta"
4. Major axis of component (milliarcsec), "axis"
5. Axial ratio (minor/major, i.e. < 1), "ratio"
6. Position angle of major axis (degrees, North through East), "phi"

7. Type:
   - 0 or 1: elliptical Gaussian (major axis is FWHM) or delta-function (major axis = 0)
   - 2: uniform elliptical disk (major axis is diameter)
   - 3: optically thin spheroid or tapered disk (major axis is diameter)
   - 4: elliptical ring (major axis is diameter)
   - 5: line (major axis is length)

For Gaussians and delta-functions, the Type can be omitted; for delta-functions, the major-axis, axial-ratio, position-angle and type can be omitted. (Not all the programs understand types 2-5.) The "origin" is an arbitrary phase-reference point. The maximum number of components varies from 600 to 10000, depending on the individual program.

Caveats:
- In CASA, directions in componentlist are currently absolute only. Hence one must specify the reference direction.

WENSS The **Westerbork Northern Sky Survey (WENSS)** is a low-frequency radio survey that covers the whole sky north of delta=30 degree at a wavelength of 92cm to a limiting flux density of approximately 18 mJy (5 sigma). This survey has a resolution of 54" x 54" cosec(delta) and a positional accuracy for strong sources of 1.5". The WSRT Northern Sky Survey catalog is available via a Web interface. Use this interface to search for sources, choose the plain text output, capture the output into a file and then convert.

FIRST **Faint Images of the Radio Sky at Twenty-cm** – is a project designed to produce the radio equivalent of the Palomar Observatory Sky Survey over 10,000 square degrees of the North and South Galactic Caps. Using the NRAO Very Large Array (VLA) and an automated mapping pipeline, they produce images with 1.8" pixels, a typical rms of 0.15 mJy, and a resolution of 5". At the 1 mJy source detection threshold, there are ~90 sources per square degree, ~35% of which have resolved structure on scales from 2-30". Go to the FIRST catalog search page, search for the sources that you want, cut out only the lines that include the sources, put in a file and then convert.

NVSS The **NRAO VLA Sky Survey** The NRAO VLA Sky Survey (NVSS) is a radio continuum survey covering the sky north of -40 deg

Caveats:

- The catalog contains upper limits on some scale sizes. We have chosen to represent these as actual sizes.

### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>filename</strong></td>
<td>Name of output component list table</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><strong>asciifile</strong></td>
<td>Name of input ascii file</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><strong>refer</strong></td>
<td>Input reference frame</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>B1950</td>
</tr>
<tr>
<td>J2000</td>
<td></td>
</tr>
<tr>
<td><strong>format</strong></td>
<td>Name of format (only ST supported)</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>ST</td>
</tr>
<tr>
<td><strong>direction</strong></td>
<td>Direction measure (for relative coordinates)</td>
</tr>
<tr>
<td>allowed:</td>
<td>record</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><strong>spectrum</strong></td>
<td>Default spectrum field, valid spectrum field [type=&quot;Constant&quot;, frequency=[type=&quot;frequency&quot; , refer=&quot;LSR&quot; , m0=[unit=&quot;GHz&quot; , value=1.0]] allowed: record</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><strong>flux</strong></td>
<td>Default flux field, valid flux field [value=[0.0, 0.0, 0.0, 0.0], unit='Jy', polarization=&quot;Stokes&quot;] allowed: record</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><strong>log</strong></td>
<td>Send a message to the logger</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>true</td>
</tr>
</tbody>
</table>
Returns
int

Example

mycl := asciitocomponentlist('sgra-stars.cl', 'sgra-stars.stfile', refer='j2000', format='ST')
**componentlist.concatenate - Function**

1.2.1 Append components from another componentlist.

**Description**

The concatenate function copies the specified component(s), from the specified list, to the end of the current list. The components are specified by numbering them from one to the length of the list. You cannot append components to a list that has been opened read only but the list you are copying from may be readonly.

You use a vector of indices to copy a number of components at once. By default all components are copied.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>list to copy from. Can be a componentlist record or a componentlist file name from disk</td>
<td>any</td>
<td>unset</td>
</tr>
<tr>
<td>which</td>
<td>which components to copy, -1 unset</td>
<td>intArray</td>
<td>-1</td>
</tr>
<tr>
<td>log</td>
<td>Send a message to the logger</td>
<td>bool</td>
<td>true</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```plaintext
cl.addcomponent(flux=1.0, dir='J2000 19h00m00 -40d00m00')
cl.addcomponent(flux=2.0, dir='J2000 19h10m00 -40d00m00')
```
cl.addcomponent(flux=3.0, dir='J2000 19h00m00 -40d00m00')
c12 = cltool();
c12.concatenate(cl.torecord(), [0,2]);
c12.done()
c12.rename('part_list.cl');
c12.done()

We make a 3 component component list and copies the first and third component to another a componentlist that was initially empty. These components are then saved to the table called part_list.cl.

cl.close() ### make sure we start with empty componentlist
c1.concatenate('crux.cl', [0,2]);
c1.rename('crux-copy.cl');
c1.done()

This example reads a componentlist from a casa table and copies the first and third component to another a componentlist that was initially empty. These components are then saved to the table called crux-copy.cl.
**componentlist.fromrecord - Function**

1.2.1 make a componentlist tool from a record

**Description**

This function allows the componentlist records that are returned by other functions (for e.g from imageanalysis tool) be converted to a tool to be manipulated or to be saved on disk.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>record</td>
<td>a component list record</td>
</tr>
<tr>
<td></td>
<td>allowed: record</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
cl2 = cltool()
cl2.fromrecord(ia.findsources())
cl2.rename('sourcesfound.cl')
cl2.done()
```
Componentlist.torecord.html

Componentlist.torecord - Function

1.2.1 convert componentlist to a record

Description

This function allows the componentlist to be converted to a record. Usually useful to pass to other functions in image analysis for e.g

Arguments

Inputs

Returns

record

Example

```
c12=cltool()
c1.open('mycomp.cl')

c12.fromrecord(ia.deconvolvecomponentlist(c1.torecord()))
c12.rename('deconvolved_sources.cl')
c12.done()
```
componentlist.remove - Function

1.2.1 Remove a component from the list.

Description

The remove function removes the specified component(s) from the list. Components are specified by numbering them from one to the length of the list. So removing component one will remove the first component. After using this function all the remaining components will be shuffled down so that component two becomes component one. You cannot remove components from a list that has been opened read only.

You can specify a vector of indices to remove a number of components at once. For example in a five element list removing elements [1,3,5] will result in a two element list, now indexed as elements one and two, containing what was previously the second and fourth components.

Components that have been deleted using this function are not lost. The recover function can be used to get them back unless the purge function has been executed. Then they are completely gone.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which indices</td>
<td>indices of which component(s) to remove a vector containing unique integers</td>
</tr>
<tr>
<td>allowed: intArray</td>
<td>between 0 and one less than the length of the list, -1 for all</td>
</tr>
<tr>
<td>Default:</td>
<td>-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>log Send a message to the logger</th>
<th>allowed: bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default: true</td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool

Example
cl.open('crux.cl')
cl.remove(1)
componentlist.purge.html

**componentlist.purge - Function**

1.2.1 Permanently delete removed components.

**Description**

The remove function deletes components from the list but does not remove them from memory. They remain accessible and can be obtained with the recover function. The purge function frees up the memory occupied by the removed components. You cannot use the recover function to obtain the removed components after the purge function has been called.

**Arguments**

**Returns**

`bool`

**Example**

```python
cl.open('crux.cl')
cl.remove(1)
cl.purge()
```
**Componentlist.recover - Function**

1.2.1 Obtain removed components.

**Description**

The recover function appends components to the end of the list that have been deleted with the remove function. This does not include components that were removed before the purge function was last executed.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log</td>
<td>Send a message to the logger</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: true</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```plaintext
cf.open('crux.cl')
cf.remove(1)
cf.recover()
```
1.2.1 Find the number of components in the list.

Description

The length function returns a non-negative integer that indicates how many components the list currently contains.

Arguments

Returns

int

Example

```python
c1.open('crux.cl')
n = c1.length()
```
componentlist.indices.html

**componentlist.indices - Function**

1.2.1 Return a vector of indices. *(Not implemented yet)*

**Description**

The indices function returns a vector of non-negative integers that can be used to index through the list. This vector always contains the integers starting at one and increasing sequentially to the length of the list. Its main use is in for loops as is illustrated in the example below.

**Arguments**

**Returns**

intArray

**Example**

```plaintext
include 'componentlist.g'
cl := componentlist('crux.cl');
allcomp := cl.indices();
cl.convertfluxunit(allcomp, 'jy');
cl.convertfluxpol(allcomp, 'stokes');
totalflux := [0,0,0,0];
for (i in allcomp) {
    totalflux +:= cl.getfluxvalue(i);
}
```
componentlist.sort.html

componentlist.sort - Function

1.2.1 Sort the components in a list

Description

The sort function can sort all the components in a list using a variety of criteria. Currently the following criteria are available.

**Flux** Sorts the list so that the brightest components, as defined by the \( \text{abs}(I) \), are at the beginning of the list.

**Position** Sorts the list so that components that are closest to a reference position, which is currently fixed at \((\text{ra,dec}) = (0,0)\), are at the beginning of the list.

**Polarization** Sorts the list so that components with the largest fractional polarization, \( \sqrt{Q^2 + U^2 + V^2} \), are at the front of the list. Components where \( I = 0 \) are placed at the end of the list.

The parsing of the string containing the sorting criteria is case insensitive. You cannot sort a list that has been opened read only.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>a string containing the criteria to use to sort the list</th>
</tr>
</thead>
<tbody>
<tr>
<td>criteria</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: Polarization</td>
</tr>
<tr>
<td></td>
<td>Position</td>
</tr>
<tr>
<td></td>
<td>Flux</td>
</tr>
<tr>
<td>log</td>
<td>Send a message to the logger</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: true</td>
</tr>
</tbody>
</table>

Returns

bool

Example
cl.open('crux.cl')
c1.sort('Polarization')
componentlist.isphysical.html

**componentlist.isphysical - Function**

1.2.1 Check if a component is physically plausible

**Description**

The isphysical function is used to check if the specified components meet a number of criteria that must be true if the component could be used to model a physical process. These criteria are:

- \( I \geq \sqrt{Q^2 + U^2 + V^2} \)
- That the flux, when represented using the Stokes representation, has a zero imaginary value.

The “Flux properties” section of the ComponentModels module documentation describes how it is possible to generate a component which has non-zero imaginary value in the Stokes representation.

It is possible to check a number of components at once by specifying the indices of all the components. The returned value will only be True if all the specified components are physical.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
</tr>
<tr>
<td>A vector of indices Indices must be between 0 and one less than the list length, inclusively allowed: intArray Default: -1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
</tr>
</tbody>
</table>

**Example**

```python
c12 = cltool()
```
c12.simulate(2)
c12.setflux(1, value=[10, 1+3j, 1-4j, 0], polarization="linear");
print c12.isphysical([0,1])
**componentlist.sample - Function**

1.2.1 Sample the flux of the list in a specified direction. *(Not implemented yet)*

**Description**

The sample function returns a vector containing the flux in Janskys/pixel of all the components in the list, in the specified direction, at the specified frequency. The returned vector always contains four elements corresponding to the Stokes parameters I,~Q,~U,~V.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>direction</td>
<td>The direction to sample any valid direction measure. A valid Direction measure or vector of string or string, e.g. <code>me.direction('J2000', '19h30m00', '-20d00m00')</code> or <code>['J2000', '19h30m00', '-20d00m00']</code> or 'J2000 19h30m00 -20d00m00'</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>J2000 00h00m00.00 90d00m00.0</td>
</tr>
<tr>
<td>pixellatsize</td>
<td>the x-size of the in pixels to use when sampling any quantity that has angular units.</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>0.0deg</td>
</tr>
<tr>
<td>pixellongsize</td>
<td>the y-size of the in pixels to use when sampling any quantity that has angular units.</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>0.0deg</td>
</tr>
<tr>
<td>frequency</td>
<td>The frequency to sample at Any frequency measure</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>1.4GHz</td>
</tr>
</tbody>
</table>

**Returns**

doubleArray

**Example**

618
include 'componentlist.g'
include 'measures.g'
cl := componentlist('crux.cl', readonly=T);
dir := dm.direction(J2000, '12h26m35.9', '-63d5m56');
pixelsize := dm.quantity('1arcsec');
flux := cl.sample(dir, pixelsize);
componentlist.rename.html

**componentlist.rename - Function**

**1.2.1** Give the list a name so it can save itself. use close to save to disk

**Description**

The rename function is used to specify the name of the table associated with this componentlist. When a componentlist is created using the emptycomponentlist constructor it is not associated with an casa table. So when the componentlist is removed from memory its contents are lost. But if a name is attached to the componentlist, using the rename function, then its contents are saved in a table with the specified name when the componentlist is closed.

**NOTE: that by just using rename the componentlist is not ensured to be on disk; to be sure use close after rename**

If the componentlist is created using the componentlist constructor then this function will rename the table associated with the list to the user specified name. You cannot rename a componentlist that has been opened read only.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>The filename of the table</td>
</tr>
<tr>
<td>log</td>
<td>Send a message to the logger</td>
</tr>
</tbody>
</table>

  | allowed: | string |
  | Default: | |
  | allowed: | bool |
  | Default: | true |

**Returns**

bool

**Example**
cl.simulate(1);
c1.setshape(0, 'gaussian', '35mas', '27mas', '-10d')
c1.setflux(0, [1.0, 0.2, 0.1, 0.01])
c1.rename('smallblob.cl')
c1.close();

cl.open('smallblob.cl')
n=c1.length()

This example starts with an empty componentlist tool and then adds one component to it. The parameters of this component are then modified to change the shape and flux and the list saved in the casa table called 'smallblob.cl' The data is not written to disk until the list is closed, and when it is the componentlist is reset. So you need to reopen it if you want to interact with it.
componentlist.simulate.html

**componentlist.simulate - Function**

1.2.1 Add some simulated components to the list

**Description**

The simulate function adds simulated components to the list. The simulation criterion is very simple, all the components added are identical! They are point sources at the J2000 north pole with a flux in Stokes I of $1 \sim \text{Jy}$, and zero in the other polarizations. The spectrum is constant. The 'set' functions (eg., setflux, setfreq) can be used to change these parameters to desired ones. The howmany argument indicates how many components to append to the list.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>howmany</td>
<td>How many components to simulate, greater than zero</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 1</td>
</tr>
<tr>
<td>log</td>
<td>Send a message to the logger</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: true</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```
cl.simulate(2)
cl.setflux(1, [2.78, 0, 0, 0]);
cl.rename('test.cl');
cl.close();
```
This example creates a component list with two components. The setflux function is used to modify the second component. The list is then saved on disk.

I use short scripts like this a lot during testing.

I expect bad things will happen if you save the list to disk, using the close function, before having shut down the editor gui (using the done button) or if you modify the same component using any of the set functions while it is being modified by the gui.
componentlist.addcomponent.html

**componentlist.addcomponent - Function**

1.2.1 Add a component to the list

**Description**

The addcomponent function is a convenience function that ties together the simulate function, and the various set functions. This function adds a component to the end of the list. For a description of the arguments see the following functions.

- **flux, fluxunit, polarization** See the setflux function.
- **ra, raunit, dec, decunit** See the setrefdir function.
- **dirframe** See the setrefdirframe function.
- **shape, majoraxis, minoraxis, positionangle** See the setshape function.
- **freq** A frequency quantity which is split into a value and units and passed to the setfreq function.
- **freqframe** See the setfreq function.
- **spectrumentype, index** The spectral index $\alpha$ such that flux density $S$ as a function of frequency $\nu$ is given by the formula: $S \propto \nu^\alpha$
  
  See the setspectrum function.
  
  OR
  
  setspectrum

- **label** See the setlabel function.

**Arguments**
Inputs

**flux**
The flux value. A vector with four real or complex numbers
allowed: any
Default:

**fluxunit**
The units of the flux. Any string with the same dimensions as the Jansky
allowed: string
Default: Jy

**polarization**
The polarization of the value field. “Stokes”, “linear” or “circular”
allowed: string
Default: Circular
- Linear
- Stokes

**dir**
The direction measure of the source, it can be any direction measure from the measures tool or a string of the type '[J2000 10h30m00 20d00m00.0]' or a vector of strings of the type ['J2000', '10:30:00.00', '-20.00.00.0']. Basically the string or strings should have the direction frame and quantities for Ra and Dec
allowed: any
Default: J2000 00h00m00.0 90d00m00.0

**shape**
The new shape type. A string that is either 'point', 'Gaussian', 'disk', or 'limbdarkeneddisk'
allowed: string
Default: disk
- limbdarkeneddisk
- Gaussian
- point

**majoraxis**
The width (FWHM in the case of a Gaussian) of the larger axis. A quantity with angular units
allowed: any
Default: 2.0arcmin

**minoraxis**
The width (FWHM in the case of a Gaussian) of the smaller axis. A quantity with angular units
allowed: any
Default: 1.0arcmin

**positionangle**
The rotation of the axes with respect to the reference frame. A quantity with angular units
allowed: any
Default: 0.0deg

**freq**
The reference frequency. A quantity with units equivalent to the 'Hz' and frame or a frequency measure, e.g ['TOPO', '1.6GHz'], or simply default frame (LSRK) '1.6GHz'
allowed: any
Default: 624SRK 1.415GHz

**spectrumtype**
The spectrum type, a string that is either 'constant' or 'spectral index'
allowed: string
Default: spectral index
- constant

**index**
The spectral index
allowed: double
Default: 1.0


**Returns**

bool
componentlist.close.html

**componentlist.close - Function**

1.2.1 Save the componentlist to disk and reset its state.

**Description**

The close function resets the componentlist to its default state. In this state it contains no components and is not associated with any table. This function flushes all the components in memory to disk if the componentlist is associated with a table. The table is then closed, and the contents of the list deleted. If the list is not associated with a table its contents are still deleted and memory used by the list is released.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log</td>
<td>Send a message to the logger</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>true</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

See the example for the rename function.
**componentlist.edit - Function**

1.2.1 Start up the component editor gui (Not implemented yet)

**Description**

The edit function starts up a graphical user interface which allows the user to view and manipulate individual components. The `which` argument specifies the component to edit.

The component being edited is copied into the componenteditor tool. Hence if you add or remove components or change the order of components in the list while the component is in the editor it will be put back in the wrong place! So do not manipulate the list while editing a component. It is also suggested you only edit one component at a time.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>which</code></td>
<td>An index specifying which component. An integer between 0 and one less than the list length</td>
</tr>
<tr>
<td><code>allowed:</code></td>
<td><code>int</code></td>
</tr>
<tr>
<td><code>Default:</code></td>
<td><code>no default</code></td>
</tr>
<tr>
<td><code>log</code></td>
<td>Send a message to the logger</td>
</tr>
<tr>
<td><code>allowed:</code></td>
<td><code>bool</code></td>
</tr>
<tr>
<td><code>Default:</code></td>
<td><code>true</code></td>
</tr>
</tbody>
</table>

**Returns**

`bool`

**Example**

See the example for the `simulate` function.
componentlist.done.html

**componentlist.done - Function**

1.2.1 Delete the componentlist tool

**Description**

The done function frees up all the memory associated with a componentlist tool. After calling this function the componentlist tool cannot be used, either to manipulate the current list, or to open a new one. This function does not delete the disk file associated with a componentlist, but it will shut down the server process if there are no other componentlist tools being used.

**Arguments**

**Returns**

bool

**Example**

See the example for the rename function.
componentlist.select.html

**componentlist.select - Function**

1.2.1 Mark components in the list

**Description**

The select function is used to mark the specified components as “selected”. This function will be used in conjunction with the planned graphical user interface. Other functions in the componentlist tool will behave no differently if a component is marked as “selected”. Components are not selected when the list is initially read from disk or when a new component is added to the list using the simulate function.

**Arguments**

| Inputs | | | | |
| --- | --- | --- | --- |
| which | A vector of indices. Indices must be between 0 and one less than the list length, inclusively | | |
| allowed: | intArray | | |
| Default: | | | |

**Returns**

bool

**Example**

```plaintext
cl.open('crux.cl')
cl.select([1,3])
```

630
componentlist.deselect - Function

Unmark components in the list

Description

The deselect function is used to remove the “selected” mark from specified components in the list. This function will be used in conjunction with the planned graphical user interface and no other functions in the componentlist will behave differently if a component is marked as “selected” or not. Components are not selected when the list is initially read from disk or when a new component is added to the list using the simulate function. Deselecting a component that is already deselected is perfectly valid and results in no change.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
</tr>
</tbody>
</table>

Returns

bool

Example

```python
cl.open('crux.cl')
cl.select([1,3])
cl.deselect([2,3])
```
componentlist.selected.html

**componentlist.selected - Function**

1.2.1 Determine which components are selected

**Description**

The selected function is used to determine which components in a list are selected. It returns a vector with indices that indicate which components are selected. A zero length vector is returned if no components are selected. Components are marked as selected using the select function. This function will be used in conjunction with the graphical user interface and other functions in the componentlist tool will behave no differently if a component is marked as “selected” or not.

**Arguments**

**Returns**

intArray

**Example**

```
c1.open('crux.cl')
c1.select([1,3])
c1.deselect([2,3])
c1.selected()
```
componentlist.getlabel.html

**componentlist.getlabel - Function**

1.2.1 Get the label of the specified component

**Description**

The getlabel function returns the label associated with the specified component. The label is an arbitrary text string that can be used to tag a component.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>which</strong></td>
<td>An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

string

**Example**

```plaintext
cl.open('crux.cl')
cl.getlabel(1)
```
**componentlist.setlabel - Function**

**1.2.1** Set the label of the specified components

**Description**

The setlabel function is used to reassign the label (an arbitrary text string) of the specified components to a new value.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>An index specifying the component to modify. An integer between 0 and one less than the list length, inclusively</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: no default</td>
</tr>
<tr>
<td>value</td>
<td>The label for the specified components</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>log</td>
<td>Send a message to the logger</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: true</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
cl.open('centarusA.cl')
cl.setlabel(1, 'Core')
```
componentlist.getfluxvalue - Function

1.2.1 Get the flux value of the specified component

Description

The `getfluxvalue` function returns the value of the flux of the specified component using the current units and the current polarization representation. The functions `getfluxunit` & `getfluxpol` can be used to get the units and polarization representation that corresponds to the supplied value.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>An index specifying which component. An integer between 0 and one less than</td>
</tr>
<tr>
<td></td>
<td>the list length, inclusively</td>
</tr>
<tr>
<td>allowed: int</td>
<td>Default: no default</td>
</tr>
</tbody>
</table>

Returns
doubleArray

Example

```plaintext
cl.open('crux.cl');
flux = cl.getfluxvalue(1);
unit = cl.getfluxunit(1);

This example returns the values, units, polarization and error of the first component in the list.
```
componentlist.getfluxunit.html

**componentlist.getfluxunit - Function**

1.2.1 Get the flux unit of the specified component

**Description**

The getfluxunit function returns the units of the flux of the specified component. The actual values are obtained using the getfluxvalue function.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

string

**Example**

See the example for the getfluxvalue function.
componentlist.getfluxpol.html

**componentlist.getfluxpol - Function**

1.2.1 Get the polarization representation for the flux of the specified component (**Not implemented yet**)

**Description**

The getfluxunit function returns the polarization representation of the flux of the specified component. The actual values are obtained using the getfluxvalue function.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

string
componentlist.getfluxerror.html

**componentlist.getfluxerror - Function**

1.2.1 Get the error in the flux of the specified component

**Description**

The getfluxerror function returns the error in the flux of the specified component using the current units and the current polarization representation. The functions getfluxvalue & getfluxunit & getfluxpol & can be used to get the value, units and polarization representation that corresponds to the specified error. No error calculations are done by this tool. The error can be stored and retrieved and if any of the parameters of the flux change the user is responsible for updating the errors.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>Index specifying which component. An integer between 0 and one less than the list length, inclusively. Allowed: int</td>
</tr>
<tr>
<td>Default</td>
<td>int</td>
</tr>
</tbody>
</table>

**Returns**

doubleArray
componentlist.setflux - Function

1.2.1 Set the flux of the specified components

Description

The setflux function is used to reassign the flux of the specified components to a new value. The flux value, unit and polarization can be specified and any number of components can be set to the new value. (Currently, the parameter, error is ignored.)

Arguments

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>A vector of indices specifying the components to modify. A vector with indices between 0 and one less than the list length, inclusively</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>The flux values for the specified components A vector with four real or complex numbers</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td>unit</td>
<td>The units of the flux. Any string with the same dimensions as the Jansky</td>
<td>string</td>
<td>Jy</td>
</tr>
<tr>
<td>polarization</td>
<td>The polarization of the value field</td>
<td>string</td>
<td>circular</td>
</tr>
<tr>
<td>error</td>
<td>The error in the value field. A complex vector of length four.</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td>log</td>
<td>Send a message to the logger</td>
<td>bool</td>
<td>true</td>
</tr>
</tbody>
</table>
Returns
bool

Example

c1.open('crux.cl');
c1.setflux(0, [1,0,0,0], unit='jy',
    polarization='Stokes', error=[.3, 0, 0, 0])
componentlist.convertfluxunit - Function

**1.2.1** Change (convert) the flux units of the specified components

**Description**

The convertfluxunit function is used to convert the flux to another unit. The units must have the same dimensions as the Jansky.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
</tr>
<tr>
<td>A vector of indices specifying the components to modify. A vector with indices between 0 and one less than the list length, inclusively</td>
</tr>
</tbody>
</table>

| unit |
| The units of the flux. Any string with the same dimensions as the Jansky | allowed: string | Default: Jy |

**Returns**

bool

**Example**

```python
c1.open('crux.cl')
print c1.getfluxvalue(1)
c1.convertflux(1, 'WU')
print c1.getfluxvalue(1)
```
componentlist.convertfluxpol.html

**componentlist.convertfluxpol - Function**

1. Change (convert) the polarization representation of the specified components

**Description**

The convertfluxpol function is used to convert the flux to another polarization representation. There are three representations used, namely, 'Stokes', 'linear' & 'circular'.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>A vector of indices specifying the components to modify. A vector with indices between 0 and one less than the list length, inclusively.</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>polarization</td>
<td>The new polarization representation</td>
<td>string</td>
<td>circular</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
cl.open('centaurusA.cl')
print cl.getfluxvalue(1)
cl.convertfluxpol(1, 'linear')
print cl.getfluxvalue(1)
```
componentlist.getrefdir.html

componentlist.getrefdir - Function

1.2.1 Return the reference direction

Description

The getrefdir function returns, as a direction measure, the reference direction for the specified component. The reference direction is for all the currently supported component shapes the direction of the centre of the component.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int</td>
</tr>
<tr>
<td>Default:</td>
<td>record</td>
</tr>
</tbody>
</table>

Example

```python
cl.open('crux.cl')
dir = cl.getrefdir(1)
```
componentlist.getrefdirra.html

componentlist.getrefdirra - Function

Get the RA of the reference direction. (Not implemented not)

Description

The getrefdirra function returns the right ascension of the reference direction of the component as a formatted string. If the reference frame is something other than J2000 or B1950 the value returned is the latitude or the azimuthal angle. The unit argument specifies the units for the returned value. It can be any angular unit (eg. 'deg', 'rad', 'arcsec', 'mas') or it can be 'angle' or 'time'. If it is 'angle' then the returned string is formatted in degrees, minutes, seconds ie., '+DDD.MM.SS.sss'. If it is 'angle' then the returned string is formatted in hours, minutes, seconds ie., 'HH:MM:SS.sss'.

The precision argument controls the numerical precision of the returned value. For the angular units it controls how many digits are in the returned string. For the 'angle' unit, precisions of two, four & six control whether the degrees, minutes or seconds are returned. Higher precisions increase the precision of the seconds field. Similarly, for the 'time' unit precisions of two, four & six control whether the hours, minutes or seconds are returned.

All directions are stored internally in double precision.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>An index specifying which component. An integer between 0 and one less than the list length, inclusively</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>unit</td>
<td>The angular unit of the returned value. Any string containing an angular unit or 'angle' or 'time'</td>
<td>string</td>
<td>deg</td>
</tr>
<tr>
<td>precision</td>
<td>The number of digits in the returned string. Numbers between 1 and 16 make the most sense</td>
<td>int</td>
<td>6</td>
</tr>
</tbody>
</table>

Returns

645
Example

#include 'componentlist.g'
c1 := componentlist('crux.cl');
print 'The first component is at RA: ', c1.getrefdirra(1, 'time'),
      ' Dec: ', c1.getrefdirdec(1, 'angle'),
      '(' c1.getrefdirframe(1), ')'
componentlist.getrefdirdec - Function

1.2.1 Get the declination of the reference direction. (Not implemented yet)

Description

The getrefdirdec function returns the declination of the reference direction of the component as a formatted string. If the reference frame is something other than J2000 or B1950 the value returned is the longitude or the altitude. See the getrefdirra function for a description of the unit and precision arguments.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>An index specifying which component. An integer between 0 and one less than the list length, inclusively</td>
</tr>
<tr>
<td>unit</td>
<td>The angular unit of the returned value. Any string containing an angular unit or 'angle' or 'time'</td>
</tr>
<tr>
<td>precision</td>
<td>The number of digits in the returned string. Numbers between 1 and 16 make the most sense</td>
</tr>
</tbody>
</table>

| allowed        | int                                                                        |
| precision      | string                                                                     |
| allowed        | string                                                                     |
| precision      | int                                                                        |

Returns

string

Example

See the example for the getrefdirra function.
componentlist.getrefdirframe.html

componentlist.getrefdirframe - Function
1.2.1 Get the reference frame of the reference direction.

Description

The getrefdirframe function returns the reference frame of the reference direction of the component as a string. The returned string is always in upper case. Common frames are, 'J2000', 'B1950' and 'GALACTIC'.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

string

Example

See the example for the getrefdirra function.
**componentlist.setrefdir** - Function

1.2.1 Set the reference direction

**Description**

The `setrefdir` function sets the reference direction of the specified components to a new value. The direction is defined by separately specifying the right ascension and the declination.

The right ascension is specified as a string or a real number.

Ra can be in standard angle units 'deg', 'rad', or time formatted as such 'HH:MM:SS.sss' eg., '19:34:63.8' or angle formatted as such '+DDD.MM.SS.sss' eg., '127.23.12.37'.

Similarly the declination is specified as a string or a real number and the decunit can be any angular unit or 'angle' or 'time'.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>A vector of indices specifying the components to modify. A vector with indices between 0 and one less than the list length, inclusively.</td>
</tr>
<tr>
<td>ra</td>
<td>The RA of the new direction, A formatted string or a number</td>
</tr>
<tr>
<td>dec</td>
<td>The declination of the new direction. A formatted string or a number</td>
</tr>
<tr>
<td>log</td>
<td>Send a message to the logger</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>any</td>
<td></td>
</tr>
<tr>
<td>any</td>
<td></td>
</tr>
<tr>
<td>bool</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool
Example

cl.simulate(3)
cl.setrefdir(0, '12:26:35.9', '-63.5.56')
cl.setrefdir(1, '12h26m35.9', '-63d5m56')
cl.setrefdir(2, '-173.35deg', '-1.10128rad')
cl.rename('testcls.cl')  # write to disk
componentlist.setrefdirframe - Function

1.2.1 Set the reference frame for the direction

Description

The setrefdirframe function sets the reference frame for the reference direction of the specified components (what a mouthful)!
Currently the reference frame does not include additional information like when and where the observation took place. Hence only reference frames that are independent of this information can be used. This includes the common ones of 'J2000', 'B1950', and 'Galactic'. The measures module contains a complete listing of all possible reference frames. The parsing of the reference frame string is case-insensitive.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>A vector of indices specifying the components to modify. A vector with indices between 0 and one less than the list length, inclusively. allowed: int Default:</td>
</tr>
<tr>
<td>frame</td>
<td>The new reference frame, A string like 'B1950', 'J2000' or 'galactic'. allowed: string Default:</td>
</tr>
<tr>
<td>log</td>
<td>Send a message to the logger allowed: bool Default: true</td>
</tr>
</tbody>
</table>

Returns

bool

Example
cl.open('crux.cl');
cl.setrefdirframe(0, 'B1950');
**componentlist.convertrefdir - Function**

1.2.1 Convert the reference direction to a new frame

**Description**

The convertrefdir function changes the specified components to use a new direction reference frame. Using this function will change the right-ascension and declination of the component(s), unlike the setrefdirframe which does not. Please see the setrefdirframe function for a description of what frames are allowed.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>A vector of indices specifying the components to modify.</td>
</tr>
<tr>
<td></td>
<td>A vector with indices between 0 and one less than the list length, inclusively</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>frame</td>
<td>The new reference frame A string like 'B1950', 'J2000' or 'galactic'</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```plaintext
cl.open('crux.cl');
cl.convertrefdirframe(0, 'J2000');
```
componentlist.shapetype.html

**componentlist.shapetype - Function**

1.2.1 Returns the shape type of the component

**Description**

The shapetype function returns the current shape of the specified component. The shape defines how the flux of the component varies with direction on the sky. Currently there are three shapes available. These are 'Point', 'Gaussian', 'Disk', and 'Limbdarkeneddisk' (experimental). This function returns one of these four strings.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
</tr>
</tbody>
</table>

An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int

Default:

**Returns**

string

**Example**

```python
cl.open('crux.cl')
print 'The first component has a', cl.shapetype(0), ' shape'
```
componentlist.getshape.html

**componentlist.getshape - Function**

1.2.1 Return the shape parameters the component

**Description**

The getshape function returns the shape parameters of a component in a record. As different shapes can have a differing number and type of parameters the shape parameters are returned in a record with fields that correspond to parameters relevant to the current shape.

For a point shape there are only two fields; type and direction. These are the shape type, and the reference direction. These values are also returned by the shapetype and getrefdir functions.

For both the Gaussian and disk shapes there are three additional fields; majoraxis, minoraxis & positionangle. These are angular quantities, and hence are records with a value and a unit.

**Arguments**

| Inputs | An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int |
| which | Default: |

**Returns**

record

**Example**

See the examples for the setshape & convertshape functions.
componentlist.setshape.html

**componentlist.setshape - Function**

1.2.1 Change the shape of the component

**Description**

The setshape function changes the shape of the specified components to the user specified shape.

The type argument defines what the sort of new shape to use. This can be either 'point', 'Gaussian', 'disk', or 'limbdarkeneddisk'. The parsing of this string is case insensitive. The 'limbdarkeneddisk' is an experimental disk model with the limb-darkening effect, where the sky brightness is described as

\[ I = I_o (1 - (r/R)^2)^{n/2} \]

with R being apparent body radius. The n can be set in optionalparms (if it is not set, the default value, 0.0 will be used).

If the shape type is 'point' then the remaining arguments in this function are ignored. There are no other parameters needed to specify a point shape.

But if the shape is 'Gaussian', 'disk', or 'limbdarkeneddisk', the remaining arguments are needed to fully specify the shape. The majoraxis, minoraxis and positionangle arguments are quantities (see the quanta module for a definition of a quantity). Hence they can be specified either as with string eg., '1arcsec' or with a record eg., [value=1, unit='deg'].

The major axis is the width of the larger axis. For the Gaussian shape this is the full width at half maximum. And the minor axis is the width of the orthogonal axis. The positionangle is the specifies the rotation of these two axes with respect to a line connecting the poles of the current direction reference frame. If the angle is positive the the north point of the component moves in the eastern direction.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>A vector of indices specifying the components to modify.</td>
</tr>
<tr>
<td>type</td>
<td>The new shape type. A string that is either 'point', 'Gaussian', 'disk', or 'limbdarkeneddisk'.</td>
</tr>
<tr>
<td>majoraxis</td>
<td>The width of the larger axis. A quantity with angular units</td>
</tr>
<tr>
<td>minoraxis</td>
<td>The width of the smaller axis. A quantity with angular units</td>
</tr>
<tr>
<td>positionangle</td>
<td>The rotation of the axes with respect to the reference frame. A quantity with angular units</td>
</tr>
<tr>
<td>majoraxiserror</td>
<td>Error of the width of the larger axis. A quantity with angular units</td>
</tr>
<tr>
<td>minoraxiserror</td>
<td>Error of the width of the smaller axis. A quantity with angular units</td>
</tr>
<tr>
<td>positionangleerror</td>
<td>Error of the rotation of the axes with respect to the reference frame. A quantity with angular units</td>
</tr>
<tr>
<td>optionalparms</td>
<td>optional parameters in a vector (for limbdarkeneddisk)</td>
</tr>
<tr>
<td>log</td>
<td>Send a message to the logger</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

657
bool

Example

cl.open('crux.cl', nomodify=False)
c1.setshape(3, 'disk', '45mas', '45mas')
print c1.getshape(3)['majoraxis']
componentlist.convertshape.html

**componentlist.convertshape - Function**

1.2.1 Change the units of the shape parameters *(Not implemented yet)*

**Description**

The convertshape function changes the units of the specified shape parameters on the specified components. When changing the units it also converts the values so that overall the angle has not changed. Depending on the component shape some arguments of this function are ignored. If the shape type is 'point’, then all but the which argument are ignored. This function is useless for points. If the shape is a 'gaussian’ or 'disk’ then this will modify the units of the major and minor axes and the positionangle. Use the getshape function to see these parameters using the new units.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>A vector of indices specifying the components to modify. A vector with indices between 0 and one less than the list length, inclusively. allowed: int Default:</td>
</tr>
<tr>
<td>majoraxis</td>
<td>The units to use on the larger axis. A string with angular units allowed: string Default: rad, deg, mas, arcsec, arcmin</td>
</tr>
<tr>
<td>minoraxis</td>
<td>The units to use on the smaller axis. A string with angular units allowed: string Default: rad, deg, mas, arcsec, arcmin</td>
</tr>
<tr>
<td>positionangle</td>
<td>The units to use for the rotation of these axes. A string with angular units allowed: string Default: rad, deg</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```plaintext
include 'componentlist.g'
c1 := componentlist('crux.cl');
c1.convertshape(3, 'arcsec', 'arcsec');
print c1.getshape(3).minoraxis;
```
componentlist.spectrumtype.html

**componentlist.spectrumtype - Function**

1.2.1 Returns the spectral shape of the component

**Description**

The spectrumtype function returns the current spectral shape of the specified component. The spectral shape defines how the flux of the component varies with frequency. Currently there are two spectral shapes available. These are 'Constant' and 'Spectral Index'. This function returns one of these two strings.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

string

**Example**

```python
cl.open('crux.cl')
print 'The first component has a', cl.spectrumtype(1), ' spectrum'
```

---

661
componentlist.getspectrum.html

**componentlist.getspectrum** - Function

1.2.1 Return the spectral parameters the component

**Description**

The getspectrum function returns the spectral parameters of a component in a record. As different spectral shapes can have a differing number and type of parameters the spectral parameters are returned in a record with fields that correspond to parameters relevant to the current spectral shape.

For a constant spectrum there are only two fields; type and frequency. These are the spectral type, and the reference frequency. These values are also returned by the spectrumtype and getfreq functions.

For the spectral index spectral shape there is also an index field. This contains a vector with four numbers, these are the spectral indices for the $I,\sim Q,\sim U,\&\sim V$ components of the flux.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

See the examples for the setspectrum \\& getspectrum functions.
componentlist.setstokesspectrum - Function

1.2.1 Change the spectrum of the component

Description

The setstokesspectrum function changes the spectrum of the specified components to the user specified spectrum. This is different from setspectrum as it provides ways to control variation of all 4 Stokes parameters with frequency. If only I variation is needed please use setspectrum.

The type argument defines what the sort of new spectrum to use. This can be either 'constant' or 'spectral index' or 'tabular'. The parsing of this string is case insensitive.

If the spectrum type is 'constant' then the remaining arguments in this function are ignored. There are no other parameters needed to specify a constant spectrum.

But if the spectrum is 'spectral index', the index argument is needed. It is a 4 element vector.

The first element \( (\alpha_0) \) is the spectral index of stokes I \( I(\nu) = I(\nu_0) \left( \frac{\nu}{\nu_0} \right)^{\alpha_0} \)

The second element \( (\alpha_1) \) is a spectral index for the fractional linear polarization \( \sqrt{ \frac{(Q(\nu))^2+U(\nu)^2}{I(\nu)^2} } = \sqrt{ \frac{(Q(\nu_0))^2+U(\nu_0)^2}{I(\nu_0)^2} } \left( \frac{\nu}{\nu_0} \right)^{\alpha_1} \). \( \alpha_1 = 0 \) implies constant fractional linear polarization w.r.t frequency.

The third element is a "Rotation Measure" factor, i.e angle of rotation \( \theta = \alpha_2 (\lambda^2 - \lambda_0^2) \) of the linear polarization at frequency \( \nu \) w.r.t frequency \( \nu_0 \).

The fourth element is a spectral index for the fractional spectral polarization \( \frac{V(\nu)}{I(\nu)} = \frac{V(\nu_0)}{I(\nu_0)} \left( \frac{\nu}{\nu_0} \right)^{\alpha_3} \)

If the spectrum is 'tabular', then index is ignored but the six parameters tabularfREQs, tabulari, tabularq, tabularu, tabularv and tabularframe are considered. tabularfREQs and tabulari, tabularq, tabularu, tabularv have to be list of same lengths and larger than 2. You need at least 2 samples to interpolate the spectral value in between. The Stokes parameters of the source is interpolated from these values. Extrapolation outside the range given in tabularfREQs is not done.

tabularfREQs should be float values in 'Hz' tabulari, tabularq, tabularu, tabularv should be float values in 'Jy'

You should ensure that the reference frequency is set to the value you desire, using the setfreq function if you change to the spectral index shape.

Arguments
**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Allowed Types</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>The index specifying the component to modify. A value between 0 and one less than the list length, inclusively</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>The new spectrum type. A string that is either 'constant' or 'spectral index' or 'tabular'</td>
<td>string</td>
<td>spectral index</td>
</tr>
<tr>
<td>index</td>
<td>The spectral index.</td>
<td>doubleArray</td>
<td>0.0</td>
</tr>
<tr>
<td>tabularfreqs</td>
<td>The frequencies of for the tabular values, in Hz</td>
<td>doubleArray</td>
<td>1.0e11</td>
</tr>
<tr>
<td>tabulari</td>
<td>tabular Stokes I values, in Jy (same length as tabularfreqs)</td>
<td>doubleArray</td>
<td>1.0</td>
</tr>
<tr>
<td>tabularq</td>
<td>tabular Stokes Q values, in Jy (same length as tabularfreqs)</td>
<td>doubleArray</td>
<td>0.0</td>
</tr>
<tr>
<td>tabularu</td>
<td>tabular Stokes U values, in Jy (same length as tabularfreqs)</td>
<td>doubleArray</td>
<td>0.0</td>
</tr>
<tr>
<td>tabularv</td>
<td>tabular Stokes V values, in Jy (same length as tabularfreqs)</td>
<td>doubleArray</td>
<td>0.0</td>
</tr>
<tr>
<td>reffreq</td>
<td>The reference frequency for spectral index</td>
<td>any</td>
<td>1.4GHz</td>
</tr>
<tr>
<td>frame</td>
<td>The frame for which the frequencies given are in</td>
<td>string</td>
<td>LSRK</td>
</tr>
</tbody>
</table>

**Returns**

664
bool

Example

This example adds a point source model and revises the model point source spectral variation. I is assigned a spectral index of 1. Fractional linear polarisation is assigned a spectral index of 0.4 and similarly for fractional circular polarisation. The linear polarisation angle is kept fixed with frequency.

```python
cl.addcomponent(shape='point', flux=[10.0, 0.4, -0.2, 0.1], dir='J2000 19h00m00 -20d00m00')
cl.setstokesspectrum(which=0, type='spectral index', index=[1.0, 0.4, 0, 0.4], reffreq='2.0GHz')
cl.rename('my19hcomp.cl')
cl.done()
```

In this example a component list is created from scratch and two sources are added. One whose spectral variation is defined by a spectral index and the other one as tabular values. Both components have full Stokes parameters spectral variation defined.

```python
cl.done() ### effectively resets state of cl tool
### add first component
cl.addcomponent(flux=[10, 0.5, -0.3, 0.2], dir='J2000 15h22m00 5d04m00')
cl.setstokesspectrum(which=0, type='spectral index', index=[1.0, 0.4, 0, 0.4], reffreq='2.0GHz')
### adding second component; flux value is unimportant as the tabular values will
### will set it
cl.addcomponent(flux=[1.0, 0, 0, 0], dir='J2000 15h22m30 5d05m00')
## define the IQUV flux variation as tabular values at different frequencies.
cl.setstokesspectrum(which=1, type='tabular', tabularfreqs=[1.0e9, 1.1e9, 1.2e9, 1.3e9, 1.4e9, 1.5e9, 1.6e9],
### saving the componentlist to disk
cl.rename('two_comp.cl')
cl.done() ### done is needed to sync to disk
```
componentlist.setspectrum - Function

1.2.1 Change the spectrum of the component

Description

The setspectrum function changes the spectrum of the specified components to the user specified spectrum.

The type argument defines what the sort of new spectrum to use. This can be either 'constant' or 'spectral index'. The parsing of this string is case insensitive.

If the spectrum type is 'constant' then the remaining arguments in this function are ignored. There are no other parameters needed to specify a constant spectrum.

But if the spectrum is 'spectral index', the index argument is needed to fully specify the spectrum by using the index argument.

If the spectrum is 'tabular', then index is ignored but the three parameters tabularfreqs, tabularflux and tabularframe are considered.

tabularfreqs and tabularflux have to be list of same lengths and larger than 2. You need at least 2 samples to interpolate the spectral value in between. The flux of the source is interpolated from these values.

Extrapolation outside the range given in tabularfreqs is not done.

You should ensure that the reference frequency is set to the value you desire, using the setfreq function if you change to the spectral index shape.

Arguments
Inputs

which
The index specifying the component to modify. A value between 0 and one less than the list length, inclusively
allowed: int
Default:

type
The new spectrum type. A string that is either 'constant' or 'spectral index' or 'tabular'
allowed: string
Default: spectral index

index
The spectral index.
allowed: double
Default: 0.0

tabularfreqs
The frequencies of for the tabular values, in Hz
allowed: doubleArray
Default: 1.0e11

tabularflux
tabular flux density values, in Jy (same length as tabularfreqs)
allowed: doubleArray
Default: 1.0

Returns

bool

Example

cl.open('centarusA.cl')
cl.setspectrum(2, 'spectral index', -0.5)
print cl.getcomponent(2)['spectrum']['index']
cl.done()

This example revises the model for Centaurus-A changing the spectral index of all the components in the left lobe. The output from the print statement is `[-0.5 0 0 0]`
In this example a component is created from scratch as a point source.
The spectrum is set to, say, measured values at 3 frequencies (1GHz, 1.1GHz and 1.4GHz) to 1.0Jy, 0.9Jy, 0.75Jy respectively.
Any frequency in between the range 1 to 1.4 GHz the flux will be estimated by interpolation.
**componentlist.getfreq - Function**

1.2.1 Get the reference frequency *(Not implemented yet)*

**Description**

The getfreq function returns, as a frequency measure, the reference frequency for the specified component. At the reference frequency the flux of the component is the value obtained with the getfluxvalue function. The flux may be different at other frequencies, depending on the components spectral shape. If the spectral shape is constant then changing the reference frequency will not affect the spectrum of the component.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>which An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int</td>
</tr>
<tr>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```plaintext
include 'componentlist.g'
c1 := componentlist('centarusA.cl');
f := c1.getfreq(2);
```

669
**componentlist.getfreqvalue - Function**

1.2.1 Get the reference frequency value *(Not implemented yet)*

**Description**

The getfreqvalue function returns as a floating point number the value of the reference frequency. To fully interpret this value you should also use the frequency unit, obtained using the getfrequnit function and the frequency reference frame, obtained using the getfreqframe function.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>which</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int</td>
</tr>
<tr>
<td>Default</td>
<td>double</td>
</tr>
</tbody>
</table>

**Returns**

double

**Example**

```plaintext
include 'componentlist.g'
cl := componentlist('centarusA.cl');
print 'The reference frequency is ', cl.getfreqvalue(1), ', ', cl.getfrequnit(1), ', (' ', cl.getfreqframe(1), ')'
```

670
The getfrequnit function returns as a string that defines the units of the reference frequency. This unit should be used in conjunction with the getfreqvalue function.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>which</th>
<th>An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Default: int</td>
</tr>
</tbody>
</table>

**Returns**

string

**Example**

See the example for the getfreqvalue function.
**Function**

**componentlist.getfreqframe**

1.2.1 Get the reference frequency frame *(Not implemented yet)*

**Description**

The getfreqframe function returns as a string the reference frame of the reference frequency of the specified component. See the measures module for a description of the available frequency reference frames. Common frames are 'LSR', 'TOPO' and 'GEO'. The frame string is always returned in upper case.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>An index specifying which component. An integer between 0 and one less than the list length, inclusively allowed: int</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

string

**Example**

See the example for the getfreqvalue function.
The setfreq function sets the reference frequency of the specified components to a new value. The frequency is defined by separately specifying the value and its units. Use the setfreqframe function to set the frequency reference frame.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>An index specifying the component to modify. An integer between 0 and one less than the list length, inclusively.</td>
</tr>
<tr>
<td>value</td>
<td>The new frequency value. A number.</td>
</tr>
<tr>
<td>unit</td>
<td>The units of the frequency. Any string with the same dimensions as the 'Hz'.</td>
</tr>
<tr>
<td>log</td>
<td>Send a message to the logger.</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
c1.open('centarusA.cl')
c1.setfreq(1, 1.415, 'GHz')
```
componentlist.setfreqframe - Function

1.2.1 Set the reference frame for the frequency

Description

The setfreqframe function sets the reference frame for the reference frequency of the specified components. Currently the reference frame does not include additional information like when are where the observation took place. Hence no conversion between reference frames is available. In the interim I recommend you always use the same frame.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>An index specifying the component to modify. An integer between 0 and one</td>
</tr>
<tr>
<td></td>
<td>less than the list length, inclusively</td>
</tr>
<tr>
<td>frame</td>
<td>The new reference frame, A string like 'LSRK', 'LSRD', 'GEO' or 'TOPO'</td>
</tr>
<tr>
<td>log</td>
<td>Send a message to the logger</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>allowed</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>TOPO</td>
</tr>
<tr>
<td>GEO</td>
<td></td>
</tr>
<tr>
<td>LSRD</td>
<td></td>
</tr>
<tr>
<td>LSRK</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>allowed</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>true</td>
</tr>
</tbody>
</table>

Returns

bool

Example

675
cl.open('centaurusA.cl')
cl.setfreqframe(0, 'LSRK')
componentlist.convertfrequnit - Function

1.2.1 Convert the reference frequency to a new unit

Description

The convertfrequnit function changes the specified components to use a new unit for the reference frequency. Using this function will change the frequency value also so that the overall reference frequency is not changed. It will affect the values and units obtained with setfreqvalue function. Any unit can be used that has the same dimensions as the 'Hz'.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>An index specifying the component to modify. An integer between 0 and one less than the list length, inclusively</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>unit</td>
<td>The new frequency unit. Any string with the same dimensions as the 'Hz'</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>'GHz'</td>
</tr>
</tbody>
</table>

Returns

bool

Example

```plaintext
c1.open('centaurusA.cl');
c1.convertfrequnit(1, 'kHz');
```

677
1.2.1 Extract a component from the list.

**Description**

The component function returns a record, showing the current state of the specified component in the list. Modifying the record that is returned by this function does not modify the component in the list. To do this you must remove the component from the list, using the remove function, and add the modified component using the add function, or use the replace object function. This function will be removed in a future release of aips++ and you are urged to use the get functions to extract information about a component.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>index of which component to extract. integers between 0 and one less than the length of the list, inclusively</td>
</tr>
<tr>
<td>iknow</td>
<td>Suppress the warning message</td>
</tr>
</tbody>
</table>

**Returns**

record
**componentlist.add - Function**

*1.2.1* Add a component to the list.

**Description**

The add function adds a component to the component list. You cannot add components to a list that has been opened read only. To use this function you need to know the details of the record format. however this has been deprecated and you are urged to use the set functions, in conjunction with the simulate function to add a component to the list.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>thecomponent</td>
<td>A record that represents a component. any record that contains the required fields allowed: record Default:</td>
</tr>
<tr>
<td>iknow</td>
<td>Suppress the warning message allowed: bool Default: true</td>
</tr>
</tbody>
</table>

**Returns**

bool
componentlist.replace.html

**componentlist.replace - Function**

1.2.1 Replace components in the list. *(Not implemented yet)*

**Description**

The replace function replaces the components from the list with the specified components from another list. The source list can be opened readonly and the length of the vectors in the first and third arguments must the the name. You cannot replace components in a list that has been opened read only.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>A vector of indices specifying the components to replace. A vector with indices between 0 and one less than the list length, inclusively</td>
</tr>
<tr>
<td>list</td>
<td>The list containing the components to copy. A componentlist tool</td>
</tr>
<tr>
<td>whichones</td>
<td>A vector of indices specifying the components to copy A vector with indices between 1 and the length of the list in the second argument</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td></td>
</tr>
</tbody>
</table>
componentlist.summarize.html

**componentlist.summarize - Function**

1.2.1 Summarize the specified component to the logger

**Description**

The summarize function summarizes the contents of the specified components to the logger.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>An index specifying which component. Unset or an integer between 0 and one less than the list length, inclusive.</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

bool
Is the argument a componentlist tool? (Not implemented yet)

Description

This global function can be used to determine if the supplied argument is a componentlist tool. If so it returns True, otherwise it returns False.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>The variable that you wish to test</th>
</tr>
</thead>
<tbody>
<tr>
<td>tool</td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

bool

Example

```plaintext
include 'componentlist.g'
if (iscomponentlist(cl)) {
    cl.simulate(2);
} else {
    fail 'Not a componentlist';
}
```
1.3  ms - Module

Module for measurement set operations

Description  A CASA measurement set is a CASA table which obeys specific conventions. These conventions are defined in note 229. Like all CASA tables the measurement set will always appear as a directory which contains a number of files and directories.

Measurement set tables come in two slightly different versions, single dish and interferometric. Single dish measurement sets store the observed data as real numbers in the FLOAT_DATA column of the measurement set, whereas interferometric ones use complex numbers in the DATA column.

A measurement set table can contain data from a variety of different observations with different spectral or polarimetric configurations, different pointings and different instruments. To do this it needs to handle data with differing shapes. The data shape referred to here is two-dimensional with the length of the axes being the number of correlations and the number of channels in the data. A typical shape might be [4, 1], which could correspond to a continuum observation where the [RR, LL, RL, LR] polarizations where correlated. In the same measurement set there may be data with a shape of [1, 32], which corresponds to a spectral line observation, with 32 channels, where only the [XX] polarizations are correlated.
1.3.1 ms - Tool

Operations on measurement sets
Requires: table

Synopsis

Description

The ms tool provides functions to manipulate the contents of measurement set tables. The functions can be categorised as shown below.

Attaching to a Measurement Set

The simplest and most common way to attach an ms tool to a measurement set is to use the ms.open function which requires that you specify the name of the measurement set table.

The function ms.fromfits converts a UVFITS file to a measurement set table prior to attaching the ms tool to the newly created measurement set. The conversion step may take some time if the FITS file is large. However it only needs to be done once. The measurement set table is not deleted when you close the ms tool, using the close function, or exit CASA. And once the measurement set table is created it is much faster to attach an ms tool to it using the ms.open function.

Getting summary information

The summary function will display, in the logger, an overview of the measurement set. This will include listings of the fields, spectral windows & polarization setups used in the measurement set.

The range function will provide more quantitative information on the minimum, maximum or used values of specified parameters. When using this function you may need to do an initial selection, as described below, depending on whether the parameters you ask for change their shape. A list of parameters accepted by the range function is given in table 1.6 and this table also indicates when an initial selection is necessary.

The lister function provides a concise listing of the data in the measurement set.

The listhistory function lists the contents of the measurement set history table. The history table contains a record of changes made to the measurement set by flagger, calibrater, imager and other tools.

Selecting data

As described in the ms module documentation a measurement set can contain data with a variety of different shapes. Some of the functions in this tool require the data to be in a fixed shape. Before you can use these functions you
need to select a subset of the data in the measurement set where all the data has a fixed shape. There are two functions which can be used to do this. These are the selectinit and command functions.

The select function can be used to further refine which subset of the data will be used by the data access functions. This function allows you to select specific rows in a measurement set using a wide range of criteria.

The select function can only select whole rows in a measurement set. To select specific channels within a row you use the selectchannel function. Similarly to select specific polarizations you should use the selectpolarization function.

**Reading and writing data**

The getdata function is used to read data from the measurement set into casapy variables. You can select which columns of the measurement set main table you are interested in and only the subset of data specified using the selection functions described above will be retrieved. Any frequency averaging (see the selectfrequency function) and polarization conversion (see the selectpolarization function) will be done when you retrieve the data. The full power of casapy and other CASA tasks and tools, like plotxy, can then be used for adhoc inspection and calculations involving the data.

If the measurement set was opened for writing then the putdata function can be used to write the data back into the measurement set. When writing data back into the measurement set you cannot change the data shape or the coordinates of the data, only the numerical values. This means that you cannot write data that has been averaged in frequency or converted to different polarizations.

When using the getdata function with a large measurement set you need to be careful not to request too much data. The measurement set is stored on disk but casapy variables are stored in memory. To allow you to access large amounts of data in an ordered way the ms tool provides functions that allow you to iterate through the data in a convenient way.

If you need to step through the data in an orderly fashion, you can use the iteration functions. These allow you to set up an iteration order (iterinit), obtain the first iteration (iterorigin), go to the next iteration (iternext) and end the iteration prematurely (iterend). The iterorigin and iternext function set the currently selected table (as used by getdata and others) to the current iteration. At the end of the iteration, the original selection is restored.

You can iterate through a measurement set you have previously selected using select, but if you use select while iterating, you cannot get back the unselected iteration (without reiterating through the table until the current point).

The writehistory function allows messages to be appended to the measurement set history table should you wish to do so. The listhistory function lists your messages and those created by flagger, calibrater, imager and other tools.

**Conversions to FITS**

Just as the fromfits function will convert a UVFITS file to a measurement set the tofits function will convert a measurement set to a UVFITS file. Similarly a single dish measurement set ie., one with a FLOAT_DATA column rather
than a DATA column, can be converted to a single dish FITS file using the
tosdfits function.
You cannot read a UVFITS file into a measurement set and write it out as a
single dish FITS file or vice-versa.

**Concatenation**
The concatenate function can be used to append the data from one
measurement set to the end of another. As all the data is copied this function
may take some time if the measurement set to be copied is large. The
measurement set needs to be opened for writing for this to work.
The virtconcatenate function enables virtual concatenation, i.e. the data is not
rewritten but just reindexed such that the two input MSs have the same
subtables. They can then be turned into a multi-MS.

**Sorting the main table by time**
The timesort function permits you to sort the MS main table by time in
ascending order. This can be useful after a concatenation.

**Sorting the main table by a custom set of columns**
The sort function permits you to sort the MS main table by a custom set of
columns in ascending order. This can be useful to compare tables generated in
different ways (e.g.: cvel and mastransform)

**Splitting** The split function allows you to make a new ms from a subset of
the actual ms.

Table 1.6: Items recognized by the range, select, getdata and
putdata functions.
Items marked with a † are only available in interferometric mea-
surement sets.
Items marked with a ‡ are only available in interferometric mea-
surement sets
that have been processed with calibrator or imager. Items marked
with a * do
not require all the data in the selected measurement set to have
the same shape.

<table>
<thead>
<tr>
<th>&amp; range</th>
<th>select</th>
<th>getdata</th>
<th>putdata</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>amplitude†</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>corrected_amplitude‡</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>model_amplitude‡</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>ratio_amplitude‡</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>residual_amplitude‡</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>obs_residual_amplitude‡</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>antenna1</td>
<td>*</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>antenna2</td>
<td>*</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>antennas</td>
<td>*</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>array_id</td>
<td>*</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>range</td>
<td>select</td>
<td>getdata</td>
<td>putdata</td>
</tr>
<tr>
<td>------------------</td>
<td>-------</td>
<td>--------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>axis_info</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>chan_freq</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>corr_names</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>corr_types</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>data†</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>corrected_data‡</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>model_data‡</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>ratio_data‡</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>residual_data‡</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>obs_residual_data‡</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>feed1</td>
<td>*</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>feed2</td>
<td>*</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>field_id</td>
<td>*</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>fields</td>
<td>*</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>flag</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>flag_row</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>flag_sum</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>ha</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>ifr_number</td>
<td>*</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>imaginary†</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>corrected_imaginary‡</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>model_imaginary‡</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>ratio_imaginary‡</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>residual_imaginary‡</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>obs_residual_imaginary‡</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>imaging_weight‡</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>last</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>num_corr</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>num_chan</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>phase†</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>corrected_phase‡</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>model_phase‡</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>ratio_phase‡</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>residual_phase‡</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>obs_residual_phase‡</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>phase_dir</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>real†</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>corrected_real‡</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>model_real‡</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>ratio_real‡</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>residual_real‡</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>obs_residual_real‡</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>ref_frequency</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>rows</td>
<td>*</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Continued from previous page

<table>
<thead>
<tr>
<th></th>
<th>range</th>
<th>select</th>
<th>getdata</th>
<th>putdata</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan_number</td>
<td>*</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>sigma</td>
<td>*</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>sigma of the data</td>
</tr>
<tr>
<td>data_desc_id</td>
<td>*</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>*</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>MJD time range in seconds</td>
</tr>
<tr>
<td>times</td>
<td>*</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>list of MJD timeslots</td>
</tr>
<tr>
<td>ut</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>add UT to axis_info</td>
</tr>
<tr>
<td>uvw</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>uvw vector</td>
</tr>
<tr>
<td>u</td>
<td>*</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>u coordinate</td>
</tr>
<tr>
<td>v</td>
<td>*</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>v coordinate</td>
</tr>
<tr>
<td>w</td>
<td>*</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>w coordinate</td>
</tr>
<tr>
<td>uvdist</td>
<td>*</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>uv distance</td>
</tr>
<tr>
<td>weight</td>
<td>*</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>weight of the data</td>
</tr>
</tbody>
</table>

Example:

```python
ms.open("3C273XC1.MS",nomodify=False)
ms.selectinit(datadescid=0)
ms.select({'antenna1':[1,3,5],'uvdist':[1200,1900]})
rec=ms.getdata(['weight','data'])
# modify rec['weight'] and rec['data'] values as desired
ms.putdata(rec)
ms.close()
```

We open the MS for writing, select an array and spectral window and then select a few antennas and a uv range. We then get out the weight values and the data. We change these values in casapy and then write them back to the measurement set. Finally, we close the ms, causing the values to be written back to disk.

Methods

- **fromfits**
  - Create a measurement set from a uvfits file
- **fromfitsidi**
  - Create a measurement set from a fits-idi file
- **listfits**
- **createmultims**
- **ismultims**
- **getreferencedtables**
- **getfielddirmeas**
- **nrow**
- **iswritable**
- **open**
- **reset**
- **close**

Returns the direction measure from the given FIELD table column and row

Returns True if the underlying Table is writable

Attach the ms tool to a measurement set table

Re-attach the tool to the original MS.

Detach the ms tool from the measurement set table
name

tofits

summary

getscansummary

getspectralwindowinfo

listhistory

writehistory

statistics

statistics2

range

list

metadata

selectinit

msselect

msselectedindices

select

selecttaql

selectchannel

selectpolarization

regridspw

cvel

cvelfreqs

getdata

putdata

concatenate

testconcatenate

virtconcatenate

timesort

sort

contsub

statwt

split

partition

iterinit

iterorigin

iternext

iterend

fillbuffer

diffbuffer

getbuffer

clipbuffer

asdmref

setbufferflags

writebufferflags

clearbuffer

continuumsub

Name of the measurement set table the tool is attached to.
Convert a measurement set to a uvfits file
(PARTIALLY IMPLEMENTED!!!) Summarize the measurement set
Get the summary of the ms
Get a summary of the spectral windows
List history of the measurement set
Add a row of arbitrary information to the measurement set history table
Get statistics on the selected measurement set
Get statistics on the selected measurement set
Get the range of values in the measurement set
List measurement set visibilities
Get the MS metadata associated with this MS.
Initialize the selection of an ms
Use the MSSelection module for data selection.
Return the selected indices of the MS database. The keys in the record are the same
Select a subset of the measurement set.
Select a subset of the measurement set.
Select and average frequency channels
Selection and conversion of polarizations
transform spectral data to different reference frame and/or regrid the frequency channels
transform spectral data to different reference frame and/or regrid the frequency channels
calculate the transformed grid of the SPW obtained by combining a given set of SPWs
Read values from the measurement set.
Write new values into the measurement set
Concatenate two measurement sets
Concatenate only the subtables of two measurement sets excluding the POINTING
Concatenate two measurement sets virtually
Sort the main table of an MS by time
Sort the main table of an MS using a custom set of columns
Subtract the continuum from the visibilities
Set WEIGHT and SIGMA from the scatter of the visibilities
make a new ms from a subset of an existing ms, adjusting subtables and indices
make a new ms from a subset of an existing ms, without changing any subtables
Initialize for iteration over an ms
Set the iterator to the start of the data.
Advance the iterator to the next lot of data
End the iteration and reset the selected table
DEPRECATED: Fill the internal buffer with data and flags.
DEPRECATED: Differentiate or difference the internal buffer.
DEPRECATED: Return the internal buffer as a Record for access from the interpreter
DEPRECATED: Clip the internal buffer with specified limits.
Test if the MS was imported with option lazy=True in importasdm and optionally
DEPRECATED: Set the flags in the buffer
DEPRECATED: Write the flags in the internal buffer back to the table.
DEPRECATED: Clear the internal buffer.
Continuum fitting and subtraction in uv plane
done  Closes the ms tool
msseltoindex  Returns ids of the selection used
hanningsmooth  Hanning smooth the frequency channels to remove Gibbs ringing.
uvsub  Subtract model from the corrected visibility data.
addephemeris  Connect an ephemeris table with the MS FIELD table
ngetdata  Read values from the measurement set. Use this method instead of the older getdata() method
niterinit  Initialize for iteration over an ms. Use this method instead of the older iterinit() method
niterorigin  Set the iterator to the start of the data. Use this method instead of the older iterorigin() method
niternext  Advance the iterator to the next lot of data. Use this method instead of the older iternext() method
niterend  Query if there are more iterations left in the iterator. Use this method instead of the older iterend() method
ms.fromfits.html

**ms.fromfits - Function**

Create a measurement set from a uvfits file

**Description**

This function will convert a UVFITS file to a measurement set table and then open the measurement set table. The newly created measurement set table will continue to exist after the tool has been closed. Setting the lock argument to True will permanently lock the table preventing other processes from writing to the measurement set. Unless you expect this to happen, and want to prevent it, you should leave the lock argument at the default value which implies auto-locking.

Note that the variety of fits files that fromfits is able to interpret correctly is limited mostly to files similar to those produced by classic AIPS. In particular, it understands only binary table extensions for the antenna (AN), frequency (FQ) and source (SU) information and ignores other extensions.

This function returns True if it successfully attaches the ms tool to a newly created Measurement Set or False if something went wrong, like an error in a file name.

**Arguments**
**Inputs**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msfile</td>
<td>Filename for the newly created measurement set</td>
</tr>
<tr>
<td>fitsfile</td>
<td>uvfits file to read</td>
</tr>
<tr>
<td>nomodify</td>
<td>open for read access only</td>
</tr>
<tr>
<td>lock</td>
<td>lock the table for exclusive use</td>
</tr>
<tr>
<td>obstype</td>
<td>specify the observation type: 0=standard, 1=fastmosaic, requiring small tiles</td>
</tr>
<tr>
<td>host</td>
<td>host to start ms tool on (IGNORED!!!)</td>
</tr>
<tr>
<td>forcenewserver</td>
<td>start a new server tool (IGNORED!!!)</td>
</tr>
<tr>
<td>antnamescheme</td>
<td>For VLA only, antenna name scheme, old style is just antenna number, new</td>
</tr>
<tr>
<td></td>
<td>style prepends VA or EV</td>
</tr>
</tbody>
</table>

**Allowed:**
- `msfile`: string
- `fitsfile`: string
- `nomodify`: bool
- `lock`: bool
- `obstype`: int
- `host`: string
- `forcenewserver`: bool
- `antnamescheme`: string

**Default:**
- `msfile`: string
- `fitsfile`: string
- `nomodify`: bool
- `lock`: bool
- `obstype`: int
- `host`: string
- `forcenewserver`: bool
- `antnamescheme`: string

**Returns**

bool

**Example**

```python
ms.fromfits("3C273XC1.MS", "3C273XC1.fits")
```
ms.fromfitsidi.html

**ms.fromfitsidi - Function**

1.3.1 Create a measurement set from a fits-idi file

**Description**

This function will convert a UVFITS file to a measurement set table and then open the measurement set table. The newly created measurement set table will continue to exist after the tool has been closed.

Setting the lock argument to True will permanently lock the table preventing other processes from writing to the measurement set. Unless you expect this to happen, and want to prevent it, you should leave the lock argument at the default value which implies auto-locking.

Note that the variety of fits files that fromfits is able to interpret correctly is limited mostly to files similar to those produced by classic AIPS. In particular, it understands only binary table extensions for the antenna (AN), frequency (FQ) and source (SU) information and ignores other extensions.

This function returns True if it successfully attaches the ms tool to a newly created Measurement Set or False if something went wrong, like an error in a file name.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowable</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>msfile</td>
<td>Filename for the newly created measurement set</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>fitsfile</td>
<td>fits-idi file to read</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>nomodify</td>
<td>open for read access only</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>lock</td>
<td>lock the table for exclusive use</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>obstype</td>
<td>specify the observation type: 0=standard, 1=fastmosaic, requiring small tiles in the measurement set</td>
<td>int</td>
<td>0</td>
</tr>
</tbody>
</table>
Returns
c

Example

ms.fromfits("3C273XC1.MS", "3C273XC1.fits")

694
ms.listfits.html

**ms.listfits - Function**  

1.3.1

**Description**

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>fitsfile</td>
<td>list HDU and typical data rows in a uvfits file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>string</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

ms.listfits('ngc5921.fits')
ms.createmultims.html

ms.createmultims - Function

1.3.1

Description

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputTableName</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>tables</td>
<td>stringArray</td>
<td></td>
</tr>
<tr>
<td>subtables</td>
<td>stringArray</td>
<td></td>
</tr>
</tbody>
</table>

| nomodify      | bool         | true         |
| lock          | bool         | false        |
| copysubtables | bool         | false        |
| omitsubtables | stringArray  | false        |

Returns

bool

Example

696
ms.ismultims.html

ms.ismultims - Function
[1.3.1]

Description

Arguments

| Inputs |

Returns
bool

Example
ms.getreferencedtables.html

**ms.getreferencedtables - Function**

[1.3.1]

### Description

### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
</table>

### Returns

`stringArray`

### Example
ms.getfielddirmeas - Function

1.3.1 Returns the direction measure from the given FIELD table column and row.

Description

This function returns the direction measures from the given direction column of the MS FIELD table as a either a measure dictionary or sexigesimal string representation.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of the direction column in the FIELD table</th>
</tr>
</thead>
<tbody>
<tr>
<td>dircolname</td>
<td>Name of the direction column in the FIELD table</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>PHASE_DIR</td>
</tr>
<tr>
<td>fieldid</td>
<td>Field ID, starting at 0</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td>time</td>
<td>(optional) time for ephemeris access (in seconds, as in</td>
</tr>
<tr>
<td></td>
<td>Main table TIME column)</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td>format</td>
<td>Output format. Either &quot;measure&quot; (measure dictionary) or &quot;string&quot; (sexigesimal representation). Minimum match supported.</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>measure</td>
</tr>
</tbody>
</table>

Returns

anyvariant

Example
**ms.nrow - Function**

Returns the number of rows in the measurement set

**Description**

This function returns the number of rows in the measurement set. If the optional argument selected is set to True, it returns the number of currently selected rows, otherwise it returns the number of rows in the original measurement.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>selected</td>
<td>return number of selected rows</td>
</tr>
<tr>
<td>allowed</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

int

**Example**

```python
ms.open('3C273XC1.MS')
print "Number of rows in ms =", ms.nrow()
#Number of rows in ms = 7669
```
ms.iswritable.html

**ms.iswritable - Function**

1.3.1 Returns True is the underlying Table is writable

**Description**

This function returns True if the underlying MeasurementSet Table was opened for writing/update.

**Arguments**

**Returns**

bool

**Example**

```python
ms.open('3C273XC1.MS',nomodify=False)
if ms.iswritable():
    print "MeasurementSet is writable"
else:
    print "MeasurementSet is readonly"
#MeasurementSet is writable
```
ms.open.html

**ms.open - Function**

1.3.1 Attach the ms tool to a measurement set table

**Description**

Use this function when you have detached (using the close function) the ms tool from a measurement set table and wish to reattach to another measurement set table.

If check=true, additional referential integrity checks on the MS are run. If any of these fail, an exception is thrown and the MS is not open (since it is not a valid MS).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of the measurement set table to open</th>
</tr>
</thead>
<tbody>
<tr>
<td>thems</td>
<td>string</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>true</td>
</tr>
</tbody>
</table>

| nomodify | prevent changes to the measurement set |
| allowed:  | bool                                   |
| Default:  | true                                   |

| lock | lock the table for exclusive use by this tool |
| allowed: | bool |
| Default: | false |

| check | Run additional internal integrity checks on the MS. |
| allowed: | bool |
| Default: | false |

**Returns**

bool

**Example**

```
ms.open('3C273XC1.MS')
```
ms.close()
ms.open("anotherms", nomodify=False, lock=False)
ms.reset.html

**ms.reset - Function**

1.3.1 Re-attach the tool to the original MS.

### Description

This function re-attaches the `ms` tool to the original MS, effectively discarding any prior operations, in particular any data selection operations using msselect function.

### Arguments

### Returns

bool

### Example
ms.close.html

**ms.close - Function**

1.3.1 Detach the ms tool from the measurement set table

**Description**

This function detaches the ms tool from the associated measurement set table after flushing all the cached changes. After calling this function the ms tool is not associated with any measurement set and using any function other than open or fromfits will result in an error message being sent to the logger. This function can be useful to avoid synchronization problems which can occur when different processes have the same ms open.

**Arguments**

**Returns**

bool

**Example**

See the example for the open function.
**ms.name - Function**

Name of the measurement set table the tool is attached to.

**Description**

This function returns the name of the measurement set table that is being manipulated. If the ms tool is not attached to any measurement set then this function will return the string “none”.

**Arguments**

**Returns**

string

**Example**

```python
ms.open('3C273XC1.MS')
print "Processing file", ms.name()
```
**ms.tofits - Function**

Convert a measurement set to a uvfits file

**Description**

This function writes a UVFITS file that contains the data in the measurement set associated with this tool. The FITS file is always written in floating point format and the data are always stored in the primary array of the FITS file. If the measurement set has been imaged or calibrated in CASA, it may contain additional data columns. You need to select ONE of these columns to be written to the FITS file. The possible options are:

- **observed**: This is the raw data as collected by the telescope. All interferometric measurement sets must contain this column. A synonym for ‘observed’ is ‘data’.
- **corrected**: This is the calibrated data. A synonym for ’corrected’ is ’corrected_data’.
- **model**: This is the visibilities that would be measured using the current model of the sky. A synonym for ’model’ is ’model_data’.

The parsing of these strings is case insensitive. If any other option is specified then the observed data will be written.

By default a single-source UVFITS file is written, but if the measurement set contains more than one field or if you set the multisource argument to True a multi-source UVFITS file will be written. Because of limitations in the UVFITS format you have to ensure that the data shape is fixed for all the data you intend to write to one FITS file. See the general description of this tool for how you can select data to meet this condition.

The combinespw argument is used to control whether data from different spectral windows will be written as different entries in the FITS FQ (frequency) table or combined as different IF’s within one entry in the FQ table. You should normally only set this to True if you know that the data from different spectral windows were observed simultaneously, and the data in the measurement set can be equally divided between all the spectral windows (i.e. each window should have the same width). Use of this switch is recommended for data to be processed in classic AIPS and difmap (if possible, e.g., standard dual IF observations).

The padwithflags argument is only relevant if combinespw is True. If true, it will fill in data that is ‘missing’ with flags to fit the IF structure. This is
appropriate if the MS had a few frequency-dependent flags applied, and was then time-averaged by split. If the spectral windows were observed at different times, padwithflags=True will add a large number of flags, making the output file significantly longer. It does not yet support spectral windows with different widths.

The FITS GC (gain curve) and TY (system temperature) tables can be optionally written by setting the writesyscal argument to True. This is a rather WSRT-specific operation at the moment and may not work correctly for measurement sets from other telescopes.

The width argument is for channel averaging while outputting the data to the fits file. The default values of 1 will copy the channels of the input as is. The start channel number is the first channel of the spw expression. The number of output channels is determined by spw expression and the width. The width is the number of channels of the input data to make 1 channel of the output data. One may overwrite the specified output file if it exists by specifying overwrite=True.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fitsfile</td>
<td>Name of the new uvfits file</td>
</tr>
<tr>
<td>column</td>
<td>Data column to write, see above for options</td>
</tr>
<tr>
<td>field</td>
<td>Field ids (0-based) or fieldnames to split out</td>
</tr>
<tr>
<td>spw</td>
<td>Spectral windows to split</td>
</tr>
<tr>
<td>width</td>
<td>number of input channels to average</td>
</tr>
<tr>
<td>baseline</td>
<td>Antenna names or Antenna indices to select</td>
</tr>
<tr>
<td>time</td>
<td>Limit data selected to be within a given time range. Syntax is the defined in the msselection link</td>
</tr>
<tr>
<td>scan</td>
<td>Limit data selected on scan numbers. Syntax is the defined in the msselection link</td>
</tr>
<tr>
<td>uvrange</td>
<td>Limit data selected on uv distance. Syntax is the defined in the msselection link</td>
</tr>
<tr>
<td>taql</td>
<td>For the TAQL experts, flexible data selection using the TAQL syntax</td>
</tr>
<tr>
<td>writesyscal</td>
<td>Write GC and TY tables</td>
</tr>
<tr>
<td>multisource</td>
<td>Write in multisource format</td>
</tr>
<tr>
<td>combinespw</td>
<td>Export spectral windows as IFs</td>
</tr>
<tr>
<td>writestation</td>
<td>Write station name instead of antenna name</td>
</tr>
<tr>
<td>padwithflags</td>
<td>If combinespw==True, pad data with flags to fit IFs</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite output file if it exists?</td>
</tr>
</tbody>
</table>

| allowed                      | string                      |
| allowed                      | string                      |
| allowed                      | any                        |
| allowed                      | any                        |
| allowed                      | int                        |
| allowed                      | any                        |
| allowed                      | string                     |
| allowed                      | bool                       |
| allowed                      | bool                       |
| allowed                      | bool                       |
| allowed                      | bool                       |
| allowed                      | int                        |
| allowed                      | bool                       |
| allowed                      | bool                       |
| allowed                      | string                     |
| allowed                      | bool                       |

| Default                      |                             |
| Default                      | corrected                  |
| Default                      | variant                    |
| Default                      | variant                    |
| Default                      | 1                          |
| Default                      | variant                    |
| Default                      | string                     |
| Default                      | variant                    |
| Default                      | variant                    |
| Default                      | variant                    |
| Default                      | variant                    |
| Default                      | string                     |
| Default                      | variant                    |
| Default                      | false                      |
| Default                      | false                      |
| Default                      | false                      |
| Default                      | false                      |
| Default                      | false                      |
| Default                      | false                      |
Returns

bool

Example

ms.open('3C273XC1.MS')
ms.tofits('3C273XC1.fits', column='DATA');
ms.done()

This example writes the observed data of a measurement set to a \uvfits\ file.

ms.open('big.ms')
ms.tofits('part.fits', column='CORRECTED', field=[0,1], spw=[2]);
ms.done()

This example writes part (the first 2 fields and the third spectral window) of the corrected data to the fits file.
ms.summary.html

**ms.summary - Function**

1.3.1 (PARTIALLY IMPLEMENTED!!!) Summarize the measurement set

**Description**

This method will print a summary of the measurement set to the system logger. The verbose argument provides some control on how much information is displayed.

For especially large datasets, the cachesize parameter can be increased for possibly better performance.

This method can also return, in the header argument, a record containing the following fields.

- **nrow** Number of rows in the measurement set
- **name** Name of the measurement set

**DESCRIPTION OF ALGORITHM TO CALCULATE THE NUMBER OF UNFLAGGED ROWS** The number of unflagged rows will only be computed if listunflis True. The number of unflagged rows (the nUnflRows columns in the scans and fields portions of the listing) is calculated by summing the fractional unflagged bandwidth for each row (and hence why the number of unflagged rows, in general, is not an integer). Thus a row which has half of its total bandwidth flagged contributes 0.5 rows to the unflagged row count. A row with 20 of 32 channels of homogeneous width contributes \(20/32 = 0.625\) rows to the unflagged row count. A row with a value of False in the FLAG_ROW column is not counted in the number of unflagged rows.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>verbose</td>
<td>Produce verbose logging output</td>
</tr>
<tr>
<td>listfile</td>
<td>Output file</td>
</tr>
<tr>
<td>listunfl</td>
<td>List unflagged row counts? If true, it can have significant negative</td>
</tr>
<tr>
<td></td>
<td>performance impact.</td>
</tr>
<tr>
<td>cachesize</td>
<td>EXPERIMENTAL. Maximum size in megabytes of cache in which data structures</td>
</tr>
<tr>
<td></td>
<td>can be held.</td>
</tr>
<tr>
<td>overwrite</td>
<td>If True, tacitly overwrite listfile if it exists.</td>
</tr>
<tr>
<td>wantreturn</td>
<td>If true, construct a record containing summary info and return it, else</td>
</tr>
<tr>
<td></td>
<td>return nothing. If you don’t need the record and just want the log output,</td>
</tr>
<tr>
<td></td>
<td>setting this to False will provide a small performance increase.</td>
</tr>
</tbody>
</table>

| Returns       | record                                                                     |

| Example       |                                                                            |
|---------------|                                                                            |
|               | ms.open('3C273XC1.MS')                                                       |
|               | outr=ms.summary(verbose=True)                                               |
|               | #print the begining of observation in this ms                               |
|               | print qa.time(qa.quantity(outr['header']['BeginTime'],'d'), form='ymd')     |
|               | #print a dictionary of the info of scan 1                                  |
|               | outr['header']['scan_1']                                                    |

This example will send a verbose summary of the measurement set to the logger.
ms.getscansummary.html

**ms.getscansummary - Function**

1.3.1 Get the summary of the ms

**Description**

This function will return a summary of the main table as a structure

**Arguments**

**Returns**

record

**Example**

```python
ms.open('3C273XC1.MS')
scanInfo = ms.getscansummary()
```
ms.getspectralwindowinfo.html

**ms.getspectralwindowinfo - Function**

1.3.1 Get a summary of the spectral windows

**Description**

This method will get a summary of the spectral window actually used in this ms. To be precise those reference by the data description table.

**Arguments**

**Returns**

record

**Example**

```python
ms.open('3C273XC1.MS')
spwInfo = ms.getspectralwindowinfo()
```
ms.listhistory.html

**ms.listhistory - Function**

1.3.1 List history of the measurement set

**Description**

This function lists the contents of the measurement set history table.

**Arguments**

**Returns**

bool

**Example**

```python
ms.open('3C273XC1.MS')
ms.listhistory()
```

The history table contents are listed in the logger.
ms.writehistory.html

**ms.writehistory - Function**

1.3.1 Add a row of arbitrary information to the measurement set history table

**Description**

This function adds a row to the history table of the specified measurement set containing any message that the user wishes to record. By default the history entry is written to the history table of the measurement set that is currently open, the message origin is recorded as 'MSHistoryHandler::addMessage()’, the originating application is ‘ms’ and the input parameters field is empty.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>Message to be recorded in message field</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>parms</td>
<td>String to be written to input parameter field</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>origin</td>
<td>String to be written to origin field</td>
<td>string</td>
<td>MSHistoryHandler::addMessage()</td>
</tr>
<tr>
<td>msname</td>
<td>name of selected measurement set</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>app</td>
<td>String to be written to application field</td>
<td>string</td>
<td>ms</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

719
ms.open('3C273XC1.MS')
ms.writehistory('an arbitrary history message')
ms.listhistory()

A row is appended to the measurement set history table.
ms.statistics.html

**ms.statistics - Function**

Get statistics on the selected measurement set

**Description**

This function computes descriptive statistics on the measurement set. It returns the statistical values as a python dictionary. The given column name must be a numerical column. If it is a complex valued column, the parameter `complex_value` defines which derived real value is used for the statistics computation.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>column</td>
<td>Column name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>complex_value</td>
<td>Which derived value to use for complex columns (amp, amplitude, phase, imag, real, imaginary)</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>useflags</td>
<td>Use the data flags</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>spw</td>
<td>Spectral Window Indices or names : example : '1,2'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>field</td>
<td>Field indices or source names : example : '2,3C48'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>baseline</td>
<td>Baseline number(s): example: ”2&amp;3:4&amp;5”</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>uvrange</td>
<td>UV-distance range, with a unit : example : '2.0-3000.0 m'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>Time range, as MJDs or date strings : example : 'xx.x.x.x~yy.y.y.y'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>correlation</td>
<td>Correlations/polarizations : example : 'RR,LL,RL,LR,XX,XY,YX'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>scan</td>
<td>Scan number : example : '1,2,3'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>array</td>
<td>Array Indices or names : example : 'VLAA'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>obs</td>
<td>Observation ID(s): examples : ” or '1~3'</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

record
Example

```python
ms.open("3C273XC1.MS")
ms.statistics(column="DATA", complex_value='amp', field="2")
```
ms.statistics2.html

**ms.statistics2 - Function**

1.3.1 Get statistics on the selected measurement set

**Description**

This function computes descriptive statistics on the measurement set. It returns the statistical values as a python dictionary. The given column name must be a numerical column. If it is a complex valued column, the parameter `complex_value` defines which derived real value is used for the statistics computation.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>column</td>
<td>Column name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>complex_value</td>
<td>Which derived value to use for complex columns (amp, amplitude, phase, imag, real, imaginary)</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>useflags</td>
<td>Use the data flags</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>useweights</td>
<td>Use the data weights</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>spw</td>
<td>Spectral Window Indices or names : example : '1,2'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>field</td>
<td>Field indices or source names : example : '2,3C48'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>baseline</td>
<td>Baseline number(s) : example: &quot;2&amp;3;4&amp;5&quot;</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>uvrangge</td>
<td>UV-distance range, with a unit : example : '2.0-3000.0 m'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>Time range, as MJDs or date strings : example : 'xx.x.x.x.x~yy.y.y.y.y'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>correlation</td>
<td>Correlations/polarizations : example : 'RR,LL,RL,LR,XX,YY,XY,YX'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>scan</td>
<td>Scan number : example : '1,2,3'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>intent</td>
<td>Scan intents : example : '<em>AMPL</em>,<em>PHASE</em>'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>array</td>
<td>Array Indices or names : example : 'VLAA'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>obs</td>
<td>Observation ID(s): examples : &quot; or '1~3'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>reportingaxes</td>
<td>Statistics reporting axes: example : 'ddid,field'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>timeaverage</td>
<td>Average data in time</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>timebin</td>
<td>Time averaging interval</td>
<td>string</td>
<td>0s</td>
</tr>
</tbody>
</table>
**Returns**
record

**Example**

```python
ms.open("3C273XC1.MS")
ms.statistics2(column="DATA", complex_value='amp', field="2")
```
ms.range.html

**ms.range - Function**

1.3.1 Get the range of values in the measurement set

**Description**

This function will return the range of values in the currently selected measurement set for the items specified. Possible items include most scalar columns, interferometer number (1000*antenna1+antenna2), uvdistance, u, v, w, amplitude, phase, real and imaginary components of the data (and corrected and model versions of these - if these columns are present). See the table at the top of the document to find the exact list.

You specify items in which you are interested using a string vector where each element is a case insensitive item name. This function will then return a record that has fields corresponding to each of the specified items. Each field will contain the range of the specified item. For many items the range will be the minimum and maximum values but for some it will be a list of unique values. Unrecognized items are ignored.

By default the FLAG column is used to exclude flagged data before any ranges are determined, but you can set useflags=False to include flagged data in the range. However, if you average in frequency, flagging will still be applied.

You can influence the memory use and the reading speed using the blocksize argument - it specifies how big a block of data to read at once (in MB). For large datasets on machines with lots of memory you may speed things up by setting this higher than the default (10 MB).

For some items, as indicated with an § in table 1.6 (in the general description of this tool), you need to call selectinit to select a portion of the data with a unique shape prior to calling this function.

Items prefixed with corrected, model, residual or obs_residual and the imaging_weight item are not available unless your measurement set has been processed either with the imager or calibrator tools.

**Arguments**
**Inputs**

<table>
<thead>
<tr>
<th>items</th>
<th>Item names</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
</tbody>
</table>

| useflags | Use the data flags               |
| allowed  | bool                             |
| Default  | true                             |

| blocksize | Set the blocksize in MB          |
| allowed   | int                              |
| Default   | 10                               |

**Returns**

record

**Example**

```python
ms.open("3C273XC1.MS")
ms.selectinit(datadescid=0)
ms.range(["time","uvdist","amplitude","antenna1"])
#{'amplitude': array([ 2.60339398e-02, 3.38518333e+01]),
 '# 'antenna1': array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13,
 # 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]),
 # 'time': array([ 4.12162940e+09, 4.12164267e+09]),
 # 'uvdist': array([ 46.26912101, 3727.97385983])}
```

In this example the minimum and maximum observation times, uvdistances, data amplitudes are returned as well as a list of all the antennas in the antenna1 column.

For this dataset the selectinit function did not need to be called as all the data is of one shape.
ms.lister.html

**ms.lister - Function**

[1.3.1] List measurement set visibilities

**Description**

This tool lists measurement set visibility data under a number of input selection conditions. The measurement set data columns that can be listed are: the raw data, corrected data, model data, and residual (corrected - model) data.

The output table format is dynamic. Field, Spectral Window, and Channel columns are not displayed if the column contents are uniform. For example, if “spw = ‘1’ ” is specified, the spw column will not be displayed. When a column is not displayed, a message is sent to the logger and terminal indicating that the column values are uniform and listing the uniform value.

Table column descriptions:

<table>
<thead>
<tr>
<th>Column Name &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date/Time &amp; Average date and time of data sample interval</td>
</tr>
<tr>
<td>Intrf &amp; Interferometer baseline (antenna names)</td>
</tr>
<tr>
<td>UVDist &amp; uv-distance (units of wavelength)</td>
</tr>
<tr>
<td>Fld &amp; Field ID</td>
</tr>
<tr>
<td>SpW &amp; Spectral Window ID</td>
</tr>
<tr>
<td>Chn &amp; Channel number</td>
</tr>
<tr>
<td>(Correlated &amp; Correlated polarizations (eg: RR, LL, XY) polarization) &amp; Sub-columns are: Amp, Phs, Wt, F</td>
</tr>
<tr>
<td>Amp &amp; Visibility amplitude</td>
</tr>
<tr>
<td>Phs &amp; Visibility phase</td>
</tr>
<tr>
<td>Wt &amp; Weight of visibility measurement</td>
</tr>
<tr>
<td>F &amp; Flag: ‘F’ = flagged datum; ‘ ’ = unflagged</td>
</tr>
</tbody>
</table>

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Options</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>Output options (not yet implemented)</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>datacolumn</td>
<td>Column to list: data, model, corrected, residual</td>
<td>string</td>
<td>data</td>
</tr>
<tr>
<td>field</td>
<td>Fields</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>spw</td>
<td>Spectral Windows</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>antenna</td>
<td>Antenna/Baselines</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>timerange</td>
<td>Time range</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>correlation</td>
<td>Polarization correlations</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>scan</td>
<td>Scan</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>feed</td>
<td>Feed (not yet implemented)</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>array</td>
<td>Array</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>observation</td>
<td>Select by observation ID(s)</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>uvrange</td>
<td>Uv-distance (output units: wavelength)</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>average</td>
<td>Average mode (not yet implemented)</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>showflags</td>
<td>Showflags (not yet implemented)</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>msselect</td>
<td>TaQL expression</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>pagerows</td>
<td>Rows per page</td>
<td>int</td>
<td>50</td>
</tr>
<tr>
<td>listfile</td>
<td>Output file</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>
Returns
bool

Example

```python
ms.open('AZ136.ms')
ms.lister()
```

These commands yeild the following listing:

<table>
<thead>
<tr>
<th>Date/Time: 2001/12/01/</th>
<th>RR: Instr</th>
<th>UVDist</th>
<th>Fld SpW</th>
<th>Amp</th>
<th>Phs</th>
<th>Wt F</th>
<th>Amp</th>
<th>Phs</th>
<th>Wt F</th>
<th>Amp</th>
</tr>
</thead>
<tbody>
<tr>
<td>19:30:05.0 0- 1 1400 0 0:</td>
<td>0.002-102.7</td>
<td>229035 F</td>
<td>0.003-178.3</td>
<td>239694 F</td>
<td>0.001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19:30:05.0 0- 2 7203 0 0:</td>
<td>0.002 127.3</td>
<td>267464 F</td>
<td>0.001 165.0</td>
<td>305192 F</td>
<td>0.003</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19:30:05.0 0- 3 9621 0 0:</td>
<td>0.002 -55.9</td>
<td>179652 F</td>
<td>0.002 -27.1</td>
<td>230130 F</td>
<td>0.001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19:30:05.0 0- 4 1656 0 0:</td>
<td>0.001 133.3</td>
<td>199677 F</td>
<td>0.002 80.6</td>
<td>258140 F</td>
<td>0.001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19:30:05.0 0- 5 3084 0 0:</td>
<td>0.002 -18.4</td>
<td>197565 F</td>
<td>0.001 -83.1</td>
<td>228541 F</td>
<td>0.002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19:30:05.0 0- 6 5020 0 0:</td>
<td>0.001-173.2</td>
<td>236475 F</td>
<td>0.002-104.0</td>
<td>257575 F</td>
<td>0.003</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19:30:05.0 0- 7 12266 0 0:</td>
<td>0.003-34.6</td>
<td>264977 F</td>
<td>0.002 5.3</td>
<td>280113 F</td>
<td>0.001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notice that the channel column is not displayed. This measurement set contains only one channel; since the channel column values are uniform, the channel column is not displayed. Instead, message "All selected data has CHANNEL = 0" is sent to the console.
Get the MS metadata associated with this MS.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cachesize</td>
<td>Maximum cache size, in megabytes, to use.</td>
</tr>
<tr>
<td></td>
<td>allowed: float</td>
</tr>
<tr>
<td></td>
<td>Default: 50</td>
</tr>
</tbody>
</table>

**Returns**

msmetadata

**Example**

```python
# get the number of spectral windows in the specified MS
ms.open"my.ms")
metadata = ms.metadata()
ms.done()
nspw = metadata.nspw()
metadata.done()
```
ms.selectinit - Function

Description

A measurement set can contain data with a variety of different shapes (as described in the overall description to this tool). To allow functions to return data in fixed shape arrays you need to select, using this function, rows that contain the same data shape. You do not need to use this function if all the data in your measurement set has only one shape. The DATA_DESC_ID column in the measurement set contains a value that maps to a particular row in the POLARIZATION and SPECTRAL_WINDOW subtables. Hence all rows with the same value in the DATA_DESC_ID column must have the same data shape. To select all the data where the DATA_DESC_ID value is $N$ you call this function with the datadescid argument set to $N$.

It is possible to have a measurement set with differing values in the DATA_DESC_ID column but where all the data is a fixed shape. For example this will occur if the reference frequency changes but the number of spectral channels is fixed. In cases like this all the data can be selected, using this function with an argument of zero. If the data shape does change and you call this function with a datadescid set to zero the return value will be False. In all other cases it will be True.

To return to the completely unselected measurement set, set the reset argument to True. This will allow you to access the full range of rows in the measurement set, rather than just the selected measurement set. The datadescid must always be a non-negative integer.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>datadescid</td>
<td>Data description id</td>
<td>allowed: int</td>
</tr>
<tr>
<td>reset</td>
<td>Reset to unselected state</td>
<td>allowed: bool</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Default</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>datadescid</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>reset</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>
Example

```python
ms.open("3C273XC1.MS")
ms.selectinit(datadescid=0)
print ms.range(['uvdist'])
ms.selectinit(reset=True)
print ms.range("uvdist")
```

In this example we display the range of uv distances for the data in the specified measurement set. The first print statement will only use data where the DATA\_DESC\_ID column is 0. This will correspond to a specific spectral window and polarization setup. The second print statement will print the range of uv distances for all the data in the measurement set (which is the same in this case).
ms.msselect.html

**ms.msselect - Function**

1.3.1 Use the MSSelection module for data selection.

**Description**

A return value of True implies that the combination of all selection expressions resulted in a non-Null combined TaQL expression. False implies that the combined TaQL could not be formed (i.e. it is Null, and the "selected MS" will be the same as the input MS).

The details of selection expressions are described in the MSSelection Memo.

Note that this function can be called multiple times but the result is cumulative. I.e. selection will work on the data already selected from all previous calls of this function. Use the function reset() to reset all selections to NULL (original database).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Record with fields contain the selection expressions. Keys recognized in the record are: &quot;spw&quot;, &quot;time&quot;, &quot;field&quot;, &quot;baseline&quot;, &quot;scan&quot;, &quot;scanintent&quot;, &quot;polarization&quot;, &quot;observation&quot;, &quot;array&quot;, &quot;uvdist&quot; and &quot;taql&quot;.</th>
</tr>
</thead>
<tbody>
<tr>
<td>items</td>
<td>allowed: record                                                                                                                                   Default:</td>
</tr>
<tr>
<td>onlyparse</td>
<td>If set to True, expressions will only be parsed but not applied to the MS for selection. When set to False, a selected MS will also be generated internally. Default is False. When only parsing is requested, the selected-MS is the same as the original MS.</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

735
CASA: staql={'field':'3C286', 'spw':'0~7:10~55'};
CASA: ms.open(MSNAME);
CASA: ms.msselect(staql, onlyparse=True); # For only getting the list
     # of indices corresponding to the selection
CASA: ndx=ms.msselectedindices();
CASA: ndx['field']
     Out[5]: array([1], dtype=int32)
     :
     :
CASA: ms.msselect(staql); # To do the actual selection.
     # From this point on, the ms-tool is
     # attached to the selected MS.
1.3.1 Return the selected indices of the MS database. The keys in the record are the same as those used in msselect function (i.e. 'spw', 'time', 'field', 'baseline', 'scan', 'scanintent', 'polarization' and 'uvdist').

Description

The return indices are the result of parsing the MSSelection expressions provided in the msselect function.

Arguments

| Inputs
| record

Example
ms.select.html

**ms.select - Function**

Select a subset of the measurement set.

**Description**

This function will select a subset of the current measurement set based on the range of values for each field in the input record. The range function will return a record that can be altered and used as the argument for this function. A successful selection returns True. Allowable fields are tabulated in table~\[1.6\] (in the general description of this tool). Unrecognized fields are ignored. You need to call selectinit before calling this function. If you haven’t then selectinit will be called for you with default arguments. Repeated use of this function, with different arguments, will further refine the selection, resulting in a successively smaller selected measurement set. If the selected measurement set does not contain any rows then this function will return False and send a warning message in the logger. Otherwise this function will return True. To undo all the selections you need to use the selectinit function (with reset=True).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>items</td>
<td>record with fields contain ranges and enumerations allowed: record</td>
</tr>
<tr>
<td>Default:</td>
<td>record</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
ms.open("3C273XC1.MS")
ms.selectinit(datadescid=0)
ms.select(\{'antenna1':[1,3,5],\'uvdist':[1200.,1900.]\})
```
ms.select({'time':[4121629420.,4121638290.]})
start = qa.getvalue(qa.convert(qa.quantity('1989/06/27/01:03:40'),'s'))
stop = qa.getvalue(qa.convert(qa.quantity('1989/06/27/03:31:30'),'s'))
rec = {}
rec['time'] = [start, stop]
ms.select(items=rec)

This example selects all the data from the measurement set where the value in the DATA_DESC_ID column is zero. This corresponds to a particular spectral window and polarization setup. It then selects all the data where the first antenna in the interferometer is number one, three or five and where the uv distance is between 1200 and 1900 meters. Finally it selects all the data which was observed between 4121629420 seconds and 4121638290 seconds (since zero hours on the day where the modified Julian day is zero). Since this time in seconds is quite obscure I have also illustrated how to use the quanta tool to convert a date/time string into seconds which can then be used to perform the same time selection.

The selections are cumulative so that at the end of this example only data in the specified time range, with the specified, interferometers, uv distances, spectral window and polarization setup are selected.
ms.selecttaql.html

**ms.selecttaql - Function**

[1.3.1] Select a subset of the measurement set.

**Description**

This function will select a subset of the current measurement set based on the standard TaQL selection string given.

Repeated use of this function, with different arguments, will further refine the selection, resulting in a successively smaller selected measurement set. If the selected measurement set does not contain any rows then this function will return False and send a warning message in the logger. Otherwise this function will return True. To undo all the selections you need to use the selectinit function (with reset=True). Note that index values used in the TaQL string are zero-based as are all tool indices.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>msselect</strong></td>
<td>TaQL selection string</td>
</tr>
<tr>
<td><strong>allowed</strong></td>
<td>string</td>
</tr>
</tbody>
</table>

*Default:*

**Returns**

bool

**Example**

```python
ms.open("3C273XC1.MS")
ms.selectinit(datadescid=0)
ms.select({'antenna1':[0,2,4],'uvdist':[1200.,1900.]})
ms.selecttaql('ANTENNA1==2')
ms.range(['ANTENNA1','ANTENNA2'])
# {'antenna1': array([2]),
```
This example selects all the data from the measurement set where the value in the DATA_DESC_ID column is zero. This corresponds to a particular spectral window and polarization setup. It then selects all the data where the first antenna in the interferometer is number zero, two or four and where the uv distance is between 1200 and 1900 meters. Finally it uses a query to select all the data for which the ANTENNA1 column is 2 (this selects the middle antenna of the previous, zero-based, selection). The selections are cumulative so that at the end of this example only data in the specified time range, with the specified, interferometers, uv distances, spectral window and polarization setup are selected.
ms.selectchannel.html

**ms.selectchannel - Function**

1.3.1 Select and average frequency channels

**Description**

This function allows you to select a subset of the frequency channels in the current measurement set. This function can also average, over frequency channels, prior to providing the values to the user.

Selection on channels is not allowed using either the select or command functions as they can only select entire rows in a measurement set. Channel selection involves accessing only some of the values in a row. Like all the selection functions this function does not change the current measurement but updates the measurement set selection parameters so that functions like getdata will return the desired subset of the data. Repeated use of this function will overwrite any previous channel selection.

There are four parameters, the number of output channels, the first input channel to use, the number of input channels to average into one output channel, and the increment in the input spectrum for the next output channel. All four parameters need to be specified.

This function return True if the selection was successful, and False if not. In the latter case an error message will also be sent to the logger.

You need to call selectinit before calling this function. If you haven’t then selectinit will be called for you with default arguments.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Number of output channels, positive integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>nchan</td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 1</td>
</tr>
<tr>
<td>start</td>
<td>First input channel to use, positive integer</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
<tr>
<td>width</td>
<td>Number of input channels to average together, positive integer</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 1</td>
</tr>
<tr>
<td>inc</td>
<td>Increment to next (group of) input channel(s), positive integer</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 1</td>
</tr>
</tbody>
</table>

**Returns**
bool

**Example**

```plaintext
ms.fromfits("NGC5921.MS",
            "/usr/lib/casapy/data/demo/NGC5921.fits")
ms.selectinit(datadescid=0)
ms.selectchannel(3,2,5,3)
```

This example selects all the data from the measurement set where the value in the DATA_DESC_ID column is zero. This corresponds to a particular spectral window and polarization setup. It then selects on frequency channels to produce 3 output channels, the first output channel is the average of channels 2,3,4,5,6 in the input, the second output channel is the average of channel 5,6,7,8,9 and the third is the average of channels 8,9,10,11,12.
ms.selectpolarization.html

**ms.selectpolarization - Function**

1.3.1 Selection and conversion of polarizations

**Description**

This function allows you to select a subset of the polarizations in the current measurement set. This function can also setup conversion to different polarization representations.

You specify the polarizations using a string vector. Allowable strings are include I, Q, U, V, RR, RL, LR, LL, XX, YY, XY, YX. These string must be specified in upper case. If the polarizations match those present in the measurement set they will be selected directly, otherwise all polarizations are read and then a conversion step is done. If the conversion cannot be done then an error will be produced when you try to access the data.

This function return True if the selection was successful, and False if not. You need to call selectinit before calling this function. If you haven’t then selectinit will be called for you with default arguments.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wantedpol</td>
<td>The polarizations wanted</td>
</tr>
<tr>
<td>allowed: stringArray</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
ms.open("3C273XC1.MS")
ms.selectinit(datadescid=0)
ms.selectpolarization(["I","V"])
```
rec = ms.getdata("data")

This example selects all the data from the measurement set where the value in the DATA_DESC_ID column is zero. This corresponds to a particular spectral window and polarization setup. It then selects the I and V polarizations and when the getdata function is called the conversion from RR, LL, LR, RL polarizations to I and V occurs.
ms.regridspw - Function

1.3.1 transform spectral data to different reference frame and/or regrid the frequency channels

Description

This function permits you to transform the spectral data of your measurement set to a given reference frame. The present reference frame information in the MS is examined and the transformation performed accordingly. Since all such transformations are linear in frequency, a pure change of reference frame only affects the channel boundary definitions.

In addition, the function permits you to permanently regrid the data, i.e. reduce the channel number and/or move the boundaries using several interpolation methods (selected using parameter ”interpolation”). The new channels are equidistant in frequency (if parameter ”mode” is chosen to be vrad or freq, or equidistant in wavelength if parameter ”mode” is chosen to be vopt or wave). If ”mode” is chosen to be ”chan”, the regridding is performed by combining the existing channels, i.e. not moving but just eliminating channel boundaries where necessary.

The regridding is applied to the channel definition and all data of the MS, i.e. all columns which contain arrays whose dimensions depend on the number of channels. The input parameters are verified before any modification is made to the MS.

The target reference frame can be set by providing the name of a standard reference frame (LSRK, LSRD, BARY, GALACTO, LGROUP, CMB, TOPO, GEO, or SOURCE, default = no change of frame) in parameter ”outframe”. For each field in the MS, the channel frequencies are transformed from their present reference frame to the one given by parameter ”outframe”.

If the regridding parameters are set, they are interpreted in the ”outframe” reference frame. The regridding is applied to the data after the reference frame transformation.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outframe</td>
<td>Name of the reference frame to transform to (LSRK, LSRD, BARY, GALACTO, LGROUP, CMB, GEO, TOPO, or SOURCE). SOURCE is meant for solar system work and corresponds to GEO + a radial velocity correction (only available for ephemeris objects). If no reference frame is given, the present reference frame given by the data is used, i.e. the reference frame is not changed. The observatory position is taken as the average of all antenna positions. allowed: string Default: LSRK</td>
</tr>
<tr>
<td>mode</td>
<td>The quantity (radio velocity (m/s), optical velocity (m/s), frequency (Hz), wavelength (m), or original channels) in which the user would like to give the regridding parameters below (&quot;center&quot;, &quot;chanwidth&quot;, &quot;bandwidth&quot;): vrad, vopt, freq, wave, or chan. allowed: string Default: chan</td>
</tr>
<tr>
<td>restfreq</td>
<td>Required in case the value of mode is &quot;vrad&quot; or &quot;vopt&quot;: Rest frequency (Hz) for the conversion of the regridding parameters &quot;center&quot;, &quot;chanwidth&quot;, and &quot;bandwidth&quot; to frequencies. allowed: double Default: -3E30</td>
</tr>
<tr>
<td>interpolation</td>
<td>Name of the interpolation method (NEAREST, LINEAR, SPLINE, CUBIC, FFTSHIFT) used in the regridding. Flagging information is combined using &quot;inclusive or&quot;. allowed: string Default: LINEAR</td>
</tr>
<tr>
<td>start</td>
<td>Desired lower edge of the spectral window after regridding in the units given by &quot;mode&quot; and in the reference frame given by &quot;outframe&quot;. If no value is given, it is determined from &quot;center&quot; and &quot;bandwidth&quot;. allowed: double Default: -3E30</td>
</tr>
<tr>
<td>center</td>
<td>(Alternative to setting the parameter &quot;start&quot;). Desired center of the spectral window after regridding in the units given by &quot;mode&quot; and in the reference frame given by &quot;outframe&quot;. If no value is given, the center is determined from &quot;start&quot; and &quot;bandwidth&quot; or, if &quot;start&quot; is not given either, it is kept as it is. allowed: double Default: -3E30</td>
</tr>
<tr>
<td>bandwidth</td>
<td>Desired width of the entire spectral window after regridding in the units given by &quot;mode&quot; and in the reference frame given by &quot;outframe&quot;. If no value is given or the given width is larger than the bandwidth of the data, the width will be truncated to the maximum width possible symmetrically around the value given by &quot;center&quot;. allowed: double Default: -1</td>
</tr>
<tr>
<td>chanwidth</td>
<td>Desired width of the channels in the units given by &quot;mode&quot; and in the reference frame given by &quot;outframe&quot;. This implies that channels will be equidistant in the unit</td>
</tr>
</tbody>
</table>
Returns
bool

Example

ms.fromfits("NGC5921.MS", "/usr/lib/casapy/data/demo/NGC5921.fits")
ms.regridspw(outframe="LSRK")

This example reads a measurement set and transforms its spectral axis to the LSRK reference frame.

ms.regridspw(outframe="BARY", mode="vrad",
center=73961800., chanwidth=50., bandwidth=1000.,
restfreq=1420405750e6)

In this example, all spectral windows in the MS will be transformed to the BARY reference frame and then be regridded such that the center of the new spectral window is at radio velocity = 73961800. m/s (BARY). If the bandwidth of the observation is large enough the total width of the spectral window will be 1000 m/s, i.e. 20 channels of width 50 m/s, 10 on each side of the given center.

ms.regridspw(mode="vopt", restfreq=1420405750e6)

In this example the channels are regridded such that they are equidistant in optical velocity. The reference frame and number of channels is kept as is.

ms.regridspw(mode="chan", center=64, chanwidth=2,
bandwidth=102)

In this example, the channels are regridded such that the new bandwidth is 102 of the original channels centered on the original channel 64, and the new channels are twice as wide as the original channels.
ms.cvel - Function

1.3.1 transform spectral data to different reference frame and/or regrid the frequency channels

Description

This function permits you to transform the spectral data of your measurement set to a given reference frame and/or regrid it. It will combine all spectral windows of the MS into one.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>&quot;channel&quot;, &quot;velocity&quot;, &quot;frequency&quot;, or &quot;channel_b&quot;, default = &quot;channel&quot;</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>channel</td>
</tr>
<tr>
<td>nchan</td>
<td>number of channels, default = -1 = all</td>
</tr>
<tr>
<td>allowed</td>
<td>int</td>
</tr>
<tr>
<td>Default</td>
<td>-1</td>
</tr>
<tr>
<td>start</td>
<td>start channel, default = 0</td>
</tr>
<tr>
<td>allowed</td>
<td>any</td>
</tr>
<tr>
<td>Default</td>
<td>variant 0</td>
</tr>
<tr>
<td>width</td>
<td>new channel width, default = 1</td>
</tr>
<tr>
<td>allowed</td>
<td>any</td>
</tr>
<tr>
<td>Default</td>
<td>variant 1</td>
</tr>
<tr>
<td>interp</td>
<td>interpolation method &quot;nearest&quot;, &quot;linear&quot;, &quot;spline&quot;, &quot;cubic&quot;, &quot;fftshift&quot;, default = linear</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>linear</td>
</tr>
<tr>
<td>phsec</td>
<td>phase center, default = first field</td>
</tr>
<tr>
<td>allowed</td>
<td>any</td>
</tr>
<tr>
<td>Default</td>
<td>variant</td>
</tr>
<tr>
<td>restfreq</td>
<td>rest frequency, default = 1.4GHz</td>
</tr>
<tr>
<td>allowed</td>
<td>any</td>
</tr>
<tr>
<td>Default</td>
<td>variant 1.4GHz</td>
</tr>
<tr>
<td>outframe</td>
<td>LSRK, LSRD, BARY, GALACTO, LGROUP, CMB, GEO, TOPO, or SOURCE default = &quot;&quot; = keep reference frame.</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>veltype</td>
<td>radio or optical, default = radio</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>radio</td>
</tr>
<tr>
<td>hanning</td>
<td>If true, perform hanning smoothing before regridding.</td>
</tr>
<tr>
<td>allowed</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>true</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

750
ms.cvelfreqs.html

**ms.cvelfreqs - Function**

1.3.1 calculate the transformed grid of the SPW obtained by combining a given set of SPWs (MS is not modified)

**Description**

Take the spectral grid of a given spectral window, transform and regrid it as prescribed by the given grid parameters (same as in cvel and clean) and return the transformed values as a list. The MS is not modified. Useful for tests of gridding parameters before using them in cvel or clean.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>spwids</td>
<td>The list of ids of the spectral windows from which the input grid is to be taken.</td>
<td>intArray</td>
<td>0</td>
</tr>
<tr>
<td>fieldids</td>
<td>The list of ids of the fields which are selected (for observation time determination), default: all</td>
<td>intArray</td>
<td>0</td>
</tr>
<tr>
<td>obstime</td>
<td>The observation time to assume, default: time of the first row of the MS =</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>mode</td>
<td>&quot;channel&quot;, &quot;velocity&quot;, &quot;frequency&quot;, or &quot;channel_b&quot;, default =</td>
<td>string</td>
<td>channel</td>
</tr>
<tr>
<td>nchan</td>
<td>number of channels, default = all =</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>start</td>
<td>start channel, default =</td>
<td>any</td>
<td>variant 0</td>
</tr>
<tr>
<td>width</td>
<td>new channel width, default =</td>
<td>any</td>
<td>variant 1</td>
</tr>
<tr>
<td>phsec</td>
<td>phase center, default = first field in selection =</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>restfreq</td>
<td>rest frequency, default =</td>
<td>any</td>
<td>variant 1.4GHz</td>
</tr>
<tr>
<td>outframe</td>
<td>LSRK, LSRD, BARY, GALACTO, LGROUP, CMB, GEO, TOPO, or SOURCE default = keep reference frame =</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>veltype</td>
<td>radio or optical, default =</td>
<td>string</td>
<td>radio</td>
</tr>
<tr>
<td>verbose</td>
<td>If true, create log output</td>
<td>bool</td>
<td>true</td>
</tr>
</tbody>
</table>
Returns
doubleArray

Example

ms.open('my.ms')
ms.cvelfreqs(spwids=[1], mode='channel', nchan=20, start=2, width=3, outframe='LSRK')

will take the grid of SPW 1 (i.e. the second in the SPW table), regrid it as in cvel with the given grid parameters and return the resulting channel centers as an array. The MS is not modified. See help cvel for more details on the grid parameters.
ms.getdata.html

ms.getdata - Function

1.3.1. Read values from the measurement set.

Description

This function will read the specified items from the currently selected measurement set and returns them in fields of a record. The main difference between this and direct access of the table, using the table tool, is that this function reads data from the selected measurement set, it provides access to derived quantities like amplitude & flag_sum and it can reorder the data. The items to read are specified, as with the range function, using a vector of strings. Table-1.6 shows the allowable items. Unrecognized items will result in a warning being sent to the logger. Duplicate items are silently ignored. The record that is returned contains fields that correspond to each of the specified items. Most fields will contain an array. The array may be one, two or three dimensional depending on whether the corresponding row in the measurement set is a scalar, one or two dimensional. Unless the ifraxis argument is set to T the length of the last axis on these arrays will correspond to the number of rows in the selected measurement set. If the ifraxis argument is set to True, the row axis is split into an interferometer axis and a time axis. For example a measurement set with 90 rows, in an array with 6 telescopes (so that there are 15 interferometers), may have a data array of shape \([4,32,90]\) if ifraxis is False or \([4,32,15,6]\), if ifraxis is True (assuming there are 4 correlations and 32 channels). If there are missing rows as will happen if not all interferometers where used for all time-slots then a default value will be inserted. This splitting of the row axis may not happen for items where there is only a single value per row. For some items the returned vector will contain only as many values as they are interferometers and it is implicit that the same value should be used for all time slots. The antenna1, antenna2, feed1, feed2 & ifr_number items fall in this category. For other items the returned vector will have as many values as there are time slots and it is implicit that the same value should be used for all interferometers. The field_id, scan_number, data_desc_id & time items fall into this category. The axis_info item provides data labelling information. It returns a record with the following fields: corr_axis, freq_axis, ifr_axis & time_axis. The latter two fields are not present if ifr_axis is set to False. The corr_axis field contains a string vector with elements like 'RR' or 'XY' that indicates which polarizations where correlated together to produce the data. The length of this vector will always be the same as the length of the first axis of the data.
array. The freq_axis field contains a record with two fields, chan_freq & resolution. Each of these fields contains vectors which indicate the centre frequency and spectral resolution (FWHM) of each channel. The length of these vectors will be the same as the length of the second axis in the data. The ifr_axis field contains fields: ifr_number, ifr_name, ifr_shortname & baseline. The ifr_number is the same as returned by the ifr_item, the ifr_name & ifr_shortname are string vectors containing descriptions of the interferometer and the baseline is the Euclidian distance, in meters between the two antennas. All of these vectors have a length equal to the number of interferometers in the selected measurement set i.e., to the length of the third axis in the data when ifraxis is True. The time_axis field contains the MJD seconds field and optionally the HA, UT & LAST fields. To include the optional fields you need to add the ha, last or ut strings to the list of requested items. All the fields in the time_axis record contain vectors that indicate the time at the midpoint of the observation and are in seconds. The MJD seconds field is since 0 hours on the day having a modified julian day number of zero and the rest are since midnight prior to the start of the observation. An optional gap size can be specified to visually separate groups of interferometers with the same antenna1 index (handy for identifying antennas in an interferometer vs time display). The default is no gap. An optional increment can be specified to return data from every row matching the increment only. When the average flag is set, the data will be averaged over the time axis if the ifraxis is True or the row axis i.e., different interferometers and times may be averaged together. In the latter case, some of the coordinate information, like antenna_id, will no longer make sense. You need to call selectinit before calling this function. If you haven’t then selectinit will be called for you with default arguments. Items prefixed with either; corrected, model, residual or obs_residual and the imaging_weight item are not available unless your measurement set has been processed either with the imager or calibrator tools.

Arguments
### Inputs

<table>
<thead>
<tr>
<th>items</th>
<th>Item names</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

| ifraxis | Create interferometer axis if True |
| allowed | bool |
| Default: | false |

| ifraxisgap | Gap size on ifr axis when antenna1 changes |
| allowed | int |
| Default: | 0 |

| increment | Row increment for data access |
| allowed | int |
| Default: | 1 |

| average | Average the data in time or over rows |
| allowed | bool |
| Default: | false |

### Returns

record

### Example

```python
ms.open("3C273XC1.MS")
ms.selectinit(datadescid=0)
# The following line causes an EXCEPTION
# d = ms.getdata(['amplitude','axis_info','ha'],ifraxis=True)
# so we settle for MJDseconds rather than HA in seconds
d = ms.getdata(['amplitude','axis_info'],ifraxis=True)
tstart = min(d['axis_info']['time_axis']['MJDseconds'])
tstop = max(d['axis_info']['time_axis']['MJDseconds'])
maxamp = max(max(max(d['amplitude'][:,:,0,0]),
max(d['amplitude'][0,:,0,0]),
max(d['amplitude'][0,0,:,0]),
max(d['amplitude'][0,0,0,:])
)
print "MJD start time (seconds) =", tstart
# MJD start time (seconds) = 4121629400.0
print "MJD stop time (seconds) =", tstop
# MJD stop time (seconds) = 4121642670.0
# MJDseconds Correlation amplitude
print "Maximum correlation amplitude =", maxamp
# Maximum correlation amplitude = 33.5794372559
chan = 0
```
corr = 0
freqGHz = d["axis_info"]['freq_axis']["chan_freq"][chan]/1.0E9
baselineStr = d["axis_info"]['ifr_axis']["ifr_name"][corr]
corrStr = d["axis_info"]['corr_axis'][corr]
tcoord = d["axis_info"]['time_axis']["MJDseconds"]
acoord = d['amplitude'][0,0,0,:]
print "Frequency", freqGHz, "GHz", "Baseline", baselineStr, "(" + corrStr + ")"
print "MJDseconds", "Correlation amplitude"
for i in range(len(tcoord)):
    print tcoord[i], acoord[i]

# Frequency [ 8.085] GHz Baseline 1-2 ( RR )
# MJDseconds Correlation amplitude
# 4121629400.0 29.2170944214
# 4121629410.0 29.1688995361
# 4121629420.0 29.2497825623
# 4121629430.0 29.2029647827
# 4121629440.0 29.166015625
# 4121629450.0 29.2417526245
# 4121629460.0 29.2867794037
# 4121638270.0 0.0
# 4121638280.0 29.4539775848
# 4121638290.0 29.472661972
# 4121638300.0 29.4424362183
# 4121638310.0 29.4234466553
# 4121638320.0 29.4018748422
# 4121638330.0 29.3326053619
# 4121638340.0 29.3575496674
# 4121642600.0 31.141132812
# 4121642610.0 31.0726108551
# 4121642620.0 31.1242599487
# 4121642630.0 31.0505466461
# 4121642640.0 31.0448284149
# 4121642650.0 30.9974422455
# 4121642660.0 30.648326874
# 4121642670.0 30.638961792

This example selects all the data from the measurement set where the value in the DATA_DESC_ID column is zero. This corresponds to a particular spectral window and polarization setup. It then gets the correlated amplitude, and the axis information from this selected measurement set. This is returned in the casapy variable d. The remainder of the example prints a table of 'hour angle' and corresponding 'correlated amplitude' for the first channel, correlation and baseline.
ms.putdata.html

**ms.putdata - Function**

1.3.1 Write new values into the measurement set

**Description**

This function allows you to write values from casapy variables back into the measurement set table. The main difference between this and directly accessing the table using the table tool is that this function writes data to the *selected* measurement set.

Unlike the getdata function you can only put items that correspond to actual table columns. You cannot change the data shape either so that the number of correlations, channels and rows (or interferometers/time slots) must match the values in the selected measurement set. If the values were obtained using the getdata function with ifraxis argument set to True, then any default values added to fill in missing interferometer/timeslots pairs will be ignored when writing the modified values back using this function.

The measurement set has to be opened for read/write access to be able to use this function.

You need to call selectinit before calling this function. If you haven’t then selectinit will be called for you with default arguments.

Items prefixed with either; corrected, model, residual or obs_residual and the imaging_weight item are not available unless your measurement set has been processed either with the imager or calibrator tools.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>items</td>
<td>Record with items and their new values</td>
</tr>
<tr>
<td>allowed:</td>
<td>record</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**
ms.open("3C273XC1.MS", nomodify=False)
ms.selectinit(datadescid=0)
rec = ms.getdata(["weight","data"])
rec[‘weight’][::] = 1
import Numeric
meanrec = Numeric.average(rec[‘data’],axis=None)
print "Mean data value = ", meanrec
rec[‘data’][::] -= meanrec
ms.putdata(rec)

This example selects all the data from the measurement set where the value in the DATA_DESC_ID column is zero. This corresponds to a particular spectral window and polarization setup. Note that the measurement set was opened for writing as well as reading. The third line reads all the weights and the data into the casapy variable rec. The weights are set to one. The more obscure syntax is used as typing rec[‘weight’] = 1 will not preserve the shape of the weight array. The data then has its mean subtracted from it. The average function is defined in Numeric module. Finally the data is written back into the measurement set table. (NOTE: normally one should not modify the raw data column. Such adjustments are more appropriate for the corrected_data column, if it exists.)
ms.concatenate.html

**ms.concatenate - Function**

[1.3.1] Concatenate two measurement sets

**Description**

This function concatenates two measurement sets together. The data is copied from the measurement set specified in the msfile argument to the end of the measurement set attached to the ms tool. If a lot of data needs to be copied this function may take some time. You need to open the measurement set for writing in order to use this function.

**Arguments**
<table>
<thead>
<tr>
<th><strong>Inputs</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>msfile</strong></td>
<td>The name of the measurement set to append</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><strong>freqtol</strong></td>
<td>Frequency difference within which 2 spectral windows are considered similar; e.g '10Hz'</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant 1Hz</td>
</tr>
<tr>
<td><strong>dirtol</strong></td>
<td>Direction difference within which 2 fields are considered the same; e.g '1mas'</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant 1mas</td>
</tr>
<tr>
<td><strong>weightscale</strong></td>
<td>Scale the weights of the MS to be appended by this factor</td>
</tr>
<tr>
<td>allowed:</td>
<td>float</td>
</tr>
<tr>
<td>Default:</td>
<td>1.</td>
</tr>
<tr>
<td><strong>handling</strong></td>
<td>Switch for the handling of the Main and Pointing tables: 0=standard, 1=no Main, 2=no Pointing, 3=no Main and Pointing, 4=virtual</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td><strong>destmsfile</strong></td>
<td>Optional support for virtual concat: empty table (no subtables) where to store the appended MS copy</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><strong>respectname</strong></td>
<td>If true, fields with a different name are not merged even if their direction agrees</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
ms.open("3C273XC1.MS", nomodify=False)
ms.concatenate("BLLAC.ms", '1GHz', '1arcsec')
ms.done()
```

This example appends the data from the BLLAC measurement set to
the end of the 3C273 measurement set. Its going to assume a frequency tolerance of 1GHz and position tolerance of 1 arcsec in deciding if the spw and field in the measurement sets are similar or not.
ms.testconcatenate.html

**ms.testconcatenate - Function**

[1.3.1] Concatenate only the subtables of two measurement sets excluding the POINTING table (resulting MAIN and POINTING table not useful)

**Description**

This function acts like `ms.concatenate()` with handling==3 (do not concatenate the MAIN and POINTING tables). This is useful for generating, e.g., SPECTRAL_WINDOW and FIELD tables which contain all used SPW and FIELD ids for a set of MSs without having to actually carry out a time-consuming concatenation on disk. The MAIN table in the resulting output MS is that of the original MS, i.e. it is not touched.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msfile</td>
<td>The name of the measurement set from which the sub-tables should be appended</td>
</tr>
<tr>
<td>freqtol</td>
<td>Frequency difference within which 2 spectral windows are considered similar; e.g '10Hz'</td>
</tr>
<tr>
<td>dirtol</td>
<td>Direction difference within which 2 fields are considered the same; e.g '1mas'</td>
</tr>
<tr>
<td>respectname</td>
<td>If true, fields with a different name are not merged even if their direction agrees</td>
</tr>
</tbody>
</table>

| Returns | bool |

**Example**

765
tb.open("3C273XC1.MS")
tb.copy("TEMP.MS", norows=True)
tb.close()
ms.open("TEMP.MS", nomodify=False)
ms.testconcatenate("3C273XC1.ms", '1GHz', '1arcsec')
ms.testconcatenate("BLLAC.ms", '1GHz', '1arcsec')
ms.done()

This example makes a copy of the structure of an MS and then appends the subtables data from two measurement sets to the empty structure. It's going to assume a frequency tolerance of 1GHz and position tolerance of 1 arcsec in deciding if the spw and field in the measurement sets are similar or not.
ms.virtconcatenate.html

**ms.virtconcatenate - Function**

1.3.1 Concatenate two measurement sets virtually

**Description**

This function virtually concatenates two measurement sets together such that they can later be turned into a multi-MS with createmultims(). You need to open the measurement set for writing in order to use this function.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>msfile</td>
<td>The name of the measurement set to append</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>auxfilename</td>
<td>The name of an auxiliary file which is needed when more than two MSs are to be concatenated.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>freqtol</td>
<td>Frequency difference within which 2 spectral windows are considered similar; e.g '10Hz'</td>
<td>any</td>
<td>variant 1Hz</td>
</tr>
<tr>
<td>dirtol</td>
<td>Direction difference within which 2 fields are considered the same; e.g '1mas'</td>
<td>any</td>
<td>variant 1mas</td>
</tr>
<tr>
<td>weightscale</td>
<td>Scale the weights of the MS to be appended by this factor</td>
<td>float</td>
<td>1.0</td>
</tr>
<tr>
<td>respectname</td>
<td>If true, fields with a different name are not merged even if their direction agrees</td>
<td>bool</td>
<td>true</td>
</tr>
</tbody>
</table>

**Returns**

bool
Example

```python
ms.open("3C273XC1.ms", nomodify=False)
ms.virtconcatenate("3C273XC1-2.ms", '3Caux.dat', '1GHz', '1arcsec')
ms.virtconcatenate("3C273XC1-3.ms", '3Caux.dat', '1GHz', '1arcsec')
ms.close()
os.remove('3Caux.dat')
m.createmultims(concatvis,       "3C273XC1.ms","3C273XC1-2.ms","3C273XC1-3.ms"],
[],
True, # nomodify
False,# lock
True) # copysubtables from first to all other members
ms.close()
```

This example virtually appends the data from the 3C273XC1-2 and 3C273XC1-3 to the end of the 3C273XC1 measurement set. Its going to assume a frequency tolerance of 1GHz and position tolerance of 1 arcsec in deciding if the spw and field in the measurementsets are similar or not. The file 3Caux.dat which is created in the process is no longer needed after the last call to virtconcatenate() and can be deleted.
ms.timesort.html

ms.timesort - Function

Description

This function sorts the main table of the measurement set by the contents of the column TIME in ascending order and writes a copy of the MS with the sorted main table into newmsfile.
If no newmsname is given, a sorted copy of the MS is written into a new MS under the name x.sorted (where x is the name of the original MS). The original MS is then closed and deleted. The new MS is renamed to the name of the original MS and then reopened.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>newmsname</td>
<td>Name of the output measurement set (default: overwrite original)</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool

Example

```python
ms.open("3C273XC1.MS", nomodify=False)
ms.timesort()
ms.done()
```

This example sorts the main table of 3C273XC1.MS by time. The original MS is overwritten by the sorted one.
ms.sort.html

**ms.sort - Function**

1.3.1 Sort the main table of an MS using a custom set of columns

**Description**

This function sorts the main table of the measurement set by the contents of the input set of columns in ascending order and writes a copy of the MS with the sorted main table into newmsfile.

If no newmsname is given, a sorted copy of the MS is written into a new MS under the name x.sorted (where x is the name of the original MS). The original MS is then closed and deleted. The new MS is renamed to the name of the original MS and then reopened.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>newmsname</td>
<td>Name of the output measurement set (default: overwrite original)</td>
</tr>
<tr>
<td>columns</td>
<td>Vector of column names (case sensitive).</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
ms.open("3C273XC1.MS", nomodify=False)
ms.sort(["ANTENNA1","ANTENNA2"])
ms.done()
```
This example sorts the main table of 3C273XC1.MS by ANTENNA1 and then ANTENNA2. The original MS is overwritten by the sorted one.
ms.contsub - Function

1.3.1 Subtract the continuum from the visibilities

Description

NOT FULLY IMPLEMENTED YET. uvcontsub uses the cb tool for now. (The only reason to implement it in ms is to save time and disk space.) This function estimates the continuum emission of the MS and writes a MS with that estimate subtracted, using the ms tool. The estimate is made, separately for the real and imaginary parts of each baseline, by fitting a low order polynomial to the unflagged visibilities selected by fitspw (depending on combine).

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputms</td>
<td>The name of the resulting measurement set</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>fitspw</td>
<td>Line-free spectral windows (and :channels) to fit to</td>
<td>variant</td>
<td>*</td>
</tr>
<tr>
<td>fitorder</td>
<td>The order of the polynomial to use when fitting.</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>combine</td>
<td>Ignore changes in these columns (spw, scan, and/or state) when fitting.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>spw</td>
<td>Spectral windows (and :channels) to select</td>
<td>variant</td>
<td>*</td>
</tr>
<tr>
<td>unionspw</td>
<td>The union of fitspw and spw, i.e. how much needs to be read. '*' always works, but may be more than you need.</td>
<td>variant</td>
<td>*</td>
</tr>
<tr>
<td>field</td>
<td>Fields to include, by names or 0-based ids. ('' =&gt; all)</td>
<td>variant</td>
<td></td>
</tr>
<tr>
<td>scan</td>
<td>Only use the scan numbers requested using the mselection syntax.</td>
<td>variant</td>
<td></td>
</tr>
<tr>
<td>intent</td>
<td>Only use the requested scan intents.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>correlation</td>
<td>Limit data to specific correlations (LL, XX, LR, XY, etc.).</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>obs</td>
<td>Only use the requested observation IDs.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>whichcol</td>
<td>'DATA', 'MODEL_DATA', 'CORRECTED_DATA', and/or 'FLOAT_DATA'</td>
<td>string</td>
<td>CORRECTED_DATA</td>
</tr>
</tbody>
</table>

| Returns        | bool                                                                        |           |         |
Example

ms.open("multiwin.ms")
ms.contsub('contsub.ms', fitspw='0:0~123;145~211,2:124~255', fitorder=0,
          field=[0], spw='0,2')

In this example the continuum estimates are made by separately averaging channels 0:0~123;145~211 and 2:124~255, and the separate estimates are subtracted from spws 0 and 2. The output only includes field 0 and spws 0 and 2 (now called 1).

ms.contsub('contsub.ms', fitspw='0:0~123;145~211,2:124~255', fitorder=0,
           field=[0], combine='spw')
ms.close()

This time the estimate was made by simultaneously averaging channels 0:0~123;145~211 and 2:124~255, and the continuum is subtracted from all the spws, including 1 (treated as a completely line-filled spw). The output only includes field 0.
ms.statwt.html

**ms.statwt - Function**

[1.3.1] Set WEIGHT and SIGMA from the scatter of the visibilities

**Description**

NOT IMPLEMENTED YET.

This function estimates the noise from the scatter of the visibilities, sets SIGMA to it, and WEIGHT to SIGMA**-2.

Ideally the visibilities used to estimate the scatter, as selected by fitspw and fitcorr, should be pure noise. If you know for certain that they are, then setting dorms to True will give the best result. Otherwise, use False (standard sample standard deviation). More robust scatter estimates like the interquartile range or median absolute deviation from the median are not offered because they require sorting by value, which is not possible for complex numbers.

To beat down the noise of the noise estimate, the sample size per estimate can be made larger than a single spw and baseline. (Using combine='spw' is to interpolate between spws with line-free channels is recommended when an spw has no line-free channels.) timebin smooths the noise estimate over time.

windowtype sets the type of time smoothing.

WEIGHT and SIGMA will not be changed for samples that have fewer than minsamp visibilities. Selected visibilities for which no noise estimate is made will be flagged. Note that minsamp is effectively at least 2 if dorms is False, and 1 if it is True.

**Arguments**
Inputs

dorms  How the scatter should be estimated (True -> rms, False -> stddev)
  allowed:  bool
  Default:  false

byantenna  How the scatters are solved for (by antenna or by baseline)
  allowed:  bool
  Default:  true

sepacs  If solving by antenna, treat autocorrs separately
  allowed:  bool
  Default:  true

fitspw  Line-free spectral windows (and :channels) to get the scatter from. (" => all)
  allowed:  variant
  Default:  *

fitcorr  Correlations (V, LL, XX, LR, XY, etc.) to get the scatter from. (" => all)
  allowed:  variant
  Default:  *

combine  Ignore changes in these columns (spw, scan, and/or state) when getting the scatter.
  allowed:  string
  Default:  *

timebin  Duration of the moving window over which to estimate the scatter. Defaults to 0s, with an effective minimum of 1 integration.
  allowed:  variant
  Default:  0s

minsamp  The minimum number of visibilities for a scatter estimate
  allowed:  int
  Default:  3

field  Fields to reweight, by names or 0-based ids. (" => all)
  allowed:  variant
  Default:  *

spw  Spectral windows to reweight. (" => all)
  allowed:  variant
  Default:  *

antenna  Select data based on antenna/baseline
  allowed:  any
  Default:  *

timerange  Select data by time range
  allowed:  string
  Default:  *

scan  Scan numbers to reweight. (" => all)
  allowed:  variant
  Default:  *

intent  Scan intents to reweight. (" => all)
  allowed:  string
  Default:  *

array  Select (sub)array(s) by array ID number
  allowed:  variant
  Default:  *

correlation  Correlations (LL, XX, LR, XY, etc.) to reweight. (" => all)
  allowed:  string
  Default:  *
Returns
bool

Example

```python
ms.open("multiwin.ms", nomodify=False)
ms.statwt(fitspw='0:0~123;145~211,2:124~255', field=[0], spw='0,2')

In this example the noise estimates are separately made from and applied to spws 0 and 2.

ms.statwt(fitspw='0:0~123;145~211,2:124~255', fitorder=0, field=[0],
          combine='spw')
ms.close()

This time the estimate for each baseline is made from the line-free channels of spws 0 and 2, and applied to all the spws, including 1 (which could be a completely line-filled spw).
```
ms.split.html

**ms.split - Function**

1.3.1 make a new ms from a subset of an existing ms, adjusting subtables and indices

**Description**

This function splits out part of the MS into a new MS. Time and channel averaging can be performed in the process (but not in the same call). When splitting multiple spectral windows, the parameters `nchan`, `start`, `step` can be vectors, so that each spectral window has its own selection on averaging and number of output channels. But the option of using only one value for each of these parameters means that it will be replicated for all the spectral windows selected.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputms</td>
<td>The name of the resulting measurement set</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>field</td>
<td>Fields to include, by names or 0-based ids. (&quot; =&gt; all)</td>
<td>variant</td>
<td></td>
</tr>
<tr>
<td>spw</td>
<td>Spectral windows (and :channels) to select</td>
<td>variant</td>
<td></td>
</tr>
<tr>
<td>step</td>
<td>Number of input per output channels - Int vector of length 1 or same as spw</td>
<td>intArray</td>
<td>1</td>
</tr>
<tr>
<td>baseline</td>
<td>Antenna names or indices to select (&quot; =&gt; all)</td>
<td>variant</td>
<td></td>
</tr>
<tr>
<td>timebin</td>
<td>Duration for averaging. Defaults to no averaging.</td>
<td>variant</td>
<td>-1s</td>
</tr>
<tr>
<td>time</td>
<td>Only use data in the given time range, using the msselection syntax.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>scan</td>
<td>Only use the scan numbers requested using the msselection syntax.</td>
<td>variant</td>
<td></td>
</tr>
<tr>
<td>uvrance</td>
<td>Limit data by uv distance using the msselection syntax.</td>
<td>variant</td>
<td></td>
</tr>
<tr>
<td>taql</td>
<td>For the TAQL experts, flexible data selection using the TAQL syntax.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>whichcol</td>
<td>'DATA', 'MODEL_DATA', 'CORRECTED_DATA', 'FLOAT_DATA', 'LAG_DATA', and/or 'all'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tileshape</td>
<td>Tile shape of the disk data columns, most users should not need to touch this parameter [0] =&gt; normal tiling, [1] =&gt; fast mosaic style tile [4,15,351] =&gt; a tile shape of 4 pol 15 chan and 351 rows</td>
<td>variant</td>
<td></td>
</tr>
<tr>
<td>subarray</td>
<td>Limit data to specific (sub)array numbers.</td>
<td>variant</td>
<td>779</td>
</tr>
<tr>
<td>combine</td>
<td>Ignore changes in these columns (scan, and/or state) when time averaging.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>correlation</td>
<td>Limit data to specific correlations (LL, XX, LR, XY, etc.).</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>
Returns
bool

Example

ms.open("multiwin.ms")
ms.split('subms.ms', field=[0], spw=[0], nchan=[10],
         start=[0], step=[5], whichcol='CORRECTED_DATA')

In this example we split out data from the 1st field and 1st spectral window. The output data will have 10 channels which is taken from 50 channels from the input data starting at channel 0 and averaging every 5.

ms.open("multiwin.ms")
ms.split('subms.ms', field=[0], spw=[0,1,2,3], nchan=[10],
         start=[0], step=[5], whichcol='CORRECTED_DATA')

In this example we split out data from the 1st field and four spectral windows. The output data will have 4 spectral windows each of 10 channels which is taken from 50 channels from the input data starting at channel 0 and averaging every 5.

ms.open("multiwin.ms")
ms.split('subms.ms', field=[0], spw=[0,1,2,3], nchan=[10,10,30,40],
         start=[0,4,9,9], step=[1,10,5,2], whichcol='CORRECTED_DATA')

In this example we split out data from the 1st field and four spectral windows. There will be four spectral windows in the output data, with 10, 10, 30 and 40 channels respectively. These are averages of the input spectral windows. The first output spectral window will be formed by picking 10 channels, starting at 0 with no averaging, of the input spwid 0. The second output spectral window will consists of 10 channels and is formed by picking 100 channels from spwid 1 of the input data, starting at channel 4, and every 10 channels to make one output channel.

ms.open("WSRT.ms")
ms.split('subms.ms', timebin='20s', whichcol='all', combine='scan')
ms.close()

This example averages a WSRT MS into 20s bins, selecting whichever of DATA, MODEL_DATA, CORRECTED_DATA, or FLOAT_DATA, or LAG_DATA is present. Normally the bins would not cross scans, but in this MS the scan number goes up with each integration, making it redundant enough with time that it would defeat any time averaging. Therefore the combine parameter forces the SCAN column to be ignored for setting the bins.
ms.partition.html

**ms.partition - Function**

1.3.1 make a new ms from a subset of an existing ms, without changing any subtables

**Description**

This function splits out part of the MS into a new MS. Time averaging can be performed in the process. Unlike split, the subtables and IDs (ANTENNA1, DATA_DESCRIPTION_ID, etc.) are never changed to account for the selection.

As a side effect of that property, partition cannot select by channel or correlation, or average channels. It CAN select by spectral window(s).

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputms</td>
<td>The name of the resulting measurement set</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>field</td>
<td>Fields to include, by names or 0-based ids. (''' =&gt; all)</td>
<td>variant</td>
<td></td>
</tr>
<tr>
<td>spw</td>
<td>Spectral windows (and :channels) to select</td>
<td>variant</td>
<td>*</td>
</tr>
<tr>
<td>baseline</td>
<td>Antenna names or indices to select (''' =&gt; all)</td>
<td>variant</td>
<td></td>
</tr>
<tr>
<td>timebin</td>
<td>Duration for averaging. Defaults to no averaging.</td>
<td>variant</td>
<td>-1s</td>
</tr>
<tr>
<td>time</td>
<td>Only use data in the given time range, using the msselection syntax.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>scan</td>
<td>Only use the scan numbers requested using the msselection syntax.</td>
<td>variant</td>
<td></td>
</tr>
<tr>
<td>uvrange</td>
<td>Limit data by uv distance using the msselection syntax.</td>
<td>variant</td>
<td></td>
</tr>
<tr>
<td>taql</td>
<td>For the TAQL experts, flexible data selection using the TAQL syntax</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>whichcol</td>
<td>'DATA', 'MODEL_DATA', 'CORRECTED_DATA', 'FLOAT_DATA', 'LAG_DATA', and/or 'all'</td>
<td>string</td>
<td>DATA</td>
</tr>
<tr>
<td>tileshape</td>
<td>Tile shape of the disk data columns, most users should not need to touch this parameter [0] =&gt; normal tiling, [1] =&gt; fast mosaic style tile [4,15,351] =&gt; a tile shape of 4 pol 15 chan and 351 rows</td>
<td>variant</td>
<td></td>
</tr>
<tr>
<td>subarray</td>
<td>Limit data to specific (sub)array numbers.</td>
<td>variant</td>
<td></td>
</tr>
<tr>
<td>combine</td>
<td>Ignore changes in these columns (scan, and/or state) when time averaging.</td>
<td>string</td>
<td>783</td>
</tr>
<tr>
<td>intent</td>
<td>Only use the requested scan intents.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>obs</td>
<td>Only use the requested observation IDs.</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>
Returns
bool

Example

ms.open("multiwin.ms")
ms.partition('partition.ms', field=[0], spw=[1], whichcol='CORRECTED_DATA')

In this example we partition out data from the 1st field and 2nd spectral window. Only the CORRECTED_DATA data column will be copied, and it will be written to the DATA column of partition.ms.

ms.open("multiwin.ms")
ms.partition('partition.ms', field=[0], spw=[0,1,2,3],
            whichcol='CORRECTED_DATA')

In this example we partition out calibrated data from the 1st field and four spectral windows.

ms.open("WSRT.ms")
ms.partition('partition.ms', timebin='20s', whichcol='all', combine='scan')
ms.close()

This example averages a WSRT MS into 20s bins, selecting whichever of DATA, MODEL_DATA, CORRECTED_DATA, or FLOAT_DATA, or LAG_DATA is present. Normally the bins would not cross scans, but in this MS the scan number goes up with each integration, making it redundant enough with time that it would defeat any time averaging. Therefore combine parameter forces the SCAN column to be ignored for setting the bins.
**ms.iterinit** - Function

Initialize for iteration over an ms

**Description**

Specify the columns to iterate over and the time interval to use for the TIME column iteration. The columns are specified by their MS column name. Note that the following columns are always added to the specified columns: array_id, field_id, data_desc_id and time. This is so that the iterator can keep track of the coordinates associated with the data (field direction, frequency etc.) If you want to sort on these columns last instead of first you need to include them in the columns specified. If you don’t want to sort on these columns at all, you can set adddefaultsortcolumns to False. You need to call selectinit before calling this. See the example below.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>columns</td>
<td>Vector of column names (case sensitive).</td>
</tr>
<tr>
<td>allowed: stringArray</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>interval</td>
<td>Time interval in seconds (greater than 0), to group together in iteration</td>
</tr>
<tr>
<td>allowed: double</td>
<td></td>
</tr>
<tr>
<td>Default: 0.0</td>
<td></td>
</tr>
<tr>
<td>maxrows</td>
<td>Max number of rows (greater than 0) to return in iteration</td>
</tr>
<tr>
<td>allowed: int</td>
<td></td>
</tr>
<tr>
<td>Default: 0</td>
<td></td>
</tr>
<tr>
<td>adddefaultsortcolumns</td>
<td>Add the default sort columns</td>
</tr>
<tr>
<td>allowed: bool</td>
<td></td>
</tr>
<tr>
<td>Default: true</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

785
See the example for the iterend function.
ms.iterorigin - Function

Set the iterator to the start of the data.

Description

Set or reset the iterator to the start of the currently specified iteration. You need to call this before attempting to retrieve data with getdata. You can set the iteration back to the start before you reach the end of the data. You need to call iterinit before calling this. See the example below.

Arguments

Returns

bool

Example

See the example for the iterend function.
ms.iternext.html

**ms.iternext - Function**

**1.3.1** Advance the iterator to the next lot of data

**Description**

This sets the currently selected table (as accessed with getdata) to the next iteration. If there is no more data, the function returns False and the selection is reset to that before the iteration started. You need to call iterinit and iterorigin before calling this. See the example below.

**Arguments**

**Returns**

bool

**Example**

See the example for the iterend function.
**ms.iterend - Function**

1.3.1 End the iteration and reset the selected table

**Description**

This sets the currently selected table (as accessed with getdata) to the table that was selected before iteration started. Use this to end the iteration prematurely. There is no need to call this if you continue iterating until iternext returns False. See the example below.

**Arguments**

**Returns**

bool

**Example**

```python
ms.open("3C273XC1.MS")
ms.selectinit(datadescid=0)
ms.iterinit(["ANTENNA1","ANTENNA2","TIME"],60.0)
ms.iterorigin()
rec=ms.getdata(["u","v","data"])
ms.iternext()
ms.iterend()
```

We open the MS, select an array and spectral window and then specify an iteration over interferometer and time, with a 60s time interval. We then set the iterator to the start of the data and get out some data. Finally we advance the iterator to the next lot of data and then end the iteration.
ms.fillbuffer.html

**ms.fillbuffer - Function**

**Description**

DEPRECATED: Read the specified data item from the table, including its flags and keep the results in an internal buffer.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>data derived item</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>ifraxis</td>
<td>Create interferometer axis if True</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```plaintext
ms.open("3C273XC1.MS")
ms.select({'antenna1':[3]})
ms.fillbuffer("PHASE",True)
```

We open the MS for reading, select a subset and then read the DATA, FLAG and FLAG\_ROW column, extract the PHASE, reorder the data to add an interferometer axis, and keep the results around in an internal buffer.
**ms.diffbuffer - Function**

**1.3.1 DEPRECATED:** Differentiate or difference the internal buffer.

**Description**

DEPRECATED: Subtract the previous point from each data point in the buffer (for window equal 2), or subtract the average over a window (for window greater than 2) from each point. The window can be in the time / row direction or the frequency / channel direction. The input data can be float or complex but the output is always float. The function returns statistics over the buffer: median for each time and channel, the average absolute deviation from the median in time and channel direction and over all pixels.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>direction</td>
<td>choose between time or channel direction: TIME or CHANNEL</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: TIME</td>
</tr>
<tr>
<td>window</td>
<td>width of averaging window in timeslots or channels; integer greater than 0</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 1</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```plaintext
ms.open("3C273XC1.MS")
ms.select({'antenna1': [3]})
ms.fillbuffer("DATA")
ms.diffbuffer("TIME", 15)
```
We open the MS for reading, select a subset and then read the DATA, FLAG and FLAG_ROW column, we then subtract the average over a 15 point time-window from each data point.
ms.getbuffer.html

**ms.getbuffer - Function**

**[1.3.1] DEPRECATED:** Return the internal buffer as a Record for access from the interpreter.

**Description**

DEPRECATED: Returns the internal buffer with either 'raw' or differenced data, flags and buffer statistics (if a difference operation was performed).

**Arguments**

**Returns**

*record*

**Example**

```python
ms.open("3C273XC1.MS")
ms.select({'antenna1': [3]})
ms.fillbuffer("PHASE")
rec=ms.getbuffer()
```

We open the MS for reading, select a subset and then read the DATA, FLAG and FLAG\_ROW column, extract the PHASE and then obtain the results in a record.
ms.clipbuffer.html

**ms.clipbuffer - Function**

[1.3.1] DEPRECATED: Clip the internal buffer with specified limits.

**Description**

DEPRECATED: This sets flags in the internal buffer based on the clip levels specified. You can flag times, channels and individual pixels based on their deviation from the median. The cliplevel is specified in units of the corresponding average absolute deviation (a robust version of rms).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>cliplevel for pixels (greater than 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixellevel</td>
<td>allowed: double</td>
</tr>
<tr>
<td></td>
<td>Default: 0.0</td>
</tr>
<tr>
<td>timelevel</td>
<td>allowed: double</td>
</tr>
<tr>
<td></td>
<td>Default: 0.0</td>
</tr>
<tr>
<td>channellevel</td>
<td>allowed: double</td>
</tr>
<tr>
<td></td>
<td>Default: 0.0</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```plaintext
ms.open("3C273XC1.MS")
ms.select({'antenna':[3]})
ms.fillbuffer("DATA")
stats=ms.diffbuffer("TIME",15)
ms.clipbuffer(6,5,5)
#2008-05-28 17:15:27 SEVERE casa::ms::open
```
We open the MS for reading, select a subset and read the data into the buffer. We then remove the average over a 15 point time window and clip the resulting data at 6 times the average absolute deviation from the median for individual pixels, and at 5 times this for channels and timeslots.
ms.asdmref - Function

[1.3.1] Test if the MS was imported with option lazy=True in importasdm and optionally change the ASDM reference

Description

If the MS is imported from an ASDM with option lazy=True, the DATA column of the MS is virtual and directly reads the visibilities from the ASDM. A reference to the original ASDM is stored with the MS. If the ASDM needs to be moved to a different path, the reference to it in the MS needs to be updated. This can be achieved with ms.asdmref(). When called with an empty string (default), the method just reports the currently set ASDM path. Return value is a string containing the new path if the path was successfully set or (in the case abspath was empty) the MS indeed contains a ASDM reference, i.e. was lazily imported.
If the ASDM does not contain an ASDM reference, the method returns an empty string. If abspath is not empty and there was an error setting the new reference, the method throws an exception.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abspath</td>
<td>new absolute path of the ASDM to be referenced (empty string = report current setting)</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>string</td>
</tr>
</tbody>
</table>

Returns

string

Example

Set the path to the referenced ASDM to "/home/alma/myanalysis/uid___A12345_X678_X910":
ms.open("uid___A12345_X678_X910.ms",False)
ms.asdmref("/home/alma/myanalysis/uid___A12345_X678_X910")
ms.close()

Test if the MS was imported with lazy=True and therefore references an ASDM:
ms.open("uid___A12345_X678_X910.ms")
myref = ms.asdmref()
ms.close()
if myref=='':
    print "This MS does not reference an ASDM."
else:
    print "This MS references the ASDM ", myref
**ms.setbufferflags** - Function

**1.3.1** DEPRECATED: Set the flags in the buffer

**Description**

DEPRECATED: Replace the flag and flag_row fields in the internal buffer with those in the input record. The input record can be e.g., a modified version of the record returned by getbuffer(). The other fields in the record are ignored.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>record with flag and flag_row allowed: record</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>flags</td>
<td>record with flag and flag_row</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
ms.open("3C273XC1.MS",False)
ms.select({'antennai':[3]})
ms.fillbuffer("PHASE")
rec=ms.getbuffer()  
rec['flag_row'][17]=True
ms.setbufferflags(rec)
```

We open the MS for reading, select a subset and read the data. We get the data into casapy, flag timeslot 17 and put the modified flags back into the buffer.
ms.writebufferflags.html

**ms.writebufferflags - Function**

[1.3.1 DEPRECATED: Write the flags in the internal buffer back to the table.]

**Description**

DEPRECATED Takes the flag and flag_row field in the internal buffer and writes them back to the FLAG and FLAG_ROW column in the currently selected table.

**Arguments**

**Returns**

bool

**Example**

```python
ms.open("3C273XC1.MS",False)
ms.select({'antenna1':[3]})
ms.fillbuffer("PHASE")
rec=ms.getbuffer()
rec['flag_row'][17]=True
ms.setbufferflags(rec)
ms.writebufferflags()
```

We open the MS for reading, select a subset and read the data. We get the data into casapy, flag timeslot 17 and put the modified flags back into the buffer. We then write the buffer flags back to the table, causing the corresponding data to be marked flagged on subsequent access.
ms.clearbuffer.html

**ms.clearbuffer - Function**

1.3.1 DEPRECATED: Clear the internal buffer.

**Description**

DEPRECATED: Clears the internal buffer, returning the memory. This can be used after the (final) clipping/flagging operations have been performed.

**Arguments**

**Returns**

bool

**Example**

```python
ms.open("3C273XC1.MS",False)
ms.select({'antenna1':[3]})
ms.fillbuffer("PHASE")
rec=ms.getbuffer()
rec['flag_row'][17]=True
ms.setbufferflags(rec)
ms.writebufferflags()
ms.clearbuffer()
```

We open the MS for reading, select a subset and read the data. We get the data into casapy, flag timeslot 17 and put the modified flags back into the buffer. We then write the buffer flags back to the table, causing the corresponding data to be marked flagged on subsequent access. Finally we clear the internal
buffer. This step can be omitted if you are about to do another fillbuffer().
ms.continuumsub.html

**ms.continuumsub - Function**

### 1.3.1 Continuum fitting and subtraction in uv plane

**Description**

This function provides a means of continuum determination and subtraction by fitting a polynomial of desired order to a subset of channels in each time-averaged uv spectrum. The fit is used to model the continuum in all channels (not just those used in the fit), for subtraction, if desired. Use the fitspw parameter to limit the spectral windows processed and the range of channels used to estimate the continuum in each (avoid channels containing spectral lines). The default solution interval 'int' will result in per-integration continuum fits for each baseline. The mode parameter indicates how the continuum model (the result of the fit) should be used: 'subtract' will store the continuum model in the MODEL_DATA column and subtract it from the CORRECTED_DATA column; 'replace' will replace the CORRECTED_DATA column with the continuum model (useful if you want to image the continuum model result); and 'model' will only store the continuum model in the MODEL_DATA column (the CORRECTED_DATA is unaffected).

It is important to start the ms tool with nomodify=False so that changes to the dataset will be allowed (see example below). For now, the only way to recover the un-subtracted CORRECTED_DATA column is to use calibrater.correct() again.

Note that the MODEL_DATA and CORRECTED_DATA columns must be present for continuumsub to work correctly. The function will warn the user if they are not present, and abort. To add these scratch columns (for now), close the ms tool, then start a calibrater or an imager tool, which will add the scratch columns. Then restart the ms tool, and try continuumsub again.

Options for shifting known bright sources to the phase center and for editing based on the rms fit will be added in the near future.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>Select fields to fit</td>
<td>allowed: any</td>
</tr>
<tr>
<td>fitspw</td>
<td>Spectral windows/channels to use for fitting the continuum; default all spectral windows in all channels</td>
<td>allowed: any</td>
</tr>
<tr>
<td>spw</td>
<td>Select spectral windows and channels from which to subtract a continuum estimate; default: all channels in all spectral windows for which the continuum was estimated</td>
<td>allowed: any</td>
</tr>
<tr>
<td>solint</td>
<td>Continuum fit timescale (units optional)</td>
<td>allowed: any</td>
</tr>
<tr>
<td>fitorder</td>
<td>Polynomial order for fit</td>
<td>allowed: int</td>
</tr>
<tr>
<td>mode</td>
<td>Desired use of fit model (see below)</td>
<td>allowed: string</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
</tr>
</tbody>
</table>

**Example**

```python
gp = ms.fromfits('ngc5921.ms','/aips++/data/demo/NGC5921.fits')
gp.close()
ms.open('ngc5921.ms') # add MODEL_DATA, CORRECTED_DATA columns
ms.continuumsub(field=2,fitspw='0:5~9;50~59',
                solint=0.0,fitorder=1,mode='sub')
gp.done()
```
This example will fit a linear continuum to channels 5-9 and 50-59 in spectral window 0 in each scan-averaged spectrum for field 2, and store the result in the MODEL\_DATA column and subtract it from the CORRECTED\_DATA column.
ms.done.html

**ms.done - Function**

Closes the ms tool

**Description**

You should call close() when you are finished using the ms tool to close the measurement set table and free any associated file locks. The measurement set is not deleted.

**Arguments**

**Returns**

bool

**Example**

```c
ms.open("3C273XC1.MS")
...
ms.done()
```
ms.msseltoindex.html

**ms.msseltoindex - Function**

[1.3.1] Returns ids of the selection used

**Description**

Utility function that will return the ids of the selection used.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>vis</td>
<td>Measurementset for which this selection applies</td>
<td>string</td>
<td>variant</td>
</tr>
<tr>
<td>spw</td>
<td>Spectral Window Ids (0 relative) to select; -1 interpreted as all</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>field</td>
<td>Field Ids (0 relative) or Field names (msselection syntax and wildcards are used) to select</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>baseline</td>
<td>Antenna Ids (0 relative) or Antenna names (msselection syntax and wildcards are used) to select</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>time</td>
<td>Limit data selected to be within a given time range. Syntax is the defined in the msselection link</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>scan</td>
<td>Limit data selected on scan numbers. Syntax is the defined in the msselection link</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>uvrang</td>
<td>Limit data selected on uv distance. Syntax is the defined in the msselection link</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>observation</td>
<td>Select data by observation ID(s). The syntax is the same as for scan numbers.</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>polarization</td>
<td>Select data by polarization(s).</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>taql</td>
<td>For the TAQL experts, flexible data selection using the TAQL syntax</td>
<td>string</td>
<td>variant</td>
</tr>
</tbody>
</table>
Returns
record

Example

```python
a = ms.msseltoindex(vis='3C273XC1.MS', field='3C*')
print a['field']
# [0]
print a
#{'antenna1': array([], dtype=int32),
  'antenna2': array([], dtype=int32),
  'channel': array([], shape=(0, 0), dtype=int32),
  'field': array([0]),
  'scan': array([], dtype=int32),
  'spw': array([], dtype=int32),
  'obsids': array([], dtype=int32)}
```

Field name '3C*', in this case 3C273, corresponds to field id 0.

N.B.: The return values of unspecified fields (like antenna* and spw in the above example) will be left empty - this does not mean that selection excludes all antennas!

Some fields (like 'field') are checked against the subtables of vis, but others are not. For example, field='123~132' will produce an error if vis does not have fields 123 to 132, but for scan and obsids '123~132' would just return an array of integers from 123 to 132 regardless of whether vis has those scan or observation IDs. (The difference comes from it being quicker to check a subtable than the main table.)
ms.hanningsmooth.html

**ms.hanningsmooth - Function**

1.3.1 Hanning smooth the frequency channels to remove Gibbs ringing.

**Description**

This function Hanning smooths the frequency channels with a weighted running average of smoothedData[i] = 0.25*correctedData[i-1] + 0.50*correctedData[i] + 0.25*correctedData[i-1]. The first and last channels are flagged. Inclusion of a flagged value in an average causes that averaged data value to be flagged.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>datacolumn</td>
<td>the name of the MS column into which to write the smoothed data</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>corrected</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
ms.open('ngc5921.ms',nomodify=False)
ms.hanningsmooth('data')
ms.close()
```
ms.uvsub.html

**ms.uvsub - Function**

1.3.1 Subtract model from the corrected visibility data.

**Description**

This function subtracts model visibility data from corrected visibility data leaving the residuals in the corrected data column. If the parameter reverse is set True, this process is reversed.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>reverse</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>When False subtracts model from visibility data; when True adds model to visibility data</td>
</tr>
<tr>
<td>allowed</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

The following example subtracts a model from the visibility data leaving the residuals in the corrected data column.

```python
ms.open('ngc5921.ms', nomodify=False)
ms.uvsub()
ms.close()
```

The following example adds the model back into the residuals.

```python
ms.open('ngc5921.ms', nomodify=False)
ms.uvsub(reverse=True)
ms.close()
```
ms.addephemeris.html

**ms.addephemeris - Function**

**1.3.1** Connect an ephemeris table with the MS FIELD table

**Description**

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The unique id number to give to this ephemeris (will overwrite pre-existing ephemeris of same id, -1 will use next unused id)</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>ephemerisname</td>
<td>The name of the ephemeris table which is to be copied into the MS</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>comment</td>
<td>Comment string (no spaces, will be part of a file name)</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>field</td>
<td>Field id(s) (0-based) or fieldname(s) to connect this ephemeris to</td>
<td>any</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
ms.addephemeris(id=0, ephemerisname="Titan_55002-55003dUTC.tab", comment="JPLTitan", field="Titan")
```

815
ms.ngetdata.html

**ms.ngetdata - Function**

1.3.1 Read values from the measurement set. Use this method instead of the older getdata() method which is marked for deprecation.

**Description**

This method extracts the data as specified in the items parameter. The data is returned as a record with each item as a separate key in the record (all in lower case).

Unless the iterator was initialized with a niterinit(), this method initializes the iterator as niterinit(“..”),0,0,0,False).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>items</td>
<td>Item names (NOT USED)</td>
<td>stringArray</td>
<td></td>
</tr>
<tr>
<td>ifraxis</td>
<td>Create interferometer axis if True (NOT USED)</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>ifraxisgap</td>
<td>Gap size on ifr axis when antenna1 changes (NOT USED)</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>increment</td>
<td>Row increment for data access (NOT USED)</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>average</td>
<td>Average the data in time or over rows (NOT USED)</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

record
ms.niterinit - Function

Initialize for iteration over an ms. Use this method instead of the older iterinit() method which is marked for deprecation.

Description

Arguments

Inputs

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>columns</td>
<td>Vector of column names (case sensitive). This parameter is not used and is</td>
</tr>
<tr>
<td></td>
<td>here only for backwards compatibility with the iterinit() method.</td>
</tr>
<tr>
<td>interval</td>
<td>Time interval in seconds (greater than 0), to group together in iteration.</td>
</tr>
<tr>
<td>maxrows</td>
<td>Max number of rows (greater than 0) to return in iteration.</td>
</tr>
<tr>
<td>adddefaultsortcolumns</td>
<td>Add the default sort columns</td>
</tr>
</tbody>
</table>

Returns

bool
ms.niterorigin - Function

Set the iterator to the start of the data. Use this method instead of the older iterorigin() method which is marked for deprecation.

Description

Set or reset the iterator to the start of the currently specified iteration. You need to call this before attempting to iteratively retrieve data with ngetdata. You can set the iteration back to the start before you reach the end of the data. You need to call iterinit before calling this. See the example below.

Arguments

Returns

bool

Example

See the example for the niterend function.
ms.niternext.html

**ms.niternext - Function**

1.3.1 Advance the iterator to the next lot of data. Use this method instead of the older iternext() method which is marked for deprecation.

**Description**

This sets the currently selected table (as accessed with ngetdata) to the next iteration. If there is no more data, the function returns False. You need to call iterinit and iterorigin before calling this. See the example below.

**Arguments**

**Returns**

bool

**Example**

See the example for the niterend function.
ms.niterend - Function

1.3.1 Query if there are more iterations left in the iterator. Use this method instead of the older iterend() method which is marked for deprecation.

Description

The serves redundant purpose and is here only for backward compatibility. This method returns True if there are no more iterations left. I.e., the iterations have ended. This same information is also returned by niternext(). With the use of the VisibilityIterator in the niterinit(), niterorigin(), niternext() methods, the iterator is set to the original state by calling niterinit() at any time.

See the example below.

Arguments

Returns

bool

Example

```python
ms.open("3C273XC1.MS")
staql={'baseline':'1 & 2'};
ms.msselect(staql);
ms.niterinit([" "],60.0)
ms.niterorigin()
while (!ms.niterend()):
    rec=ms.ngetdata(["u","v","data"])
    ms.niternext()
ms.close()
```

We open the MS, select a baseline and then specify an iteration
over time, with a 60s time interval. We then set the iterator to the start of the data and get out some data. We advance the iterator to the next lot of data and continue till the end of iterations is indicated. Finally, we close the ms tool which restores the tool to its original state.
1.3.2 msmetadata - Tool

Operations to retrieve metadata from a measurement set

Requires:

Synopsis

Description

The msmetadata tool provides methods to retrieve metadata from measurement sets.

Attaching to a Measurement Set

The simplest and most common way to attach an msmetadata tool to a measurement set is to use the msmetadata.open method which requires that you specify the name of the measurement set table.

NOTE: Any modifications to an MS while an associated msmetadata tool is open will not be reflected in the msmetadata tool. You must close and reopen the tool if you want to capture changes made to metadata of an MS if such a change occurs.

Example:

```python
msmetadata.open("3C273XC1.MS")
# get the number of spectral windows
nspw = msp_metadata.nspw()
msmetadata.done()
```

We open the tool by querying the MS for its metadata. We then get the number of spectral windows in the dataset and close the tool.

Methods

- almaspws: Get a list of spectral window IDs with ALMA-specific attributes.
- antennadiameter: Get the diameter for the specified antenna.
- antennaid: Get the zero-based antenna ID for the specified antenna name.
- antennanames: Get the names of the antennas for the specified zero-based antenna IDs.
- antennaoffset: Get the offset position of the specified antenna relative to the array reference position.
- antennaposition: Get the position of the specified antenna.
- antennastations: Get the station names of the specified antennas.
- antennasforscan: Get an array of the unique antenna IDs for the specified scan, observation ID, and array ID.
- bandwidths: Get the bandwidths in Hz for the specified spectral windows. If spw less than zero, return bandwidths for all spectral windows.
- baseband: Get the baseband for the specified spectral window.
baselines
chanavgspws
chanelbwspws
chanfreqs
chares
chanwidths
close
corprodsforpol
corrtypesforpol
datadescids
done
effexposuretime
exposuretime
fdmspws
fieldnames
fieldslorintent
fieldslorname
fieldslorscan
fieldslorscans
fieldslorsource
fieldslorspw
fieldslortimes
intents
intentsforfield
intentsforobs
intentsforobsptime
meanfreq
name
nantennas
namesforfields
namesforspws
nbaselines
nchan
ncorrforpol
nfields
nobservations
nspw
nstates
nscans
nsources
nrows
observers
observatorynames
observatoryposition
open
Get the phasecenter direction from a field ID and time if necessary.
Get the pointing direction for antennas at the specified row number in the main MS table.
Get the polarization ID associated with the specified data description ID.
Get an array projects as they are listed in the OBSERVATIONS table.
Get the values of the PROPER_MOTION column from the SOURCE table.
Get the reference direction from a field ID and time if necessary.
Get the reference frequency of the specified spectral window.
Get the rest frequencies from the SOURCE table for the specified source and spectral window.
Get an array of the unique scan numbers associated with the specified field, observation ID, and array ID.
Get a dictionary of which maps field ID to scan numbers for the specified observation ID.
Get an array of the unique scan numbers associated with the specified intent, observation ID, and array ID.
Get a dictionary of which maps spw ID to scan numbers for the specified observation ID.
Get an array of the unique scan numbers for the specified state, observation ID, and array ID.
Get the schedule information for the specified observation ID.
Get the values of the DIRECTION column from the SOURCE table.
Get the source ID from the field table for the specified field ID.
Get the values of the SOURCE_ID column from the SOURCE table.
Get the values of the SOURCE_NAME column from the SOURCE table.
Get the spws associated with the specified baseband or dictionary that maps baseband to spws.
Get the spectral window ID associated with the specified data description ID.
Get an array of the unique spectral window IDs for the specified field.
Get a dictionary which maps field IDs to spectral window IDs.
Get an array of the unique spectral window IDs for the specified intent.
Get the IDs of the specified spw names.
Get an array of the unique spectral window IDs for the specified scan number, observation ID, and array ID.
Get a dictionary which maps scan number to spectral windows for the specified observation ID.
Get an array of the unique state IDs for the specified scan number, observation ID, and array ID.
Get dictionary summarizing the MS.
Get an array of spectral window IDs used for TDM. These are windows that have 64, 128, or 256 channels.
Get the time range for the specified observation ID.
Get an array of the unique times for the specified field.
Get an array of the unique times for the specified intent.
Get the unique times for the specified scan number, observation ID, and array ID.
Get an array of the unique times for the specified scan numbers, observation ID, and array ID.
Get the spectral transitions from the SOURCE table for the specified source and spectral window.
Get an array of spectral window IDs used for WVR. These are windows that have 4 channels.
msmetadata.almaspws.html

**msmetadata.almaspws - Function**

L3.2 Get a list of spectral window IDs with ALMA-specific attributes.

**Description**

Get spectral window IDs based on ALMA-specific criteria. The inputs are or’ed together to form the returned list. If complement=True, then the complement of the selection is returned.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>chavg</td>
<td>Get channel average spectral windows?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>fdm</td>
<td>Get FDM spectral windows?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>sqld</td>
<td>Get square law (i.e. total power) detector spectral windows?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>tdm</td>
<td>Get TDM spectral windows?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>wvr</td>
<td>Get WVR spectral windows?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>complement</td>
<td>Return the complement of the selected set?</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

intArray

**Example**

826
msmd.open("my.ms")
# get all square law detector spectral window IDs
msmd.almaspws(sqld=True)
# get all spectral window IDs other than those associated with square law detectors
msmd.almaspws(sqld=True, complement=True)
msmetadata.antennadiameter.html

**msmetadata.antennadiameter - Function**

1.3.2 Get the diameter for the specified antenna.

**Description**

Get the diameter for the specified antenna. The antenna can be specified either by its zero-based ID from the ANTENNA table or by its name in that table. The returned dictionary is a valid quantity. If a negative integer is provided for the antenna, then all antenna diameters will be returned in a dictionary that has keys that are the antenna IDs and values that are dictionaries, each being a valid quantity representing the diameter for that antenna ID.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Zero-based antenna in the ANTENNA table, or antenna name. A negative integer will cause all antenna diameters to be returned.</th>
</tr>
</thead>
<tbody>
<tr>
<td>antenna</td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant -1</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
msmd.open("my.ms")
# Get the diameter of the antenna named "VB2"
diameter = msmd.antennadiameter("VB2")
msmd.done()
```
**msmetadata.antennaisds - Function**

Get the zero-based antenna ID for the specified antenna name.

**Description**

Get the zero-based antenna IDs for the specified antenna names and the specified diameter range. An array of unique IDs in order of the specified names is returned. If no names and no diameter range is specified, all IDs are returned.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Antenna names (string or string array) for which to get the corresponding IDs. Note that * matches any number of characters of all character classes.</td>
</tr>
<tr>
<td>mindiameter</td>
<td>Minimum antenna diameter, expressed as a quantity.</td>
</tr>
<tr>
<td>maxdiameter</td>
<td>Maximum antenna diameter, expressed as a quantity.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>variant 0m</td>
<td>variant 0m</td>
</tr>
<tr>
<td>any</td>
<td>variant 1pc</td>
</tr>
</tbody>
</table>

**Returns**

intArray

**Example**

```python
msmd.open("my.ms")
# get the zero-based antenna IDs for the antenna named "VB2"
antenna_id = msmd.antennaisds("VB2")[0]
# get the zero-based antenna IDs for all antennas with diameters between 9m and 11m
```
antenna_ids = msmd.antennaids(mindiameter="9m", maxdiameter=qa.quantity("11m"))
msmd.done()
msmetadata.antennanames.html

**msmetadata.antennanames - Function**

1.3.2 Get the names of the antennas for the specified zero-based antenna IDs.

**Description**

Get the name of the antenna for the specified zero-based antenna ID. If `antennoids` is not specified, all antenna names are returned.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>antennoids</td>
<td>Zero-based antenna IDs (int or int array) for which to get the antenna names.</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant -1</td>
</tr>
<tr>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

stringArray

**Example**

```python
msmd.open("my.ms")
# get the name associated with antenna ID 31
antenna_name = msmd.antennanames(31)[0]
msmd.done()
```
msmetadata.antennaoffset.html

msmetadata.antennaoffset - Function

1.3.2 Get the offset position of the specified antenna relative to the array reference position.

Description

Get the offset position of the specified antenna relative to the array reference position. Antenna may be specified as a zero-based integer (row number in the ANTENNA table) or a string representing a valid antenna name. The returned record contains the longitude, latitude, and elevation offsets as quantity records. The reported longitude and latitude offsets are measured along the surface of a sphere whose center is coincident with the center of the earth and whose surface contains the observatory reference position.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Zero-based antenna in the ANTENNA table, or antenna name.</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant 0</td>
</tr>
</tbody>
</table>

Returns

record

Example

```python
msmd.open("my.ms")
# get the offset of the (zero-based) 3rd antenna in the ANTENNA table
antenna_offset = msmd.antennaoffset(3)
# get the offset of antenna DV02
antenna_offset = msmd.antennaoffset(’DV02’)
msmd.done()
```
**msmetadata.antennaposition** - Function

1.3.2 Get the position of the specified antenna.

**Description**

Get the position of the specified antenna. The returned record represents a position measure, and can be used as such by the measures (me) tool.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>Zero-based antenna ID in the ANTENNA table or antenna name.</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant 0</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
msmd.open("my.ms")
# get the position of the (zero-based) 3rd antenna in the ANTENNA table
antenna_position = msmd.antennaposition(3)
# get the position of the antenna named DV07
antenna_position = msmd.antennaposition("DV07")
msmd.done()
```
msmetadata.antennastations - Function

1.3.2 Get the station names of the specified antennas.

Description

Get the station names of the specified antennas.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>which</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero-based antenna ID(s) in the ANTENNA table or antenna name(s). Single numeric id less than zero retrieves all station names.</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant -1</td>
</tr>
</tbody>
</table>

Returns

stringarray

Example

```
msmd.open("my.ms")
# get all station names
stations = msmd.antennastations(-1)
# get the stations of the antennas named DV07 and DV01
stations = msmd.antennaposition(["DV07", "DV01"])
msmd.done()
```
msmetadata.antennasforscan.html

**msmetadata.antennasforscan** - Function

1.3.2 Get an array of the unique antenna IDs for the specified scan, observation ID, and array ID.

**Description**

Get an array of the unique antenna IDs for the specified scan, observation ID, and array ID.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>Scan number for which to return the intents.</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>obsid</td>
<td>Observation ID. If less than 0, all observation IDs are used.</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>arrayid</td>
<td>Array ID. If less than 0, all array IDs are used.</td>
<td>int</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Returns**

intArray

**Example**

```python
msmd.open("my.ms")
# get the antennas associated with scan 4 (all observation IDs, all array IDs)
antennas = msmd.antennasforscan(4)
msmd.done()
```
msmetadata.bandwidths.html

msmetadata.bandwidths - Function

1.3.2 Get the bandwidths in Hz for the specified spectral windows. If spw less than zero, return bandwidths for all spectral windows.

Description

Get the bandwidths in Hz for the specified spectral windows. If spw less than zero, return bandwidths for all spectral windows.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>spw</td>
</tr>
<tr>
<td>allowed:</td>
</tr>
<tr>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

anyvariant

Example

```python
msmd.open("my.ms")
# get bandwidth for spectral window 2.
baseband = msmd.bandwidth(2)
msmd.done()
```
msmetadata.baseband.html

**msmetadata.baseband - Function**

1.3.2 Get the baseband for the specified spectral window.

**Description**

Get the baseband for the specified spectral window.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spw</td>
<td>Spectral window ID.</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

int

**Example**

```python
msmd.open("my.ms")
# get baseband for spectral window 2.
baseband = msmd.baseband(2)
msmd.done()
```

837
msmetadata.baselines.html

**msmetadata.baselines - Function**

1.3.2 Get a two dimensional boolean array representing baselines for data recorded in the MS.

**Description**

Get a two dimensional boolean array representing baselines for data recorded in the MS. A value of True means there is at least one row in the MS main table for that baseline, False means no rows for that baseline. Autocorrelation "baseline" information is also present via the values along the diagonal.

**Arguments**

**Returns**

anyvariant

**Example**

```python
msmd.open("my.ms")
# get the baseline matrix for this data set
baselines = msmd.baselines()
msmd.done()
```
msmetadata.chanavgspws.html

**msmetadata.chanavgspws - Function**

1.3.2 Get an array of spectral window IDs used for channel averages. These are windows that do have 1 channel.

**Description**

Get an array of spectral window IDs used for channel averages. These are windows that do have 1 channel.

**Arguments**

**Returns**

intArray

**Example**

```
msmd.open("my.ms")
# get the spectral window IDs used for channel averages.
chan_avg_spws = msmd.chanavgspws()
msmd.done()
```
msmetadata.chaneffbws - Function

1.3.2 Get an array of channel effective bandwidths for the specified spectral window.

Description

Get an array of channel effective bandwidths for the specified spectral window. The parameter asvel indicates if velocity widths (True) or frequency widths (False) should be returned. The unit parameter specifies the units that the returned values should have. If empty (default), ”Hz” will be used if asvel=False, or ”km/s” will be used if asvel=True.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>spw</td>
<td>Spectral window ID.</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>unit</td>
<td>Desired unit of returned quantities. Empty means ”Hz”</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>if asvel=False, ”km/s” if asvel=True.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>asvel</td>
<td>Should return values be equivalent velocity widths?</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

Returns
doubleArray

Example

msmd.open("my.ms")
# get the channel effective bandwidths for spectral window 2, in m/s
chan_ebw = msmd.chaneffbws(2, "m/s", True)
msmd.done()
**msmetadata.chanfreqs - Function**

Get an array of channel frequencies for the specified spectral window.

**Description**

Get an array of channel frequencies for the specified spectral window.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>spw</td>
<td>Spectral window ID.</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>unit</td>
<td>Convert frequencies to this unit.</td>
<td>string</td>
<td>Hz</td>
</tr>
</tbody>
</table>

**Returns**

doubleArray

**Example**

```python
msmd.open("my.ms")
# get the channel frequencies for spectral window 2.
chan_freqs = msmd.chanfreqs(2)
msmd.done()
```
msmetadata.chanres.html

msmetadata.chanres - Function

1.3.2 Get an array of channel resolutions for the specified spectral window.

Description

Get an array of channel resolutions for the specified spectral window. The parameter asvel indicates if velocity widths (True) or frequency widths (False) should be returned. The unit parameter specifies the units that the returned values should have. If empty (default), "Hz" will be used if asvel=False, or "km/s" will be used if asvel=True.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>spw</td>
<td>Spectral window ID.</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>unit</td>
<td>Desired unit of returned quantities. Empty means &quot;Hz&quot; if asvel=False, &quot;km/s&quot; if asvel=True.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>asvel</td>
<td>Should return values be equivalent velocity resolutions?</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

Returns
doubleArray

Example

```
msmd.open("my.ms")
# get the channel resolutions for spectral window 2, in m/s
chan_res = msmd.chanres(2, "m/s", True)
msmd.done()
```
msmetadata.chanwidths.html

**msmetadata.chanwidths - Function**

Get an array of channel widths for the specified spectral window.

**Description**

Get an array of channel widths for the specified spectral window.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>spw</td>
<td>Spectral window ID.</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>unit</td>
<td>Convert frequencies to this unit.</td>
<td>string</td>
<td>Hz</td>
</tr>
</tbody>
</table>

**Returns**

doubleArray

**Example**

```python
msmd.open("my.ms")
# get the channel widths for spectral window 2.
chan_freqs = msmd.chanwidths(2)
msmd.done()
```
msmetadata.close - Function

1.3.2 Close this tool and reclaim system resources associated with it.

Description

This method will close the tool and reclaim system resources it has been using. Returns true if successful.

Arguments

Returns

bool

Example

msmd.open("my.ms")
# do things with tool
# finish, close tool and free up resources.
msmd.close()
msmetadata.corrprodsforpol.html

msmetadata.corrprodsforpol - Function

1.3.2 Get the correlation products associated with the specified polarization ID

Description

Get the correlation products associated with the specified polarization ID.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>pol</td>
<td>Polarization ID. Must be nonnegative.</td>
<td>int</td>
<td>-1</td>
</tr>
</tbody>
</table>

Returns

anyvariant

Example

```python
msmd.open("my.ms")
# get correlation products for polarization ID 3
corrprods = msmd.corrprodsforpol(3)
msmd.done()
```
msmetadata.corrtypesforpol.html

msmetadata.corrtypesforpol - Function

1.3.2 Get the correlation types associated with the specified polarization ID

Description

Get the correlation types associated with the specified polarization ID.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Descriptions</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pol</td>
<td>Polarization ID. Must be nonnegative.</td>
<td>int</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>

Returns

intArray

Example

```python
msmd.open("my.ms")
# get correlation types for polarization ID 3
corrtypes = msmd.corrtypesforpol(3)
msmd.done()
```
msmetadata.datadescids.html

msmetadata.datadescids - Function

1.3.2 Get the data description IDs associated with the specified spectral window and/or polarization ID

Description

Get a list of data description IDs associated with the specified spectral window ID and/or polarization ID. Values of less than zero for either means all IDs should be used in the selection.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spw</td>
<td>Spectral window ID. Less than zero implies any, allowed: int Default: -1</td>
</tr>
<tr>
<td>pol</td>
<td>Polarization ID. Less than zero implies any. allowed: int Default: -1</td>
</tr>
</tbody>
</table>

Returns

intArray

Example

msmd.open("my.ms")
# get all data description IDs associated with spw 2.
msmd.datadescids(spw=2)
# same as before but limit the IDs returned to those associated with # polarization ID 3
msmd.datadescids(spw=2, pol=3)
msmd.done()
msmetadata.done.html

**msmetadata.done - Function**

1.3.2 Close this tool and reclaim system resources associated with it.

**Description**

This method will close the tool and reclaim system resources it has been using. Returns true if successful.

**Arguments**

**Returns**

bool

**Example**

```plaintext
msmd.open("my.ms")
# do things with tool
# finish, close tool and free up resources.
msmd.done()
```
msmetadata.effexposuretime.html

**msmetadata.effexposuretime - Function**

1.3.2 Get the effective exposure (on-source integration time)

**Description**

Get the effective exposure time (equivalent to what might be more commonly known as total integration time or total sample time) is calculated by summing over all rows in the main MS table, excluding autocorrelations or rows where FLAG_ROW is false, thusly:

\[
\frac{\text{sum}(\text{over } i) (\text{exposure}[i] \times \text{sum}(\text{over } j)(\text{UFBW}[i, j]) / \text{ncorrelations}[i])}{\text{nmaxbaselines}}
\]

where exposure[i] is the value of EXPOSURE for the ith row, the inner sum is performed over each correlation for that row, UFBW is the unflagged fractional bandwidth is determined by summing all the widths of the unflagged channels for that correlation and dividing by the total bandwidth of all spectral windows observed at the timestamp of row i, ncorrelations is the number of correlations determined by the number of rows in the FLAG matrix for MS row i, and nmaxbaselines is the maximum number of antenna pairs, \(\text{nantennas} \times (\text{nantennas}-1) / 2\), where nantennas is the number of antennas in the ANTENNA table. This method returns a quantity (a dictionary having a numerical value and a string unit).

**Arguments**

**Returns**

record

**Example**

```python
msmd.open("my.ms")
# get the effective exposure time.
exposure_time = msmd.effexposuretime()
msmd.done()
```
msmetadata.exposuretime.html

**msmetadata.exposuretime - Function**

1.3.2 Get the exposure time for the specified scan, spwid, polarizaiton ID, array ID, and observation ID.

**Description**

Get the exposure time for the specified scan, spwid, polarizaiton ID, array ID, and observation ID. This is the exposure time of the record with the lowest time stamp of the records associated with these parameters. Returns a quantity dictionary. If polid is not specified (or specified and negative) and there is only one polarization ID in for the specified combination of scan, spwid, obsID, and arrayID, then that polarization ID is used. If there are multiple polarization IDs for the combination of other parameters, a list of these is logged and an empty dictionary is returned.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>Scan number.</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>spwid</td>
<td>Spectral window ID.</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>polid</td>
<td>Polarization ID.</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>obsid</td>
<td>Observation ID.</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>arrayid</td>
<td>Array ID.</td>
<td>int</td>
<td>0</td>
</tr>
</tbody>
</table>

**Returns**

record
Example

msmd.open("my.ms")
# get the exposure time for scan 1, spwid 2, and polid 3
# for obsid=0 and arrayid=0
integration_time = msmd.getexposuretime(scan=1, spwid=2, polid=3)
msmd.done()
msmetadata.fdmspws.html

**msmetadata.fdmspws - Function**

1.3.2 Get an array of spectral window IDs used for FDM. These are windows that do not have 64, 128, or 256 channels.

**Description**

Get an array of spectral window IDs used for FDM. These are windows that do not have 64, 128, or 256 channels.

**Arguments**

**Returns**

intArray

**Example**

```python
msmd.open("my.ms")
# get the spectral window IDs used for FDM.
fdm_spws = msmd.fdmspws()
msmd.done()
```
msmetadata.fieldnames.html

msmetadata.fieldnames - Function

1.3.2 Get an array of field names as they appear in the FIELD table.

Description

Get an array of field names as they appear in the FIELD table.

Arguments

Returns

stringArray

Example

msmd.open("my.ms")
# get list of field names in the ms
fieldnames = msmd.fieldnames()
msmd.done()
msmetadata.fieldsforintent.html

msmetadata.fieldsforintent - Function

1.3.2 Get an array of the unique fields for the specified intent.

Description

Get an array of the unique fields for the specified intent. Note that * matches any number of characters of all character classes.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intent</td>
<td>Intent (case sensitive) for which to return the fields.</td>
</tr>
<tr>
<td>allowed</td>
<td>If true, return the field names. If false, return the zero-based field IDs.</td>
</tr>
</tbody>
</table>

Returns

any

Example

msmd.open("my.ms")
# get the field names for intent "observe target"
field_names = msmd.fieldsforintent("observe target", True, regex=False)
# get the field IDs for intent "observe target"
field_IDs = msmd.fieldsforintent("observe target", False, regex=False)
# get all field IDs for all intents which contain 'WVR'
field_IDs = msmd.fieldsforIntent("*WVR*")
msmd.done()
msmetadata.fieldsforname.html

**msmetadata.fieldsforname - Function**

**1.3.2** Get an array of the unique, zero-based field IDs for the specified field name.

**Description**

Get an array of the unique, zero-based field IDs for the specified field name. If the field name is the empty string (the default), a list of all unique field IDs in the main table of the MS will be returned.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Field name (case sensitive) for which to return the fields, allowed: string</td>
</tr>
</tbody>
</table>

**Returns**

intArray

**Example**

```python
msmd.open("my.ms")
# get the field IDs for field name "Enceladus"
fields = msmd.fieldsforname("Enceladus")
msmd.done()
```
msmetadata.fieldsforscan.html

msmetadata.fieldsforscan - Function

L3.2 Get an array of the unique fields for the specified scan number, observation ID, and array ID.

Description

Get an array of the unique fields for the specified scan number, observation ID, and array ID.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>Scan number for which to return the fields.</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>asnames</td>
<td>If true, return the field names. If false, return the zero-based field IDs.</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>obsid</td>
<td>Observation ID. A negative value means use all observation IDs.</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>arrayid</td>
<td>Array ID. A negative value means use all array IDs.</td>
<td>int</td>
<td>-1</td>
</tr>
</tbody>
</table>

Returns

any

Example

msmd.open("my.ms")
# get the field names for scan number 5 (for all array IDs and all observation IDs).
field_names = msmd.fieldsforscan(5, True)
# get the field IDs for scan number 5 (for all array IDs and all observation IDs)
field_IDs = msmd.fieldsforscan(5, False)
msmd.done()
msmetadata.fieldsforscans.html

**msmetadata.fieldsforscans - Function**

1.3.2 Get an array or dictionary of the unique fields for the specified scan numbers, observationID, and array ID.

**Description**

Get an array or dictionary of the unique fields for the specified scan numbers, observation ID, and array ID. If asnames=True, the values returned will be the field names, if False, they will be field IDs. If asmap=True, the structure returned will be a dictionary which maps scan number (as a string) to fields. In this case, both obsid and arrayid must be nonnegative. If asmap=False, a single array of fields is returned that matches the query. In this case, if obsid and/or arrayid are negative, then it indicates that all fields matching any obsid and/or arrayid should be returned. An empty array specified for scans means that all scans for the selected obsid and arrayid should be included.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scans</td>
<td>Scan numbers for which to return the fields.</td>
</tr>
<tr>
<td>allowed: intArray</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>intArray</td>
</tr>
<tr>
<td>asnames</td>
<td>If true, return the field names. If false, return the zero-based field IDs.</td>
</tr>
<tr>
<td>allowed: bool</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>obsid</td>
<td>Observation ID. A negative value means use all observation IDs.</td>
</tr>
<tr>
<td>allowed: int</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>-1</td>
</tr>
<tr>
<td>arrayid</td>
<td>Array ID. A negative value means use all array IDs.</td>
</tr>
<tr>
<td>allowed: int</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>-1</td>
</tr>
<tr>
<td>asmap</td>
<td>Return a dictionary mapping scan numbers to fields?</td>
</tr>
<tr>
<td>allowed: bool</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

860
Example

msmd.open("my.ms")
# get the field names for scan numbers 5 and 10 (all obsids, all arrayids)
field_names = msmd.fieldsforscan([5, 10], True)
# get the field IDs for scan numbers 5 and 10 (all obsids, all arrayids)
field_IDs = msmd.fieldsforscan([5, 10], False)
# get mapping of scans to fields for arrayid=2 and obsid=4
scans_to_fields = msmd.fieldsforscan(obsid=4, arrayid=2, asmap=True)
msmd.done()
**msmetadata.fieldsforsource - Function**

1.3.2 Get an array of the unique fields for the specified source ID.

**Description**

Get an array of the unique fields for the specified source.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>Zero-based source ID for which to return the fields.</td>
</tr>
<tr>
<td>allowed</td>
<td>int</td>
</tr>
<tr>
<td>Default</td>
<td>-1</td>
</tr>
<tr>
<td>asnames</td>
<td>If true, return the field names. If false, return the zero-based field IDs.</td>
</tr>
<tr>
<td>allowed</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

any

**Example**

```python
msmd.open("my.ms")
# get the field names for source ID 1
field_names = msmd.fieldsforsource(1, True)
# get the field IDs for source ID 1
field_IDS = msmd.fieldsforsource(1, False)
msmd.done()
```
msmetadata.fieldsforsources.html

**msmetadata.fieldsforsources - Function**

Get a map of source IDs to fields.

**Description**

Get a map of source IDs to fields. The keys (source IDs) will be strings.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asnames</td>
<td>If true, return the field names. If false, return the zero-based field IDs.</td>
</tr>
<tr>
<td>allowed: bool</td>
<td>Default: false</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
msmd.open("my.ms")
# get the source to field name map
sources_to_fields = msmd.fieldsforsources(True)
# access the field names for source 1
field = sources_to_fields["1"]
msmd.done()
```
msmetadata.fieldsforspw.html

msmetadata.fieldsforspw - Function

1.3.2 Get an array of the unique fields for the specified spectral window.

Description

Get an array of the unique fields for the specified spectral window.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spw</td>
<td>Zero-based spectral window ID for which to return the fields. allowed: int Default: -1</td>
</tr>
<tr>
<td>asnames</td>
<td>If true, return the field names. If false, return the zero-based field IDs. allowed: bool Default: false</td>
</tr>
</tbody>
</table>

Returns

any

Example

```python
msmd.open("my.ms")
# get the field names for spectral window 1
field_names = msmd.fieldsforspw(1, True)
# get the field IDs for spectral window 1
field_IDs = msmd.fieldsforspw(1, False)
msmd.done()
```
msmetadata.fieldsfortimes.html

msmetadata.fieldsfortimes - Function

1.3.2 Get an array of the unique, zero-based, field IDs for the specified time range (time-tol to time+tol).

Description

Get an array of the unique, zero-based, field IDs for the specified time range (time-tol to time+tol).

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>Time at center of time range.</td>
<td>double</td>
<td>-1</td>
</tr>
<tr>
<td>tol</td>
<td>Time on either side of center for specifying range.</td>
<td>double</td>
<td>0</td>
</tr>
</tbody>
</table>

Returns

intArray

Example

msmd.open("my.ms")
# get the field IDs associated with the specified time range
fields = msmd.fieldsfortimes(4.8428293714e+09, 20)
msmd.done()
msmetadata.intents.html

**msmetadata.intents - Function**

Get an array of the unique intents associated with the MS.

**Description**

Get an array of the unique intents associated with the MS.

**Arguments**

**Returns**

stringArray

**Example**

```python
msmd.open("my.ms")
# get the intents associated with the MS
intents = msmd.intents()
msmd.done()
```
msmetadata.intentsforfield.html

**msmetadata.intentsforfield - Function**

1.3.2 Get an array of the unique intents for the specified field.

**Description**

Get an array of the unique intents for the specified field.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>field</strong></td>
<td>Field ID or name for which to return the intents.</td>
</tr>
<tr>
<td>allowed: any</td>
<td>Default: variant -1</td>
</tr>
</tbody>
</table>

**Returns**

stringArray

**Example**

```python
msmd.open("my.ms")
# get the intents associated with field 4
intents = msmd.intentsforfield(4)
# get intents for field "MOS"
intents2 = msmd.intentsforfield("MOS")
msmd.done()
```
msmetadata.intentsforscan.html

**msmetadata.intentsforscan** - Function

**1.3.2** Get an array of the unique intents for the specified scan, observation ID, and array ID.

**Description**

Get an array of the unique intents for the specified scan, observation ID, and array ID.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>Scan number for which to return the intents.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td>obsid</td>
<td>Observation ID. A negative value means use all observation IDs.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td>arrayid</td>
<td>Array ID. A negative value means use all array IDs.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
</tbody>
</table>

**Returns**

*stringArray*

**Example**

```python
msmd.open("my.ms")
# get the intents associated with scan 4 (all obsids, all arrayids)
extents = msmd.intentsforscan(4)
msmd.done()
```
msmetadata.intentsforspw.html

**msmetadata.intentsforspw - Function**

1.3.2 Get an array of the unique intents for the specified spectral window ID.

**Description**

Get an array of the unique intents for the specified spectral window ID.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spw</td>
<td>Spectral window ID (&gt;=0) for which to return the intents.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
</tbody>
</table>

**Returns**

stringArray

**Example**

```python
msmd.open("my.ms")
# get the intents associated with spectral window ID 3
intents = msmd.intentsforspw(3)
msmd.done()
```
msmetadata.meanfreq.html

msmetadata.meanfreq - Function

Get the mean frequency for the specified spectral window.

Description

Get the mean frequency for the specified spectral window.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>spw</td>
<td>Spectral window ID.</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>unit</td>
<td>Convert frequencies to this unit.</td>
<td>string</td>
<td>Hz</td>
</tr>
</tbody>
</table>

Returns
double

Example

msmd.open("my.ms")
# get the mean frequency for spectral window 2.
mean_freq = msmd.meanfreq(2)
msmd.done()
msmetadata.name.html

**msmetadata.name - Function**

Get the name of the attached MS.

**Description**

Get the name of the attached MS.

**Arguments**

**Returns**

string

**Example**

```python
msmd.open("my.ms")
# get its name
myname = msmd.name()
msmd.done()
```
msmetadata.nantennas - Function

**1.3.2** Get the number of antennas associated with the MS.

**Description**

Get the number of antennas associated with the MS.

**Arguments**

**Returns**

int

**Example**

```python
msmd.open("my.ms")
number_of_antennas = msmd.nantennas()
msmd.done()
```
msmetadata.namesforfields.html

**msmetadata.namesforfields - Function**

1.3.2 Get the name of the specified field.

**Description**

Get the name of the specified field.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldids</td>
<td>Zero-based field IDs for which to get the names (integer or integer array).</td>
</tr>
<tr>
<td></td>
<td>Unspecified will return all field names.</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
</tbody>
</table>

**Returns**

*stringArray*

**Example**

```python
msmd.open("my.ms")
# get the name for field 8 and 2.
field_names = msmd.namesforfields([8, 2])
# get all field names
all_field_names = namesforfields()
msmd.done()
```
msmetadata.namesforspws.html

**msmetadata.namesforspws - Function**

1.3.2 Get the name of the specified spws.

**Description**

Get the name of the specified spw(s).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spwids</td>
<td>Zero-based spw ID(s) for which to get the names (integer or integer array). Unspecified will return all spw names.</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
</tbody>
</table>

**Returns**

stringArray

**Example**

```python
msmd.open("my.ms")
# get the name for spws 8 and 2.
spw_names = msmd.namesforspws([8, 2])
# get all spw names
all_spw_names = msmd.namesforspws()
msmd.done()
```
msmetadata.nbaselines.html

**msmetadata.nbaselines - Function**

1.3.2 Get the number of baselines represented in the main MS table.

**Description**

Get the number of unique baselines (antenna pairs) represented in the main MS table. This can, in theory, be less than n*(n-1)/2 (n being the number of antennas in the ANTENNA table), if data for certain baselines are not included in the main MS table. Autocorrelation "baselines" are included in this count if ac=True.

**Arguments**

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ac</td>
<td>Include auto-correlation &quot;baselines&quot;?</td>
</tr>
<tr>
<td>allowed</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

int

**Example**

```python
msmd.open("my.ms")
number_of_baselines = msmd.nbaselines()
number_of_baselines_including_ac = msmd.nbaselines(True)
msmd.done()
```
**msmetadata.nchan - Function**

1.3.2 Get the number of channels associated with the specified spectral window.

**Description**

Get the number of channels associated with the specified spectral window.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spw</td>
<td>Zero-based spw ID for which to get the number of channels. allowed: int Default:</td>
</tr>
</tbody>
</table>

**Returns**

int

**Example**

```python
msmd.open("my.ms")
nchan = msmd.nchan(3)
msmd.done()
```
msmetadata.ncorrforpol.html

**msmetadata.ncorrforpol - Function**

Get the number of correlations for the specified polarization ID.

**Description**

Get the number of correlations for the specified polarization ID. If the specified polarization ID is negative, an array of numbers of correlations is returned. The indices of that array represent polarization IDs.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>polid</td>
<td>Zero-based polarization ID. A negative number will cause all the numbers of correlations to be returned.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
</tbody>
</table>

**Returns**

variant

**Example**

```python
msmd.open("my.ms")
# get the number of correlations associated with polarization ID 4
polid = msmd.ncorrforpol(4)
# get the array of numbers of correlations from the POLARIZATION table
polids = msmd.ncorrforpol(-1)
msmd.done()
```
msmetadata.nfields.html

**msmetadata.nfields - Function**

Get the number of fields associated with the MS.

**Description**

Get the number of fields associated with the MS.

**Arguments**

**Returns**

```
int
```

**Example**

```
msmd.open("my.ms")
number_of_fields = msmd.nfields()
msmd.done()
```
msmetadata.nobservations.html

**msmetadata.nobservations - Function**

1.3.2 Get the number of observations associated with the MS from the OBSERVATIONS table.

**Description**

Get the number of observations associated with the MS from the OBSERVATIONS table.

**Arguments**

**Returns**

int

**Example**

```python
msmd.open("my.ms")
number_of_obs_ids = msmd.nobservations()
msmd.done()
```
msmetadata.nspw.html

**msmetadata.nspw** - Function

Get the number of spectral windows associated with the MS.

### Description

This method will return the number of spectral windows in the associated MS.

### Arguments

#### Inputs

<table>
<thead>
<tr>
<th>includewvr</th>
<th>Include wvr spectral windows? If false, exclude wvr windows from count.</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>true</td>
</tr>
</tbody>
</table>

### Returns

int

### Example

```python
msmd.open("my.ms")
number_of_spectral_windows = msmd.nspw()
msmd.done()
```
msmetadata.nstates.html

**msmetadata.nstates - Function**

1.3.2 Get the number of states (from the STATE table) associated with the MS.

**Description**

This method will return the number of states (number of rows in the STATES table) in the associated MS.

**Arguments**

**Returns**

int

**Example**

```python
msmd.open("my.ms")
number_of_states = msmd.nstates()
msmd.done()
```
msmetadata.nscans.html

**msmetadata.nscans - Function**

Get the number of scans associated with the MS.

---

**Description**

Get the number of scans associated with the MS.

**Arguments**

**Returns**

int

**Example**

```python
msmd.open("my.ms")
number_of_scans = msmd.nscans()
msmd.done()
```
msmetadata.nsources.html

**msmetadata.nsources - Function**

Get the number of unique values from the SOURCE_ID column in the SOURCE table.

**Description**

Get the number of unique values from the SOURCE_ID column in the SOURCE table. The number of rows in the SOURCE table may be greater than this value.

**Arguments**

**Returns**

int

**Example**

```python
msmd.open("my.ms")
number_of_unique_source_ids = msmd.nsources()
msmd.done()
```
msmetadata.nrows.html

**msmetadata.nrows - Function**

Get the number of visibilities (from the main table) associated with the MS.

**Description**

Get the number of visibilities (from the main table) associated with the MS.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoc</td>
<td>Include autocorrelation data? If False, only cross correlation rows will be summed.</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>true</td>
</tr>
<tr>
<td>flagged</td>
<td>Include flagged data? If False, only unflagged or partially flagged rows will be summed.</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>true</td>
</tr>
</tbody>
</table>

**Returns**

double

**Example**

```python
msmd.open("my.ms")
# get the total number of rows
nrows = msmd.nrows()
# get the number of cross correlation rows
ncross = msmd.nrows(auto=False)
# get the number of unflagged rows
ngood = msmd.nrows(flagged=False)
# get the number of unflagged cross correlation rows
ncrossunflagged = msmd.nrows(auto=False, flagged=False)
msmd.done()
```

884
msmetadata.observers.html

**msmetadata.observers - Function**

Get an array observers as they are listed in the OBSERVATIONS table.

**Description**

Get an array of observers as they are listed in the OBSERVATIONS table.

**Arguments**

**Returns**

*stringArray*

**Example**

```python
msmd.open("my.ms")
# get the observers
observers = msmd.observers()
msmd.done()
```
msmetadata.observatorynames.html

**msmetadata.observatorynames - Function**

1.3.2 Get an array of MS telescope (observatory) names as they are listed in the OBSERVATIONS table.

**Description**

Get an array of MS telescope (observatory) names as they are listed in the OBSERVATIONS table.

**Arguments**

**Returns**

stringArray

**Example**

```plaintext
msmd.open("my.ms")
# get the telescope names
telescope_names = msmd.telescopenames()
msmd.done()
```
msmetadata.observatoryposition.html

msmetadata.observatoryposition - Function

1.3.2 Get the position of the specified telescope.

Description

Get the position of the specified telescope.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Zero-based telescope position in the OBSERVATIONS table (see msmd.telescopenames()).</th>
</tr>
</thead>
<tbody>
<tr>
<td>which</td>
<td>int</td>
</tr>
<tr>
<td>allowed:</td>
<td>0</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
</tbody>
</table>

Returns

record

Example

msmd.open("my.ms")
# get the position of the 0th telescope
telescope_position = msmd.telescopeposition(0)
msmd.done()
msmetadata.open.html

**msmetadata.open - Function**

1.3.2 Attach the MS metadata tool to the specified MS

**Description**

Attach this tool to the specified MS.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msfile</td>
<td>Name of the existing measurement set</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>maxcache</td>
<td>Maximum cache size, in megabytes, to use.</td>
</tr>
<tr>
<td></td>
<td>allowed: float</td>
</tr>
<tr>
<td></td>
<td>Default: 50</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

msmd.open("my.ms")
# do stuff and close it
msmd.done()
msmetadata.phasecenter.html

**msmetadata.phasecenter - Function**

1.3.2 Get the phasecenter direction from a field ID and time if necessary

**Description**

Get a direction measures for the phasecenter of the field id and time specified

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fieldid</td>
<td>Zero-based field ID for which to get the phasecenter; default fieldid=0</td>
</tr>
<tr>
<td>epoch</td>
<td>Optional time, expressed as a measures epoch dictionary, if field id has a polynomial in time phasecenter or an ephemerides table attached to the ID. Default value means evaluate at the origin TIME in the FIELD table</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
msmd.open("my.ms")
# get phasecenter for field ID 1
mydir = msmd.phasecenter(1);
# if the phasecenter is a polynomial or has an ephemerides attached to
# it a time is needed to get the phase direction
ep=me.epoch('utc', '2015/03/15/15:30:55')
mydir2=msmd.phasecenter(2, ep)
msmd.done()
```
msmetadata.pointingdirection.html

**msmetadata.pointingdirection - Function**

Get the pointing direction for antennas at the specified row number in the main MS table.

**Description**

Get the pointing direction for antennas at the specified row number in the main MS table. Returns a record containing the time, antenna IDs and corresponding pointing directions.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rownum</td>
<td>Row number in the main MS table.</td>
</tr>
<tr>
<td>interpolator</td>
<td>Interpolate pointings in case the interval in the main table is shorter than that in the pointing table (often the case in fast-scanning in single dish observations)</td>
</tr>
<tr>
<td>initialrow</td>
<td>Initial guess of row index in pointing table to start search.</td>
</tr>
</tbody>
</table>

**Allowed**

- **rownum**: int
- **interpolate**: bool
- **initialrow**: int

**Default**

- **rownum**: 0
- **interpolate**: false
- **initialrow**: 0

**Returns**

- **record**

**Example**

```python
msmd.open("my.ms")
# get the pointing directions for row ID 500
dirs = msmd.pointingdirection(500)
msmd.done()
```
**msmetadata.polidfordatadesc - Function**

1.3.2 Get the polarization ID associated with the specified data description ID.

**Description**

Get the polarization ID associated with the specified data description ID. If the specified data description ID is negative, an array of polarization IDs is returned. The indices of that array represent data description IDs.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddid</td>
</tr>
</tbody>
</table>

**Returns**

variant

**Example**

```python
msmd.open("my.ms")
# get the polarization ID associated with data description ID 3
polid = msmd.polidfordatadesc(3)
# get the array of polarization IDs in the order they appear in the DATA_DESCRIPTION table
polids = msmd.polidfordatadesc(-1)
msmd.done()
```
msmetadata.projects.html

**msmetadata.projects** - Function

Get an array of projects as they are listed in the OBSERVATIONS table.

**Description**

Get an array of projects as they are listed in the OBSERVATIONS table.

**Arguments**

**Returns**

`stringArray`

**Example**

```python
msmd.open("my.ms")
# get the projects
projects = msmd.projects()
msmd.done()
```
Get the values of the PROPER_MOTION column from the SOURCE table.

**Description**

Get the values of the DIRECTION column from the SOURCE table. Returns a dictionary in which the keys are the associated zero-based row numbers, represented as strings, in the SOURCE table. The associated values are two element dictionaries, with keys "longitude" and "latitude", containing the longitudinal and latitudinal components of the proper motion, which are valid quantity dictionaries.

**Arguments**

**Returns**

record

**Example**

```python
msmd.open("my.ms")
# get PROPER_MOTION column values from the SOURCE table
mu = msmd.propermotions()
msmd.done()
# the direction associated with zero-based row number 10
mu10 = mu["10"]
```
msmetadata.refdir.html

**msmetadata.refdir - Function**

Get the reference direction from a field ID and time if necessary

**Description**

Get a direction measure for the reference direction of the field and time specified

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>Zero-based field ID or field name for which to get the reference direction; default field=0</td>
</tr>
<tr>
<td>epoch</td>
<td>Optional time, expressed as a measures epoch dictionary, if associated field has a polynomial in time reference direction or an ephemerides table attached it. Default value means evaluate at the origin TIME in the FIELD table</td>
</tr>
</tbody>
</table>

| allowed: | any |
| Default: | variant 0 |
| allowed: | record |
| Default: | |

**Returns**

record

**Example**

```python
msmd.open("my.ms")
# get reference direction for field ID 1
mydir = msmd.refdir(1);
# if the reference direction is a polynomial or has an ephemerides attached to it a time is needed to get the reference direction
ep=me.epoch('utc', '2015/03/15/15:30:55')
mydir2=msmd.phasecenter(2, ep)
```
msmd.done()
msmetadata.reffreq.html

**msmetadata.reffreq - Function**

1.3.2 Get the reference frequency of the specified spectral window.

**Description**

Get the reference frequency of the specified spectral window. The returned frequency is in the form of a valid measures dictionary.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spw</td>
<td>Zero-based spectral window ID.</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
msmd.open("my.ms")
# get the reference frequency for spw ID 20
reffreq = msmd.reffreq(20)
msmd.done()
```
msmetadata.restfreqs.html

msmetadata.restfreqs - Function

1.3.2 Get the rest frequencies from the SOURCE table for the specified source and spectral window.

Description

Get the rest frequencies from the SOURCE table for the specified source and spectral window. The return value will be a dictionary of frequency measures if the rest frequencies are defined for the specified inputs, or False if they do not.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceid</td>
<td>Zero-based source ID (from the SOURCE::SOURCE_ID column).</td>
<td>allowed: int</td>
</tr>
<tr>
<td>spw</td>
<td>Zero-based spectral window ID.</td>
<td>allowed: int</td>
</tr>
</tbody>
</table>

Returns
anyvariant

Example

msmd.open("my.ms")
# get the rest frequencies for source ID 2 and spw ID 20
reffreq = msmd.restfreqs(2, 20)
msmd.done()
msmetadata.scannumbers.html

msmetadata.scannumbers - Function

1.3.2 Get an array of the unique scan numbers associated with the MS for the specified observation ID and array ID.

Description

This method will return an array of unique scan numbers in the associated MS for the specified observation ID and array ID.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obsid</td>
<td>Observation ID. A negative value indicates all observation IDs should be used.</td>
</tr>
<tr>
<td>arrayid</td>
<td>Array ID. A negative value indicates all array IDs should be used.</td>
</tr>
</tbody>
</table>

| allowed: | int |
| Default: | -1  |

Returns

intArray

Example

msmd.open("my.ms")
# scan numbers for all obsids and all arrayids
scan_numbers = msmd.scan_numbers()
msmd.done()
msmetadata.scansforfield.html

msmetadata.scansforfield - Function

1.3.2 Get an array of the unique scan numbers associated with the specified field, observation ID, and array ID.

Description

Get an array of the unique scan numbers associated with the specified field, observation ID, and array ID.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>intent</td>
<td>Field ID or field name (case sensitive) for which to return the scan numbers.</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
<td></td>
</tr>
<tr>
<td>obsid</td>
<td>Observation ID. A negative value indicates all observation IDs should be used.</td>
<td>allowed: int</td>
</tr>
<tr>
<td>arrayid</td>
<td>Array ID. A negative value indicates all array IDs should be used.</td>
<td>allowed: int</td>
</tr>
</tbody>
</table>

Returns

intArray

Example

msmd.open("my.ms")
# get the scan numbers associated with field "planet Z" (all obsids, all arrayids)
scan_numbers = msmd.scansforfield("planet Z")
# get the scan numbers associated with field ID 5 (all obsids, all arrayids)
scan_numbers = msmd.scansforfield(5)
msmd.done()
msmetadata.scansforfields - Function

Get a dictionary of which maps field ID to scan numbers for the specified observation ID and array ID.

Description

Get a dictionary of which maps field ID to scan numbers for the specified observation ID and array ID. The keys (field IDs) will be strings. obsid and arrayid must both be non-negative.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>obsid</td>
<td>Observation ID. Must be non-negative.</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>arrayid</td>
<td>Array ID. Must be non-negative.</td>
<td>int</td>
<td>0</td>
</tr>
</tbody>
</table>

Returns

record

Example

msmd.open("my.ms")
# get the field - scan map for arrayID 1 and obsID 2
field_to_scans = msmd.scansforfields(arrayid=1, obsid=2)
# access the scans associated with field ID 2
field_to_scans2 = field_to_scans["2"]
msmd.done()
msmetadata.scansforintent.html

msmetadata.scansforintent - Function

Get an array of the unique scan numbers associated with the specified intent, observation ID, and arrayID.

Description

Get an array of the unique scan numbers associated with the specified intent, observation ID, and arrayID. The "**" character matches any number of characters from all character classes.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>intent</td>
<td>Intent (case-sensitive) for which to return the scan numbers.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>obsid</td>
<td>Observation ID. A negative value indicates all observation IDs should be used.</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>arrayid</td>
<td>Array ID. A negative value indicates all array IDs should be used.</td>
<td>int</td>
<td>-1</td>
</tr>
</tbody>
</table>

Returns

intArray

Example

```python
msmd.open("my.ms")
# get the scan numbers associated with intent "detect planet X" (all obsids, all arrayids)
scan_numbers = msmd.scansforintent("detect planet X", regex=False)
# got all the scan numbers associated with all intents which contain 'WVR' (all obsids,
```
scan_numbers = msmd.scansforintent("*WVR*")
msmd.done()
msmetadata.scansforspw.html

**msmetadata.scansforspw - Function**

1.3.2 Get an array of the unique scan numbers associated with the specified zero-based spectral window ID, observation ID, and array ID.

**Description**

Get an array of the unique scan numbers associated with the specified zero-based spectral window ID, observation ID, and array ID.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>spw</td>
<td>Zero-based spectral window ID for which to return the scan numbers.</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>obsid</td>
<td>Observation ID. A negative value indicates all observation IDs should be used.</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>arrayid</td>
<td>Array ID. A negative value indicates all array IDs should be used.</td>
<td>int</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Returns**

intArray

**Example**

```python
msmd.open("my.ms")
# get the scan numbers associated with spectral window ID 14, all obsids, all arrayids
scan_numbers = msmd.scansforspw(14)
msmd.done()
```
msmetadata.scansforspws.html

msmetadata.scansforspws - Function

1.3.2 Get a dictionary of which maps spw ID to scan numbers for the specified observation ID and array ID.

Description

Get a dictionary of which maps spw ID to scan numbers for the specified observation ID and array ID. The keys (spectral window IDs) will be strings. obsid and arrayid must both be non-negative.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>obsid</td>
<td>Observation ID. Must be non-negative.</td>
<td></td>
</tr>
<tr>
<td>arrayid</td>
<td>Array ID. Must be non-negative.</td>
<td></td>
</tr>
</tbody>
</table>

| allowed: | int |
| Default: | 0   |

Returns

record

Example

```
msmd.open("my.ms")
# get the spw - scan map for arrayID 1 and obsID 2
spw_to_scans = msmd.scansforspws(arrayid=1, obsid=2)
# access the scans associated with spw ID 2
spw_to_scans2 = spw_to_scans["2"]
msmd.done()
```
**msmetadata.scansforstate** - Function

Get an array of the unique scan numbers for the specified state, observation ID, and array ID.

### Description

Get an array of the unique scan numbers for the specified state, observation ID, and array ID.

### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>ID of state for which to return the scan numbers.</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>obsid</td>
<td>Observation ID. A negative value indicates all observation IDs should be used.</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>arrayid</td>
<td>Array ID. A negative value indicates all array IDs should be used.</td>
<td>int</td>
<td>-1</td>
</tr>
</tbody>
</table>

### Returns

intArray

### Example

```python
msmd.open("my.ms")
# get the scan numbers associated with state 2, all obsids, all arrayids
scans = msmd.scansforstate(2)
msmd.done()
```
msmetadata.scansfortimes.html

**msmetadata.scansfortimes - Function**

1.3.2 Get an array of the unique scan numbers for the specified time range (time-tol to time+tol), observation ID, and array ID.

**Description**

Get an array of the unique scan numbers for the specified time range (time-tol to time+tol), observation ID, and array ID.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>Time at center of time range.</td>
</tr>
<tr>
<td>tol</td>
<td>Time difference on either side of center for specifying range.</td>
</tr>
<tr>
<td>obsid</td>
<td>Observation ID. A negative value indicates all observation IDs should be used.</td>
</tr>
<tr>
<td>arrayid</td>
<td>Array ID. A negative value indicates all array IDs should be used.</td>
</tr>
</tbody>
</table>

| allowed:     | double |
| Default:     | -1      |
| allowed:     | double |
| Default:     | 0       |
| allowed:     | int     |
| Default:     | -1      |
| allowed:     | int     |
| Default:     | -1      |

**Returns**

intArray

**Example**

```python
msmd.open("my.ms")
# get the scan numbers associated with the specified time range (all obsids, all array ids)
```
scans = msmd.scansfortimes(4.84282937e+09, 20)
msmd.done()
msmetadata.schedule.html

**msmetadata.schedule - Function**

1.3.2 Get the schedule information for the specified observation ID.

**Description**

Get the schedule information for the specified observation ID.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>obsid</td>
<td>Observation ID.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

stringArray

**Example**

```python
msmd.open("my.ms")
# get the schedule information for observation ID = 2
schedule = msmd.schedule()[2]
msmd.done()
```
msmetadata.sideband.html

**msmetadata.sideband - Function**

Get the sideband for the specified spectral window.

**Description**

Get the sideband for the specified spectral window.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>spw</td>
<td>Spectral window ID.</td>
<td>int</td>
<td>int</td>
</tr>
</tbody>
</table>

**Returns**

int

**Example**

```python
msmd.open("my.ms")
# get sideband for spectral window 2.
sideband = msmd.sideband(2)
msmd.done()
```

912
msmetadata.sourcedirs - Function

Get the values of the DIRECTION column from the SOURCE table.

Description

Get the values of the DIRECTION column from the SOURCE table. Returns a dictionary in which the keys are the associated row numbers, represented as strings, in the SOURCE table. Each value in the returned dictionary is a valid direction measure.

Arguments

Returns

record

Example

msmd.open("my.ms")
# get DIRECTION column values from the SOURCE table
sourcedirs = msmd.sourcedirs()
msmd.done()
# the direction associated with zero-based row number 10
dir10 = sourcedirs["10"]
# convert it to B1950, using the measure interface
dir10_B1950 = me.convert(dir10, "B1950")
msmetadata.sourceidforfield.html

msmetadata.sourceidforfield - Function

1.3.2 Get the source ID from the field table for the specified field ID.

Description

Get the source ID from the field table for the specified field ID.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>Zero-based field ID for which to return the source ID from the field table.</td>
<td>int</td>
<td>-1</td>
</tr>
</tbody>
</table>

Returns

int

Example

msmd.open("my.ms")
# get source ID associated with field ID 2
sourceid = msmd.sourceidforfield(2)
msmd.done()
msmetadata.sourceidsfromsourcetable.html

**msmetadata.sourceidsfromsourcetable - Function**

Get the values of the SOURCE_ID column from the SOURCE table.

**Description**

Get the values of the SOURCE_ID column from the SOURCE table. It is unfortunate that the SOURCE table has a column named SOURCE_ID, because implicitly the "ID" of a row in an MS subtable is generally meant to reflect a row number in that table, but that is not the case for the SOURCE table.

**Arguments**

**Returns**

intArray

**Example**

```plaintext
msmd.open("my.ms")
# get SOURCE_ID column values from the SOURCE table
sourceids = msmd.sourceidsfromsourcetable()
msmd.done()
```
msmetadata.sourcenames.html

msmetadata.sourcenames - Function

Get the values of the SOURCE_NAME column from the SOURCE table.

Description

Get the values of the SOURCE_NAME column from the SOURCE table.

Arguments

Returns

stringArray

Example

msmd.open("my.ms")
# get SOURCE_NAME column values from the SOURCE table
sourcenames = msmd.sourcenames()
msmd.done()
msmetadata.spwsforbaseband.html

**msmetadata.spwsforbaseband - Function**

1.3.2 Get the spws associated with the specified baseband or dictionary that maps baseband to spws.

**Description**

Get the spectral windows associated with the specified baseband or dictionary that maps baseband to spectral windows.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseband</td>
<td>Baseband number. If &lt;0, return a dictionary mapping basebands to spws.</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>sqldmode</td>
<td>If ”include”, include SQLD windows, if ”exclude”, exclude SQLD windows, if ”only”, include only SQLD windows. Case insensitive, minimum match honored.</td>
<td>string</td>
<td>include</td>
</tr>
</tbody>
</table>

**Returns**

variant

**Example**

```python
msmd.open("my.ms")
# get the spectral window IDs associated with all the basebands in this dataset
basebandtospwdict = msmd.spwsforbasebands()
# get an array of spws associated with baseband 2.
spwsforbb2 = msmd.spwsforbasebands(2)
msmd.done()
```
msmetadata.spwfordatadesc.html

**msmetadata.spwfordatadesc - Function**

1.3.2 Get the spectral window ID associated with the specified data description ID.

**Description**

Get the spectral window ID associated with the specified data description ID. If the specified data description ID is negative, an array of spectral window IDs is returned. The indices of that array represent data description IDs.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddid</td>
<td>Zero-based data description ID. A negative number will cause all the spectral window IDs to be returned.</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Returns**

variant

**Example**

```python
msmd.open("my.ms")
# get the spectral window ID associated with data description ID 3
spw = msmd.spwfordatadesc(3)
# get the array of spectral window IDs in the order they appear in the DATA_DESCRIPTION
spws = msmd.spwfordatadesc(-1)
msmd.done()
```
msmetadata.spwsforfield.html

**msmetadata.spwsforfield - Function**

1.3.2 Get an array of the unique spectral window IDs for the specified field.

**Description**

Get an array of the unique spectral window IDs for the specified field.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>Field (case sensitive string or zero-based integer ID) for which to return the spectral window IDs.</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
</tbody>
</table>

**Returns**

intArray

**Example**

```python
msmd.open("my.ms")
# get the spectral window IDs associated with field "Fomalhaut"
spws = msmd.spwsforfield("Fomalhaut")
# get spectral window IDs associated with field ID 2
spws = msmd.spwsforfield(2)
msmd.done()
```
Get a dictionary which maps field IDs to spectral window IDs. The field IDs are keys in the returned dictionary. To access a particular element, one must ensure the key is a string.

Arguments

Returns
record

Example

msmd.open("my.ms")
# get the map of field IDs to spw IDs
field_to_spw_map = msmd.spwsforfields()
spws_for_field_5 = field_to_spw_map[str(5)]
msmd.done()
msmetadata.spwsforintent.html

**msmetadata.spwsforintent - Function**

1.3.2 Get an array of the unique spectral window IDs for the specified intent.

**Description**

Get an array of the unique spectral window IDs for the specified intent. The "*" character matches any number of characters from all character classes.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intent</td>
<td>Intent (case sensitive) for which to return the spectral window IDs.</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

intArray

**Example**

```python
msmd.open("my.ms")
# get the spectral window IDs associated with "MY COOL INTENT"
spws = msmd.spwsforintent("MY COOL INTENT")
# got all the spw IDs associated with all intents which contain 'WVR'
scan_numbers = msmd.spwsforintent("*WVR*")
msmd.done()
msmd.done()
```
msmetadata.spwsfornames.html

**msmetadata.spwsfornames - Function**

1.3.2 Get the IDs of the specified spw names.

**Description**

Get the IDs of the specified spw(s). Returns a dictionary where the keys are the requested spectral window names that are present in the data set and the values are arrays of the spectral window IDs corresponding to the name. If a specified name is not present, a warning message is logged and that name is not included in the returned dictionary. Specifying no names results in a dictionary containing the name to spw ID mapping for the entire data set.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>spwids</th>
<th>Names of the spws for which IDs are needed (string or string array). Unspecified will return all spw names.</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>variant</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
msmd.open("my.ms")
# get the IDs for spws named "CO" and "HCN"
spw_ids = msmd.spwsfornames(["CO", "HCN"])
# get the complete spw name to ID map
spw_names_to_ids = msmd.spwsfornames()
msmd.done()
```
msmetadata.spwsforscan.html

**msmetadata.spwsforscan - Function**

Get an array of the unique spectral window IDs for the specified scan number, observation ID, and array ID.

**Description**

Get an array of the unique spectral window IDs for the specified scan number, observation ID, and array ID.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>Scan number for which to return the spectral window IDs.</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>obsid</td>
<td>Observation ID. A negative value means that all observation IDs should be used.</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>arrayid</td>
<td>Array ID. A negative value means that all array IDs should be used.</td>
<td>int</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Returns**

intArray

**Example**

```python
msmd.open("my.ms")
# get the spectral window IDs associated with scan number 20, all obsids, all arrayids.
spws = msmd.spwsforscan(20)
msmd.done()
```
msmetadata.spwsforscans.html

**msmetadata.spwsforscans - Function**

Get a dictionary of which maps scan number to spectral windows for the specified observation ID and array ID.

**Description**

Get a dictionary of which maps scan number to spectral windows for the specified observation ID and array ID. The keys (scan numbers) will be strings. obsid and arrayid must both be non-negative.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>obsid</td>
<td>Observation ID. Must be non-negative.</td>
<td>allowed: int</td>
</tr>
<tr>
<td>arrayid</td>
<td>Array ID. Must be non-negative.</td>
<td>allowed: int</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
msmd.open("my.ms")
# get the scan - spw map for arrayID 1 and obsID 2
scan_to_spws = msmd.spwsforscans(arrayid=1, obsid=2)
# access the spws associated with scan 2
spws_for_scan2 = scan_to_spws["2"]
msmd.done()
```

924
msmetadata.statesforscan.html

**msmetadata.statesforscan - Function**

Get an array of the unique state IDs for the specified scan number, observation ID, and array ID.

**Description**

Get an array of the unique state IDs for the specified scan number, observation ID, and array ID.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>Scan number for which to return the state IDs.</td>
</tr>
<tr>
<td>obsid</td>
<td>Observation ID. A negative value means that all observation IDs should be used.</td>
</tr>
<tr>
<td>arrayid</td>
<td>Array ID. A negative value means that all array IDs should be used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>allowed</th>
<th>int</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Returns**

intArray

**Example**

```python
msmd.open("my.ms")
# get the state IDs associated with scan number 251, all obsids, all arrayids
states = msmd.statesforscan(251)
msmd.done()
```
msmetadata.statesforscans.html

**msmetadata.statesforscans - Function**

1.3.2 Get a dictionary which maps scan numbers to state IDs for the specified array and observation IDs.

**Description**

Get a dictionary which maps scan numbers to state IDs for the specified array and observation IDs. The returned dictionary will have scan numbers, as strings, as keys.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>obsid</td>
<td>Observation ID, must be nonnegative.</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>arrayid</td>
<td>Array ID, must be nonnegative.</td>
<td>int</td>
<td>0</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
msmd.open("my.ms")
# get the map of scan numbers to state IDs for arrayID=1, obsID=2
scans_to_states = msmd.statesforscans(obsID=2, arrayID=1)
states_for_scan_5 = scans_to_states[\str(5)]
msmd.done()
```

926
msmetadata.summary.html

**msmetadata.summary - Function**

1.3.2 Get dictionary summarizing the MS.

**Description**

Get dictionary summarizing the MS.

**Arguments**

**Returns**

record

**Example**

```python
msmd.open("my.ms")
# get the summary
summary = msmd.summary()
msmd.done()
```
msmetadata.tdmspws.html

**msmetadata.tdmspws - Function**

1.3.2 Get an array of spectral window IDs used for TDM. These are windows that have 64, 128, or 256 channels.

**Description**

Get an array of spectral window IDs used for TDM. These are windows that have 64, 128, or 256 channels.

**Arguments**

**Returns**

intArray

**Example**

```plaintext
msmd.open("my.ms")
# get the spectral window IDs used for TDM.
tdm_spws = msmd.tdmspws()
msmd.done()
```
msmetadata.timerangeforobs.html

**msmetadata.timerangeforobs - Function**

1.3.2 Get the time range for the specified observation ID

**Description**

Get the time range for the specified observation ID. The return value is a dictionary containing keys "begin" and "end". Each of the associated value are dictionaries representing epochs which are valid measure records. The values are taken directly from the OBSERVATION subtable; no half-intervals are added or subtracted.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obsid</td>
<td>Zero-based observation ID for which to get the time range.</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
msmd.open("my.ms")
# get the time range associated with observation ID 3
timerange = msmd.timerangeforobs(3)
msmd.done()
```
msmetadata.timesforfield.html

msmetadata.timesforfield - Function

1.3.2 Get an array of the unique times for the specified field.

Description

Get an array of the unique times for the specified field.

Arguments

| Inputs | | |
|---|---|
| field | Zero-based field ID for which to return the times. |
| | allowed: int |
| | Default: -1 |

Returns
doubleArray

Example

```javascript
msmd.open("my.ms")
# get the times associated with field 3
times = msmd.timesforfields(3)
msmd.done()
```
### msmd.timesforintent - Function

Get an array of the unique times for the specified intent.

#### Description

Get an array of the unique times for the specified intent.

#### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>intent</td>
<td>Intent for which to return the times.</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

#### Returns

doubleArray

#### Example

```python
msmd.open("my.ms")
# get the times associated with intent "myintent"
times = msmd.timesforintent("myintent")
msmd.done()
```
msmetadata.timesforscan.html

**msmetadata.timesforscan** - Function

1.3.2 Get the unique times for the specified scan number, observation ID, and array ID.

**Description**

Get the unique times for the specified scan number, observation ID, and array ID. If perspw=True, the returned data structure is a dictionary that has keys representing zero-based spectral window IDs and values representing the unique values of the TIME column corresponding to the specified scan and that corresponding spectral window ID. If False, an array of unique values from the TIME column for the specified scan is returned; there is no separation into spectral window IDs.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>Scan number for which to return the times.</td>
</tr>
<tr>
<td>obsid</td>
<td>Observation ID. A negative value indicates all observation IDs should be used.</td>
</tr>
<tr>
<td>arrayid</td>
<td>Array ID. A negative value indicates all array IDs should be used.</td>
</tr>
<tr>
<td>perspw</td>
<td>Return output dictionary with keys representing spectral window IDs (True), or an array of all times (False).</td>
</tr>
</tbody>
</table>

**Returned**

anyvariant

**Example**

932
msmd.open("my.ms")
# get the times associated with scan number 10, all obsids, all arrayids.
times = msmd.timesforscans(10)
msmd.done()
msmetadata.timesforscans.html

msmetadata.timesforscans - Function

1.3.2 Get an array of the unique times for the specified scan numbers, observation ID, and array ID.

Description

Get an array of the unique times for the specified scan numbers, observation ID, and array ID.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>scans</td>
<td>Scan numbers for which to return the times.</td>
<td>intArray</td>
<td>-1</td>
</tr>
<tr>
<td>obsid</td>
<td>Observation ID. A negative value indicates all observation IDs should be used.</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>arrayid</td>
<td>Array ID. A negative value indicates all array IDs should be used.</td>
<td>int</td>
<td>-1</td>
</tr>
</tbody>
</table>

Returns
doubleArray

Example

```python
msmd.open("my.ms")
# get the times associated with scan numbers 10 and 20, all obsids, all arrayids
times = msmd.timesforscans([10,20])
msmd.done()
```
msmetadata.transitions - Function

Get the spectral transitions from the SOURCE table for the specified source and spectral window.

Description

Get the spectral transitions from the SOURCE table for the specified source and spectral window. The return value will be an array of transitions if the transitions are defined for the specified inputs, or False if they do not.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceid</td>
<td>Zero-based source ID (from the SOURCE::SOURCE_ID column).</td>
</tr>
<tr>
<td>spw</td>
<td>Zero-based spectral window ID.</td>
</tr>
</tbody>
</table>

| allowed: | int |
| Default: | 0 |

Returns

anyvariant

Example

```python
msmd.open("my.ms")
# get the transitions for source ID 2 and spw ID 20
reffreq = msmd.transitions(2, 20)
msmd.done()
```
msmetadata.wvrspws.html

**msmetadata.wvrspws - Function**

1.3.2 Get an array of spectral window IDs used for WVR. These are windows that have 4 channels.

### Description

Get an array of spectral window IDs used for WVR. These are windows that have 4 channels. If complement is True, return the complement set instead (all non-wvr spw IDs).

### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>complement</td>
<td>If True, return all non-wvr spws.</td>
</tr>
<tr>
<td>allowed</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>-false</td>
</tr>
</tbody>
</table>

### Returns

intArray

### Example

```python
msmd.open("my.ms")
# get the spectral window IDs used for WVR.
wvr_spws = msmd.wvrspws()
msmd.done()
```
1.3.3 msplot - Tool

Plot data from a measurement set

Requirements:

Synopsis

Description

The msplot tool is a plotting tool for a measurement sets. The functionality of the msplot tool extends that of the tableplot tool to add knowledgeable about measurement sets. The msplot tool does for measurement sets what the tableplot tool does for tables. Much of the functionality is similar to that of tableplot tool and it may be useful to read the tableplot tool documentation.

There is also a similar plotting tool for calibration data, calplot tool. The calplot tool documentation may be useful to read since the calplot tool is similar to msplot tool.

Overview of msplot tool functionality

At present, the msplot tool plots from a single measurement sets only. Eventually data from more than one measurement set will be able to be accessed and plotted at the same time. Plots from the same MeasurementSet can be overlayed and more then one plotting panel can be created so different plots can be viewed simultaneously.

• Opening and Closing - Before doing any plots you must call mp.open with the measurement set to be plotted. When finished use mp.reset to reset the measurement set; clear any data selection done with mp.setdata as well as resets all of the plotting options back their default values. mp.done to close the measurement set, and the MS plotter.

• Plotting - The msplot tool provides several common plots that can be called easily, but the mp.plotxy function is a generic function for plotting. The available common plots are: mp.array, mp.azimuth, mp.baseline, mp.elevation, mp.hourangle, mp.parallacticangle, mp.uvcoverage, mp.uvdist, mp.vischannel, and mp.vistime.

There are a number of plotting options that can be set to change the color, labels, symbol, number of plotting panels, and many others. The mp.plotoptions function controls the plot options. It is important to note that once a plot option is set, it remains set until a subsequent call to the plotoptions method turns it off.
In addition iterative plots are provided. Iterative plots allow a user to iterate over some column plotting for that particular item. For example, a user may wish to plot the uv distances for each antenna separately by using the iteration value of 'ANTENNA' with the mp.dist function.

All of methods listed above have an interation parameter. Valid values for this parameter are: BASELINE, ANTENNA, FIELD, SPW, SCAN, FEED, and ARRAY_ID. To plot the the next plots in the iteration use the mp.iterplotnext. To stop the iterative plot use mp.iterplotstop. Note that a useful operation to perform before doing and interactive plot is to set the subplot option to plot multiple plots with the mp.plotoptions. function. For example, subplot=131 will yeild a plot of three rows of plots.

• **Data Selection** - The mp.setdata function can be used to select which data is to be viewed, more precisely antennas, fields, UV ranges, and time selections can specified.

• **(Un)Flagging** - (Un)Flagging can be performed both in memory and on disk. Flagging has recently undergone some refactoring and it is now possible to save what has been flagged in stages (versions). This gives msplot the ability to undo flagging if so desired. Also is a new button on the plotting window that turns on the flagging mode, allowing users to interactively select the areas of interest. Soon there will be other buttons for (un)flagging, and for displaying information about the flagged regions. To (un)flag data first make a plot, see the plotting section above. The next step is to mark a region(s) on the plot. This is done with the mp.markregion function. If a specific region is given to this function this region is used, otherwise the regions can be marked on the plotter by selecting square regions with the mouse.

Now there are regions marked on the plot, there are three different actions that can be taken at this point. The mp.locatedata function displays information about each of the points in the marked regions. (Warning: if a lot of points are selected a large amount of data will be given to the logger, this can slow down the logger drastically!) The other two actions that can be taken are closely related mp.flagdata and mp.unflagdata, to flag or unflag data respectively. Both of these functions allow the flagging to be done in memory (default) or on disk.

One final function for flagging is mp.clearflags. **Warning: This function clears all flags, all data will be unflagged with this method.**

**Methods**

- **msplot**
  - Construct a msplot tool for plotting measurement sets
- **open**
  - Set the measurement set to be plotted.
clearplot  Clear the plotting window or a particular panel, or all panels.
emperorsNewClose  Like the Emperor's New Clothes.
reset  Reset the state of MS plot back to its default state.
closeMS  Close the measurement set being used.
close  See done – close and done do the same thing.
done  Close the current MeasurementSet, and destroy the plotter – ending all plotting.
plotoptions  Set the style of the plot.
summary  List a short summary, description, of the data in the open measurement set.
setdata  Select a subset of the measurement set to operate on.
extendflag  Set the scope of flagging extension.
avedata  Specify which data is to be averaged in the MS (or selected MS).

plotxy  A generic plotting routine for Measurement sets.
checkplotxy  Routine for checking the sanity of a plotxy plot.
itplotstart  Plot the first set of iterative plots.
itplotnext  Continue plotting on an iteration axes.
itplotstop  Stop an iterative plot.
savefig  Save the currently plotted image.
markregion  Mark a rectangular region to flag or to investigate the data in the area.
flagdata  Set flags for all selected regions marked using mp.markregion()
unflagdata  Unset flags in all regions marked using mp.markregion() Similar to the mp.flagdata()
clearflags  Clear all flags in the table. Note: This clears *all* flags and should be used with caution.
locatedata  Print info about data selected using mp.markregion().
saveflagversion  Save current flags, applied to the current measurement set with a version name.
restoreflagversion  Restore flags for the current Measurement Set.
deleteflagversion  For the current measurement set delete a saved flag_version.
getflagversionlist  Print out a list of saved flag_versions, for the current Measurement Set.
msplot.msplot.html

msplot.msplot - Function

Construct a msplot tool for plotting measurement sets

Description

Create a msplot tool object.
This is the most commonly used constructor. It creates an msplot tool which is associated with a particular measurement set, and a tableplot tool that is used to do the plotting.

Arguments

Returns

msplotobject
msplot.open.html

msplot.open - Function

1.3.3 Set the measurement set to be plotted.

Description

Set the measurement set to be plotted. This method must be invoked before any of the other msplot tool functions.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>msname</td>
<td>measurement set name, including path</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>dovel</td>
<td>whether to calculate velocity or not</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>restfreq</td>
<td>a rest frequency quanta or transition name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>frame</td>
<td>frequency frame for spectral axis</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>doppler</td>
<td>doppler mode</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool

Example

```bash
# Open a msplot tool with a measurement set
mp.open( msname='./data/3C273XC1.ms' );
```
msplot.clearplot.html

**msplot.clearplot - Function**

Clear the plotting window or a particular panel, or all panels.

**Description**

Clear the plotting window. Either clear the whole window (default) or a particular panel (specified by the subplot parameter).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>subplot</td>
<td>Three (or four) digits number: first digit for nrows, second for ncols, the rest for panel number.</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>000</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
# open a MS dataset, Plot the array on the left and uvcoverage on the right.
# Also set the X and Y axes labels, and the title,
mp.open( msname='ngc5921.ms');
mp.plotoptions( subplot=121 )
mp.array()
mp.plotoptions( subplot=122 )
mp.uvcoverage()

# Now clear the uvcoverage plot area and plot the uvdist instead.
mp.clearplot( subplot=122 )
mp.uvdist()

# Now clear all plots
```
mp.clearplot()
msplot.emperorsNewClose.html

**msplot.emperorsNewClose - Function**

1.3.3 Like the Emperor’s New Clothes.

**Description**

The mp.close() method has been a much contested method. With user’s not really wanting to close things, but pretend to. So just to make all happy we have a much anticipated new close method, sure to bedazzle and shine!

Note that you may find the mp.reset function very useful for controlling the state of the msplot tool.

**Arguments**

**Inputs**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
</table>

**Returns**

bool
msplot.reset.html

**msplot.reset - Function**

Reset the state of MS plot back to its default state.

**Description**

Reset the state of MS plot back to its default state. Calling this function will cause all of the plot options to be reset to their default values, and to reset any data selection performed by the mp.setdata().

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
</table>

**Returns**

bool
msplot.closeMS.html

**msplot.closeMS - Function**

1.3.3 Close the measurement set being used.

**Description**

Close the measurement set being used. As a side affect any data selections (via mp.setdata or mp.sespectral) are reset to their initial state, and the plot options (set via mp.plotoptions) are also set to their initial stat.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
</table>

**Returns**

bool
msplot.close.html

**msplot.close - Function**

*1.3.3* See done – close and done do the same thing.

**Description**

See done

**Arguments**

**Returns**

bool
msplot.done.html

**msplot.done - Function**

1.3.3 Close the current MeasurementSet, and destroy the plotter – ending all plotting.

**Description**

End the msplot tool

**Arguments**

**Returns**

bool
msplot.plotoptions - Function

1.3.3 Set the style of the plot.

Description

Set the style of the plot. This function allows the plot title, axis labels, font, plotting color, plot symbol and many, other aspects of the plot to be set by the user, giving the user much flexibility over the look of their plots.

The various aspects that can be controlled by the user are as follows:

- **Labels:** The title that appears at the top of the plot is controlled by the parameter. The label along the x-axis and y-axis are set using the xlabel and ylabel parameters. If no values are not provided by the user then these labels are constructed from the data selected to be plotted.

  To control the font size of the labels use the font parameter. The x-axis and y-axis labels are always set to be 2pts. smaller than the title, which is set at the given font size specified.

- **Size Controls:** To control the size of the window, the windowsize, and aspectratio options can be set.

  To control the range of points plotted use the plotrange option. This plot option accepts either time strings in the form YYYY/MM/DD/hh:mm:ss or real values.

- **Data Point Styles:** There are a number of plotoptions for controlling the color and style of the points plotted.

  The plotsymbol option set both the color and/or shape of the points plotted. It accepts the same syntax as that used by the pylab plot function. There are six different colors used: 'k' black, 'r'ed, 'g'reen, 'b'lue, 'c'yan, 'y'ellow, wnd 'w'hite. The plotsymbols include, but are not limited to '+', 'o', '-', and '–'. For a full list see the matplotlib documentation.

  The markersize and linewidth options control how big the plot symbols/lines are.

  The multicolor plot option is unique to CASA. It is used to specify whether or not different channels/correlations are plotted in different colors. The colors used can not be set by any of the plot options. Basically, when plotting the colors are cycled through, changing whenever a different channel or correlation is encountered.
For large measurement sets it may be useful to plot only a portion of the measurement set. The `skiprows` allows every nth row to be plotted rather than all of the plots.

- **Multiple plots**: One of the more useful abilities of the CASA plotters is the ability to plot several plots simultaneously, either side-by-side, and/or one on top of the other.

The `subplot` option determines the number of panels to create, each panel contains a plot of some data. Although the `subplot` option is a single integer, it is really treated as three separate integers:

- `nrows`: number of rows of panels,
- `ncols`: number of columns of panels, and
- `panel`: the panel number, which panel to plot on.

For example, `subplot=132` specifies that there are three panels side-by-side (three columns of panels), and that we are plotting on the second panel. The top, left corner panel is panel number 1, and the panel number increases to the right first, continuing on the next row when the end of a row is reached. The example section shows some examples that create a number of different panel arrangements.

Related to the multi-panel plots is the `removeoldpanels` option. This option when set to `True`, the default value, mimics the native `matplotlib` behaviour, clearing any panels that lie partially or completely under a new panel being plotted. If it is set to `False` new panels could potentially plot over top of old panels depending on the `subplot` values.

In addition to having plots there is the ability to overplot. The `overplot` option when set to true will, instead of clearing the panel that is currently being plotted on, plot over what is already there. When `overplot` is set to true the next plot is plotted over top of what is already there, and if the `plotsymbol` has not been specified the `msplot` tool will automatically pick a different color to plot with. An example where this may be useful if plotting different spectral windows separately, but on the same plot.

Related to overplotting is the `replacetopplot` and `showflags` option. By default `replacetopplot` is set to `False`, but if it is set to `True` then only the last plot on a panel is replaced. This option comes in handy when a mistake has been made in the last plot. If the `showflags` option is set to `True` then the flagged data is plotted. A nice feature is the ability to plot the flagged and unflagged data on the same plot, by doing the same plot with both the `overplot` and `showflags` plot turned on. Note that flagged data is always plotted in a magenta color.

**Arguments**

950
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>subplot</td>
<td>Three (or four) digits number: first digit for nx, second for ny, the rest for pannel number.</td>
<td>int</td>
<td>111</td>
</tr>
<tr>
<td>plotsymbol</td>
<td>String specifying the colour to plot in, as well as the symbol to plot. This argument takes the same values as pylab plot command. Some of the valid symbols are: 'o', '+', ...</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>plotcolor</td>
<td>String specifying the colour to plot in. This can be one of the predefined pylab colour names. This over-rides the colour specified in plotsymbol.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>multicolor</td>
<td>chan: means different channels in different colours, corr: means different correlations in different colours, both: means different correlations and channels in multicolour none: plot everything the same color.</td>
<td>string</td>
<td>none</td>
</tr>
<tr>
<td>plotrange</td>
<td>Plot data within the specified range of values will be plotted. The range of values is given as a string in the form [xmin, xmax, ymin, ymax]. For most plots the xmin/max and ymin/max values are expected to be numeric values. However, for time plots, ie. where one or more of the axis is time, the xmin/max and ymin/max are expected to be strings in the from YYYY/MM/DD/hh:mm:ss.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>timeplot</td>
<td>Indicate if the data is to be interpreted as time time values. Valid values are 'o'ff, 'x'-axis, 'y'-axis, 'b'oth axes.</td>
<td>char</td>
<td>'o'</td>
</tr>
<tr>
<td>markersize</td>
<td>Specify the size (in pixels) of the markers being plotted. Markers are specified with the plotsymbol option.</td>
<td>double</td>
<td>8.0</td>
</tr>
<tr>
<td>linewidth</td>
<td>Occasionally lines, rather then points, are plotted. This option allows the width of the plotted lines to be specified in points (pixels).</td>
<td>double</td>
<td>1.0</td>
</tr>
<tr>
<td>overplot</td>
<td>To do overplot or not.</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>replacetopplot</td>
<td>true : when overplot=false, replace the top-most layer only false : overplot=false always creates a fresh stack of plots.</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>removeoldpanels</td>
<td>true : mimic the native matplotlib behaviour of clearing up plots that lie partially or completely underneath</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>
Returns
bool

Example

# open a MS dataset, set the plot options.
# Also set the X and Y axes labels, and the title,
mp.open( mspath='data/3C273XC1.ms');
labels := ['Amplitude vs UVdist', 'uvdist', 'amplitude'];
mp.pyplotoptions( windowsize=5, aspectratio=0.8, fontsize=14.0, 
xlabel='uvdist', ylabel='amplitude', title='Amplitude vs. UVdist',
plotsymbol='g+');

# Create 3 panels for plotting, and starting an iterative plot. Three
# plots will be plotted for each iteration of the plot.
mp.open( './data/ngc5921.ms')
mp.pyplotoptions( subplot=311 )
mp.vischannel( column='data', what='amp', iteration='baseline' )

# Create 3 panels for plotting, but they are different sizes.
# Two panels at the top, smaller with the array and uvcoverage plots.
# A single wider panel at the bottom (the whole second row) containing
# the uvdistance plot. The uvdistance plot, plots the corrected data
# overtop of the actual data.
mp.open( './data/ngc5921.ms');
mp.pyplotoptions( subplot=221 );
mp.array();
mp.pyplotoptions( subplot=222 );
mp.viscoverage();
mp.pyplotoptions( subplot=212 );
mp.visdistance();
mp.pyplotoptions( overplot=1, plotcolor=3);
mp.visdistance( column='corrected_data');

# Plot the flagged and unflagged data on the same plot, plotting the
# visibility amplitude vs. the channel.
mp.open(ngc5921PATH);
mp.vischannel();
mp.pyplotoptions( showflags=1, overplot=1 );
mp.vischannel();
msplot.summary.html

**msplot.summary - Function**

1.3.3 List a short summary, description, of the data in the open measurement set.

**Description**

List a summary, description, of the selected data in the open measurement set. The information that is displayed includes:
- antenna names
- field names
- scan numbers
- spectral window list, including the number of channels for each one.
- correlations

Eventually the summary will include the time range and uv distance range as well.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>selected</td>
<td>Determine if we print a summary of the selected (true) data or a summary of the full measurement set (false).</td>
</tr>
<tr>
<td>allowed: bool</td>
<td>Default: true</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
# create a msplot tool and set the subset data for plotting.
mp.open( mname='./data/3C273XC1.ms');

# View a summary of the whole measurement set.
mp.summary( selected=false);

# View what we've selected.
mp.setdata( "spw=3~7", antenna="0~200" );
mp.summary( selected=true );
```

954
msplot.setdata.html

**msplot.setdata - Function**

1.3.3 Select a subset of the measurement set to operate on.

**Description**

Select a subset of the measurement set. All plots will operate on this subset of the measurement set based on the values given.

All of the *Index* fields expect lists of integers. All of the *Name* fields accept strings. Where both indices and expressions are allowed as inputs (antenna’s for example), they will be combined together when selecting the data.

The expression strings contain values separated by ‘,’. The wildcard character ‘*’ can be used with the names. The ‘>’ and ‘<’ characters can be used to indicate values that are greater than, or less than (respectively) a particular value.

Note: that integer values in the antennaNames list will be interpreted as indices. The ‘∼’ indicates a range for values.

Spectral windows *names* are bit of a special case as channel information can be specified too. Spectral windows are in the following format: (spwlist):(channellist) where the spectral window and channel list follow the expression conventions listed above. "RR,LL,RL" "[RR LL RR]" "(XX)" where different types of polarizations are separated by a space or a comma.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline</td>
<td>Baseline selection expression.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>field</td>
<td>Field selection string.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>scan</td>
<td>Scan selection string.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>uvrange</td>
<td>UV Range selection string.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>array</td>
<td>Array selection string.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>feed</td>
<td>Feed selection string.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>spw</td>
<td>Spectral Window selection string.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>correlation</td>
<td>Correlation selection string.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>Time selection values are in the form: YYYYMMDDhh:mm:ss∼YYYYMMDDhh:mm:ssstep where the first or second term may be dropped, and more then one range may be specified. Ranges are separated by a &quot;,&quot;. The step value is a real value representing the number of seconds to skip or average, depending on the value given.</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```bash
# create a msplot tool and set the subset data for plotting.
```
mp.open( msnamex=./data/3C273XC1.ms);

# Select all antenna's that begin with VLA or N, field 1,2 and 3,  
# 'RR' and 'LL' correlations, and spectral windows 3,4, and 5. 
mp.setdata( antennaNames='VLA:N*', fieldNames='1~3', correlations='RR, LL',
          spwIndex=[3,4,5] );

# Select all fields, LR correlations, uvdists greater than 125 kilolambda,  
# and times of June 27, 1989 at 3:31:40. 
mp.setdata( fieldNames='*', correlations='LR', uvDists='>125kl' \ 
           times='1989/06/27/03:31:40' );
msplot.extendflag - Function

1.3.3 Set the scope of flagging extension

Description

Set the scope of flagging extension

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>extendcorr</td>
<td>Indicate correlation based flagging extension. Valid values are: allhalf</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>extendchan</td>
<td>Indicate channel based flagging extension. Valid values are: all</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>extendspw</td>
<td>Indicate spectral window based flagging extension. Valid values are: all</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>extendant</td>
<td>Indicate antenna (baselines) based flagging extension. Valid values are: all</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>extendtime</td>
<td>Indicate time based flagging extension. Valid values are: all</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

bool

Example

959
# create a msplot tool and set the subset data for plotting.
mp.open( msname='./data/3C273XC1.ms');

# TO COME
mp.average( ??? )
msplot.avedata.html

**msplot.avedata - Function**

1.3.3 Specify which data is to be averaged in the MS (or selected MS).

**Description**

No description here at the moment. This method is under active development and is changing frequently. Documentation will be provided when development settles down.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>chanavemode</strong></td>
<td>Indicate what if any averaging should be done on the selected channels. The channel selection and averaging is done via the spw parameter. Valid values are: none: No averaging, default value step: Plot every nth point scalarstep: incoherent average of every n points vectorstep: coherent average of every n points scalarchunk: incoherent average of blocks with n points vectorchunk: coherent average of blocks with n points allowed: string Default: none</td>
</tr>
<tr>
<td><strong>corravemode</strong></td>
<td>Indicate what if any averaging should be done on the selected correlations. See chanavemode for a detailed description of the valid values. Valid values are: none, step, scalarstep, vectorstep, scalarchunk, or vectorchunk allowed: string Default: none</td>
</tr>
<tr>
<td><strong>datacolumn</strong></td>
<td>Indicate the visibility data to be averaged. Valid values are: DATA CORRECTEDDATA MODELDATA allowed: string Default: DATA</td>
</tr>
<tr>
<td><strong>averagemode</strong></td>
<td>Indicate the mode for channel and/or time averaging. Valid values are: vector scalar allowed: string Default: vector</td>
</tr>
<tr>
<td><strong>averagechan</strong></td>
<td>Indicate the number of channels to average. The default value of 1 means no channel averaging. allowed: string Default: 1</td>
</tr>
<tr>
<td><strong>averagetime</strong></td>
<td>Indicate the length of time interval to average. Valid values are double values of time in seconds. The default value 0 means no time averaging. allowed: string Default: 0</td>
</tr>
<tr>
<td><strong>averageflagged</strong></td>
<td>Indicate either flagged or unflagged data to average. allowed: bool Default: false</td>
</tr>
<tr>
<td><strong>averagescan</strong></td>
<td>Indicate whether time averaging cross scan boundaries. allowed: bool Default: false</td>
</tr>
<tr>
<td><strong>averagebl</strong></td>
<td>Indicate whether averaging cross baseline boundaries. allowed: bool Default: false</td>
</tr>
<tr>
<td><strong>averagearray</strong></td>
<td>Indicate whether averaging cross array boundaries. allowed: bool Default: false</td>
</tr>
<tr>
<td><strong>averagechanid</strong></td>
<td>Indicate whether using averaged channel id or not. allowed: bool Default: false</td>
</tr>
<tr>
<td><strong>averagevel</strong></td>
<td>Indicate whether calculating averaged velocity id or not. allowed: bool Default: false</td>
</tr>
</tbody>
</table>
Returns

bool

Example

# create a msplot tool and set the subset data for plotting.
mp.open( msname='./data/3C273XC1.ms');

# TO COME
mp.average( ??? )
msplot.plot.html

msplot.plot - Function

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>The type of plot to do. Valid values are strings for each of the msplot function, the valid strings are: &quot;array&quot; &quot;azimuth&quot;, &quot;baseline&quot;, &quot;elevation&quot;, &quot;hourangle&quot;, &quot;parallacticangle&quot;, &quot;uvcoverage&quot;, &quot;uvdist&quot;, &quot;vischannel&quot;, and &quot;vistime&quot;</td>
<td>string</td>
<td>allowed: string</td>
</tr>
<tr>
<td>column</td>
<td>Column name in main table of measurement set to plot. Valid values are: data, corrected, model, residual, and weight.</td>
<td>string</td>
<td>allowed: string</td>
</tr>
<tr>
<td>value</td>
<td>String: amp, phase</td>
<td>string</td>
<td>allowed: string</td>
</tr>
<tr>
<td>iteration</td>
<td>List of strings: Antenna1, Antenna2, Feed1, Feed2, Field_id, Scan_number, and Time. Spectral Window/Polarization_id( not available yet )</td>
<td>stringArray</td>
<td>allowed: stringArray</td>
</tr>
</tbody>
</table>

Returns

bool

Example

```python
# create a msplot tool.
mp.open( msname='./data/ngc7538.ms');

# Plot the antenna distribution.
mp.plot( type='array' );
```

964
# Plot the uv distances with the corrected data column plotted
# over the data column
mp.plotoptions( overplot=False, plotcolor='chocolate' );
mp.plot( type='uvdist', value='phase' );
mp.plotoptions( overplot=True, plotcolor='lemonchiffon' );
mp.plot( type='uvdist', column='corrected', value='phase' );

# Do an iterative plot on baselines, on the visibility amplitude/channel
# plot. We display 6 plots at a time.
mp.clearplot();
mp.plotoptions( subplot=231 );
mp.plot( type='vischannel', column='data', value='amp', iteration='baseline' );
mp.iterplotnext();
mp.iterplotnext();
mp.iterplotstop();

965
msplot.checkplot - Function

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plottype</td>
<td>The type of plot to check. Valid values are strings for each of the msplot function, the valid strings are: &quot;array&quot;, &quot;azimuth&quot;, &quot;baseline&quot;, &quot;elevation&quot;, &quot;hourangle&quot;, &quot;parallacticangle&quot;, &quot;uvcoverage&quot;, &quot;uvdist&quot;, &quot;vischannel&quot;, &quot;visfrequency&quot;, &quot;vistime&quot;, and &quot;visvelocity&quot;. allowed: string. Default:</td>
</tr>
<tr>
<td>column</td>
<td>Column name in main table of measurement set to plot. Valid values are: data, corrected, model, residual, and weight. allowed: string. Default: data.</td>
</tr>
<tr>
<td>iteration</td>
<td>List of strings: Antenna1, Antenna2, Feed1, Feed2, Field_id, Scan_number, and Time. Spectral Window/Polarization_id (not available yet). allowed: stringArray. Default:</td>
</tr>
</tbody>
</table>

Returns

bool

Example

# create a msplot tool.
mp.open( msname='./data/ngc7538.ms');

# select the data: spectral windows 0 and 1, channels 3 through 5 # and RL correlations
mp.setdata( spwNames=['(0,1):[3-5]'], correlations=['RL']);
mp.checkplot( plottype='uvdist', column='data', value='amp');
msplot.plotxy - Function

A generic plotting routine for Measurement sets.

Description

Plot X versus Y for all meaningful columns in the MAIN table of a MS and derived quantities.

1. X and Y may be one of the followings:
   - Antenna1, Antenna2, Feed1, Feed2, Field_id,
   - ifr_number( not available yet ), Scan_number,
   - Time, channel( not available yet ), uvdistance,
   - frequency(not available yet ), u, v, w,
   - weight, data, model, corrected, residual
   ( derived quantities will be listed later).

2. iteration axis may be one of the followings:
   - Antenna1, Antenna2, Feed1, Feed2, Field_id, Scan_number,
   - Time, Spectral Window/Polarization_id( not available yet )

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-axis, a column name in measurement sets’ main table. Valid values are: antenna1, antenna2, azimuth, baseline, channel, corrected, data, elevation, feed1, feed2, field_id, frequency, hourangle, ifr_number, model, parallactic_angle, residual, scan_number, time, u, uvdist, v, velocity, w, and weight. allowed: string Default: uvdist</td>
</tr>
<tr>
<td>y</td>
<td>Y-axis, a column name in the measurement sets’ main table. Valid values are the same as the X-axis values. allowed: string Default: data</td>
</tr>
<tr>
<td>xcolumn</td>
<td>The column name in the measurement sets’ msin table. Valid values are: data, corrected, model, and residual. allowed: string Default: data</td>
</tr>
<tr>
<td>ycolumn</td>
<td>The column name in the measurement sets’ msin table. Valid values are: data, corrected, model, and residual. allowed: string Default: data</td>
</tr>
<tr>
<td>xvalue</td>
<td>String, needed if X or Y is data quantity: amp, phase, real, imag allowed: string Default: amp</td>
</tr>
<tr>
<td>yvalue</td>
<td>String, needed if X or Y is data quantity: amp, phase, real, imag allowed: string Default: amp</td>
</tr>
<tr>
<td>iteration</td>
<td>List of strings: Antenna1, Antenna2, Feed1, Feed2, Field_id, Scan_number, and Time. Ex: iteration over baselines : ['antenna1','antenna2'] Spectral Window/Polarization_id( not available yet ) allowed: stringArray Default:</td>
</tr>
</tbody>
</table>

**Returns**
bool

**Example**
# create a mplot tool.
mp.open( msname='./data/ngc7538.ms');

# select data. correlations are separated by a space:
#   correlations=['RR RL']
mp.setdata( spwNames=['(0,1):[3-5]'], correlations=['RL'] );

# Do an iterative plot over baselines, plotting uvdist vs. data for each antenna.
mp.plotxy( X='uvdist', Y='', column='data', iteration=['antenna1', 'antenna2']);
msplot.checkplotxy.html

**msplot.checkplotxy - Function**

1.3.3 Routine for checking the sanity of a plotxy plot.

**Description**

Do a sanity checks of all the inputs that have been given. This includes checking the plot option values (mp.plotoptions()), data selection (mp.setdata()), and spectral selections (mp.setspectral()). Also included in the sanity checks is a check to see how many points will be plotted. If there are millions of points to be plotted, but do not want to wait for a large plot try setting the skiprows or averagenrows plot options.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X-axis, a column name in measurement sets’ main table. Valid values are: antenna1, antenna2, azimuth, baseline, channel, corrected, data, elevation, feed1, feed2, field_id, frequency, hourangle, ifr_number, model, parallactic_angle, residual, scan_number, time, u, uvdist, v, velocity, w, and weight. Allowed: string Default: uvdist</td>
</tr>
<tr>
<td>y</td>
<td>Y-axis, a column name in the measurement sets’ main table. Valid values are the same as the X-axis values. Allowed: string Default: data</td>
</tr>
<tr>
<td>xcolumn</td>
<td>The column name in the measurement sets’ msoin table. Valid values are: data, corrected, model, and residual. Allowed: string Default: data</td>
</tr>
<tr>
<td>ycolumn</td>
<td>The column name in the measurement sets’ msoin table. Valid values are: data, corrected, model, and residual. Allowed: string Default: data</td>
</tr>
<tr>
<td>xvalue</td>
<td>String, needed if X or Y is data quantity: amp, phase, real, and imag Allowed: string Default: amp</td>
</tr>
<tr>
<td>yvalue</td>
<td>String, needed if X or Y is data quantity: amp, phase, real, and imag Allowed: string Default: amp</td>
</tr>
<tr>
<td>iteration</td>
<td>List of strings: Antenna1, Antenna2, Feed1, Feed2, Field_id, Scan_number, and Time. Spectral Window/Polarization_id (not available yet) Allowed: stringArray Default:</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
# create a msopt tool, do some data selections and
```
# plot option selection and check the validity of them.
mp.open(msname='./data/3C273XC1.ms');
mp.setdata(antennaNames='VLA:N*', fieldNames='1~3', correlations='RR, LL,'
spwIndex=[3,4,5]);
mp.plotoptions(windowsize=5,aspectratio=0.8, fontsize=14.0,
xlabel='uvdist', ylabel='amplitude', title='Amplitude vs. UVdist',
plotsymbol='g+');
mp.checkplotxy(X='uvdist', Y='', column='data', iteration=['antenna1', 'antenna2']);
msplot.iterplotstart - Function

Plot the first set of iterative plots.

Description

Begin a series of plots using subtables constructed via an iteration axes, which is set in either the mp.plot or mp.plotxy methods. Use iterplotnext() to step through. Multi-panel plots as well as overplots are supported with this function. Overplots have a restriction in that both plots must have the same axes.

Only forward step through is allowed.

Arguments

Inputs

Returns
bool

Example

# create a msplot tool, select the data with field name 3C273 for plot,
# and initialize a plot of Amplitude vs UV distance for
# channel 1 and stokes 1, iterating over Antenna1, and creating
# two plot panels per iteration page.
mp.open( msname=['./data/3C273XC1.ms']);
plotoptions.nxpanels := 1;
plotoptions.nypanels := 2;
plotoptions.windowsize := 6;
plotoptions.aspectratio := 1.2;
plotoptions.fontsize := 14.0;
mp.setdata( fieldNames=['3C273'] );
labels := ['Amplitude vs UVdist (iterating over Antenna1)','uvdist','amplitude'];
mp.plotxy( x='SQRT(SUMSQUARE(UVW[1:2]))' y='AMPLITUDE(DATA[1,1])', iteration='ANTENNA1' );
mp.iterplotstart();
To iterate over baseline, for stokes 1, channel 1.
plotopts.nxpanels := 1;
plotopts.nypanels := 4;
labels := ['Amplitude vs UVdist (iterating over Baseline)', 'uvdist', 'amplitude'];
iteraxes := ['ANTENNA1', 'ANTENNA2'];
mp.plotxy( x=['SQRT(SUMSQUARE(UVW[1:2]))', y='AMPLITUDE(DATA[1,1])'], iteration=iteraxes )
mp.iterplotstart();
msplot.iterplotnext - Function

1.3.3 Continue plotting on an iteration axes.

Description

Start/Continue plotting by stepping through the iteration axes.

Arguments

Returns

bool

Example

# Iterate through the data, plotting the uvdist for each antenna1
# The same plot can be achieved with mp.uvdist( iteration='antenna1' )
mp.open( msname=['./data/3C273XC1.ms']);
mp.plotoptions( subplot=121, windowsize=6, aspectratio=1.2, fontsize=14.0);
mp.setdata( uvDists=['>25k1']);
mp.plotxy(X='uvdist',Y='',iteration='antenna1');
mp.iterplotnext();
mp.iterplotnext();
mp.iterplotstop();
msplot.iterplotstop - Function

Stop an iterative plot.

Description

To be called at the end of the plot iterations, or in between if desired.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rmplotter</td>
<td>Indicates of the plot window should be removed (true) from the display or left (false)</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>

Returns
bool

Example

# see the example for iterplotnext()
msplot.savefig.html

**msplot.savefig - Function**

1.3.3 Save the currently plotted image.

**Description**

Store the contents of the plot window in a file. The file format (type) is based on the file name, i.e., the file extension given determines the format the file is saved as. The accepted formats are eps, ps, png, pdf, and svg. Internally, this function uses the matplotlib pl.savefig function. Note that if a full path is not given that the files will be saved in the current working directory.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>Name the plot image is to be saved to.</td>
</tr>
<tr>
<td>dpi</td>
<td>Number of dots per inch (resolution) to save the image at.</td>
</tr>
<tr>
<td>orientation</td>
<td>Either landscape or portrait. Supported by the postscript format only.</td>
</tr>
<tr>
<td>papertype</td>
<td>Valid values are: letter, legal, exective, ledger, a0-a10 and b0-b10. This option is supported by the postscript format only.</td>
</tr>
<tr>
<td>facecolor</td>
<td>Color of space between the plot and the edge of the square. Valid values are the same as those accepted by the plotcolor option.</td>
</tr>
<tr>
<td>edgecolor</td>
<td>Color of the outer edge. Valid values are the same as those accepted by the plotcolor option.</td>
</tr>
</tbody>
</table>
Returns
bool

Example

# Open a msplot tool with a measurement set
mp.open( msname='./data/3C273XC1.ms' );
# Plot something and save it in a pdf file.
mp.plot( 'uvdist' )
mp.savefig( 'uvdist.pdf', edgecolor='black' )
Mark a rectangular region to flag or to investigate the data in the area.

Description

Mark a rectangular region on the plot. Each call to `markflag` allows one region to be drawn. Any number of successive calls can be made. This function marks and stores a list of marked regions. These regions can then be (un)flagged, or information about the marked data can be retrieved. To flag the data the `(un)flagdata` function must be used and to find out information about the data the `locatedata` function must be used.

In the case of multi-panel plots, the subplot parameter must be specified with each call. The subplot value corresponds to a row-major ordering of panels, see the `subplot` plot option information.

Marking the region requires two consecutive mouse clicks at the two diagonally opposite corners. A hatched rectangle will appear over the selected region.

Alternative a specific region can be given to this function with the `region` parameter.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subplot</td>
<td>Three digits number: first digit for nx, second for ny, last for panel number.</td>
</tr>
<tr>
<td>region</td>
<td>[xmin, ymin, xmax, ymax] bounding box</td>
</tr>
</tbody>
</table>

Returns

`bool`

Example

980
# mark 2 flag regions on a multi-panel plot, one in panel 1 and one
# in panel 2.
  tp.markflags(subplot=131, region=[100, -100, 50, -50]);
  tp.markflags(subplot=221);
msplot.flagdata - Function

1.3.3 Set flags for all selected regions marked using mp.markregion()

Description

Set flags for all regions marked using `markflags()`. The plot is automatically redrawn after applying flags.

If reduction TaQL functions such as `sum`, `mean` are used, flags corresponding to all accessed values will be modified. For example, with a measurement set table, flagging on the mean amplitude of stokes 1 and channels 1 to 5, given by 'MEAN(AMPLITUDE(DATA[1,1:5]))' results in flags being set for all 5 accessed channels.

For a measurement set, by default, flags are set only for accessed channels and stokes when the DATA column is used. However all channels/stokes can be flagged for the marked flag regions by setting the corresponding row flag.

Arguments

Inputs

Returns

bool

Example

# mark 2 flag regions on a multi-panel plot, one in panel 1 and one # in panel 2. Then apply the flags and write to disk.
mp.markflags(subplot=221, region=[0,15,10,30]);
mp.markflags(subplot=222, region=[15,30,10,30]);
mp.flagdata();
msplot.unflagdata.html

**msplot.unflagdata - Function**

Unset flags in all regions marked using `mp.markregion()` Similar to the `mp.flagdata()`

**Description**

Unset flags for all regions marked using `markflags()`. See the `flagdata()` function for more information.

**Arguments**

- **Inputs**

**Returns**

bool

**Example**

```python
# mark 2 flag regions on a multi-panel plot with three rows of plots.
# One region is marked on panel 1 and one region on panel 2. Then the
# marked regions are applied unflagging data and writing the changes
# to disk.
mp.markflags(subplot=311);
mp.markflags(subplot=312);
mp.unflagdata();
```
msplot.clearflags.html

**msplot.clearflags - Function**

1.3.3 Clear all flags in the table. Note: This clears *all* flags and should be used with caution.

**Description**

Currently, this function clears all flags from the table. This will be modified to allow for selective un-flagging of previously flagged regions (specified by indexing into a stored history of marked flag-regions).

**Arguments**

**Returns**

bool

**Example**

```
# clear all flags from the subset of the measurement set.
mp := msplot( msname=['./data/3C273XC1.ms']);
mp.setdata( spwIndex=[0] );
mp.clearflags();
mp.done();
```
msplot.locatedata.html

**msplot.locatedata - Function**

1.3.3 Print info about data selected using mp.markregion().

**Description**

New functionality that is being added to the plotting facilities, as a result we’ve purposely not put any description in as we are still exploring how this function should work.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
</table>

**Returns**

bool

**Example**

---
msplot.saveflagversion.html

**msplot.saveflagversion - Function**

1.3.3 Save current flags, applied to the current measurement set with a version name.

### Description

### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Version name</th>
<th>Version name</th>
</tr>
</thead>
<tbody>
<tr>
<td>versionname</td>
<td>Version name</td>
<td>versionname</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>versionname</td>
<td>versionname</td>
</tr>
<tr>
<td>comment</td>
<td>Comment for this flag table</td>
<td>comment</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>comment</td>
<td>comment</td>
</tr>
<tr>
<td>merge</td>
<td>merge type: &quot;replace&quot; existing flag version, &quot;and&quot; logical AND with existing flag version, or &quot;or&quot; logical OR with existing flag version</td>
<td>merge</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
<td>merge</td>
</tr>
<tr>
<td>Default:</td>
<td>replace</td>
<td>merge</td>
</tr>
</tbody>
</table>

### Returns

bool

### Example

986
msplot.restoreflagversion.html

**msplot.restoreflagversion - Function**

1.3.3 Restore flags for the current Measurement Set.

**Description**

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>versionname</td>
<td>List of flag versions to restore from.</td>
</tr>
<tr>
<td>allowed</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td>replace</td>
</tr>
<tr>
<td>merge</td>
<td>merge type: &quot;replace&quot; existing flag version, &quot;and&quot; logical AND with existing flag version, or &quot;or&quot; logical OR with existing flag version</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>replace</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

987
msplot.deleteflagversion.html

**msplot.deleteflagversion - Function**

For the current measurement set delete a saved flag version.

### Description

### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Version name</th>
</tr>
</thead>
<tbody>
<tr>
<td>versionname</td>
<td>Version name</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

### Returns

bool

### Example
msplot.getflagversionlist.html

**msplot.getflagversionlist - Function**

1.3.3 Print out a list of saved flag versions, for the current Measurement Set.

**Description**

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
</table>

**Returns**

bool

**Example**

---

measures-Module.html

### 1.4 measures - Module

Measures handling

**Description**  A measure is a [quantity](#) with a specified reference frame (e.g. UTC, J2000, mars). The measures module provides an interface to the handling of measures. The basic functionality provided is:

- *Conversion* Conversion of measures, especially between different frames (e.g. UTC to LAST)
Calculation

This functionality is provided in a command line interface.

Measures

Measures are e.g. an epoch, or coordinates, which have, in addition to values (as quantities), also a reference specification, and possibly an offset. They are represented as records with fields describing the various entities embodied in the measure. These entities can be obtained by the access methods `gettype`, `getref`, `getoffset`, `getvalue`.

Each measure has its own list of reference codes (see the individual methods for creating them, like `direction`). If an empty or no code reference code is given, the default code for that type of measure will be used (e.g. it is J2000 for a direction). If an unknown code is given, this default is also returned, but with a warning message.

The values of a measure (like the right-ascension for a direction) are given as quantities. Each of them can be either a scalar quantity with a scalar or vector for its actual value (see the following example). E.g a vector of length 2 of quanta will be seen in a direction constructor as a longitude and a latitude.

```
# print "\t----\t Module Ex 1 \t----"
print me.epoch('utc','today') # note that your value will be different
#{'type': 'epoch', 'm0': {'value': 54175.86592379628, 'unit': 'd'}, 'refer': 'UTC'}
print me.direction('j2000','5h20m','-30.2deg')
#{'type': 'direction', 'm1': {'value': -0.52708943410228748, 'unit': 'rad'}, 'm0': {'value': 1.3962634015954634, 'unit': 'rad'}, 'refer': 'J2000'}
a = me.direction('j2000','5h20m','-30.2deg')
print me.gettype(a)
#Direction
print me.getoffset(a)
#{},
print me.getref(a)
#J2000
print me.getvalue(a)
#{'m1': {'value': -0.52708943410228748, 'unit': 'rad'}, 'm0': {'value': 1.3962634015954634, 'unit': 'rad'}}
print me.getvalue(a)['m0']
#{'value': 1.3962634015954634, 'unit': 'rad'}
print me.getvalue(a)['m1']
#{'value': -0.52708943410228748, 'unit': 'rad'}
print 'Last example! Exiting ...'
exit()
#
```

Known measures are:

990
• epoch: an instance in time (internally expressed as MJD or MGSD)
• direction: a direction towards an astronomical object (including planets, sun, moon)
• position: a position on Earth
• frequency: electromagnetic wave energy
• radialvelocity: radial velocity of astronomical object
• doppler: doppler shift (i.e. radial velocity in non-velocity units like 'Optical', 'Radio'.)
• baseline: interferometer baseline
• uvw: UVW coordinates
• earthmagnetic: Earth’s magnetic field

In addition to the reference code (like J2000), a measure needs sometimes more information to be convertible to another reference code (e.g. a time and position to convert it to an azimuth/elevation). This additional information is called the reference frame, and can specify one or more of 'where am i', 'when is it', 'what direction', 'how fast'.

The frame values can be set by the doframe tool function.
1.4.1 measures - Tool

measures tool

Requires:

Synopsis

Methods

- **dirshow**: Show direction measure as a string.
- **show**: Show a measure as a string
- **epoch**: define an epoch measure
- **direction**: define a direction measure
- **getvalue**: get the value of a measure
- **gettype**: get the type of a measure
- **getref**: get the reference code of a measure
- **getoffset**: get the offset of a measure
- **cometname**: get the current comet name
- **comettype**: get the current comet table type
- **cometdist**: get the distance of the current comet in the current frame
- **cometangdiam**: get the angular diameter of the current comet in the current frame
- **comettopo**: get the current comet table coordinates
- **framecomet**: set the current comet table
- **position**: define a position measure
- **observatory**: get position of an observatory
- **obslist**: get a list of known observatories
- **linelist**: get a list of known spectral lines
- **spectralline**: get frequency of a spectral line
- **source**: get a list of known sources
- **source**: get direction of a source
- **frequency**: define a frequency measure
- **doppler**: define a doppler measure
- **radialvelocity**: define a radialvelocity measure
- **shift**: Shift a direction measure by an offset angle at a position angle.
- **uvw**: define a uvw measure
- **touvw**: calculate a uvw measure from a baseline
- **expand**: expand n positions to n*(n-1)/2 baselines
- **earthmagnetic**: define an earthmagnetic measure
- **baseline**: define a baseline measure
- **asbaseline**: define a baseline from a position measure
- **listcodes**: get known reference code names (list indices do not necessarily correspond to enumeration)
- **measure**: convert a measure to another reference
- **doframe**: save a measure as frame reference
- **framenow**: set the active frame time at now
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>showframe</td>
<td>show the currently active frame reference</td>
</tr>
<tr>
<td>toradialvelocity</td>
<td>convert a doppler type value to a real radial velocity</td>
</tr>
<tr>
<td>tofrequency</td>
<td>convert a doppler type value to a frequency</td>
</tr>
<tr>
<td>todoppler</td>
<td>convert a frequency or radialvelocity measure to a doppler measure</td>
</tr>
<tr>
<td>forestfrequency</td>
<td>convert a frequency and doppler measure to a rest frequency</td>
</tr>
<tr>
<td>rise</td>
<td>get rise and set sidereal time</td>
</tr>
<tr>
<td>riserset</td>
<td>get rise and set times</td>
</tr>
<tr>
<td>posangle</td>
<td>get position angle of two directions</td>
</tr>
<tr>
<td>separation</td>
<td>get separation angle between two directions</td>
</tr>
<tr>
<td>addxvalue</td>
<td>get some additional measure information</td>
</tr>
<tr>
<td>type</td>
<td>type of tool</td>
</tr>
<tr>
<td>done</td>
<td>free resources used by tool.</td>
</tr>
<tr>
<td>ismeasure</td>
<td>Check if measure</td>
</tr>
</tbody>
</table>
measures.dirshow.html

**measures.dirshow - Function**

1.4.1
Show direction measure as a string.

**Description**

dirshow will convert a direction measure to a string

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Returns**

string

**Example**

```plaintext
print "\t----\t dirshow Ex 1 \t----"
print me.dirshow(me.direction('venus'))
# [0, 90] deg VENUS
```
measures.show

**measures.show - Function**

1.4.1 Show a measure as a string

**Description**

show will convert a measure to a string.

All measures are catered for (at this moment *direction, position, epoch, radialvelocity, frequency, doppler, baseline, uvw, earthmagnetic*).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>measure value to be converted to string</td>
</tr>
<tr>
<td></td>
<td>allowed: record</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>refcode</td>
<td>add the reference code to output</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: true</td>
</tr>
</tbody>
</table>

**Returns**

string

**Example**

```python
print "\t----\t show Ex 1 \t----"
print me.show(me.frequency('lsrk', qa.constants('HI')))
#1.42041e+09 Hz LSRK
print me.show(me.frequency('lsrk', qa.constants('HI')), refcode=false)
#1.42041e+09 Hz
```
measures.epoch.html

measures.epoch - Function

1.4.1 define an epoch measure

Description

epoch defines an epoch measure from the CLI. It has to specify a reference code, an epoch quantity value (see introduction for the action on a scalar quantity with either a vector or scalar value), and optionally it can specify an offset, which in itself has to be an epoch. Allowable reference codes are: UTC TAI LAST LMST GMST1 GAST UT1 UT2 TDT TCG TDB TCB.

Note that additional ones may become available. Check in CASA with:

```python
print "\t----\t epoch Ex 1 \t----"
print me.listcodes(me.epoch())
```

See quantity for possible time formats.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rf</td>
<td>reference code</td>
</tr>
<tr>
<td>v0</td>
<td>epoch value</td>
</tr>
<tr>
<td>off</td>
<td>optional offset epoch measure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>allowed</th>
<th>any</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>variant</td>
</tr>
<tr>
<td>allowed</td>
<td>record</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
</tbody>
</table>

Returns

record
Example

print "\t----\t epoch Ex 2 \t----"
print me.epoch('utc','today')
#{'m0': {'value': 54048.861237743055, 'unit': 'd'},
  # 'refer': 'UTC',
  # 'type': 'epoch'}
measures.direction.html

measures.direction - Function

[1.4.1] define a direction measure

Description

direction defines a direction measure from the CLI. It has to specify a reference code, direction quantity values (see introduction for the action on a scalar quantity with either a vector or scalar value), and optionally it can specify an offset, which in itself has to be a direction. Allowable reference codes are: J2000 JMEAN JTRUE APP B1950 BMEAN BTRUE GALACTIC HADEC AZEL SUPERGAL ECLIPTIC MECLIPTIC TECLIPTIC MERCURY VENUS MARS JUPITER SATURN URANUS NEPTUNE PLUTO MOON SUN COMET.

Note that additional ones may become available. Check in CASA with:

```python
print "\t----\t direction Ex 1 \t----"
print me.listcodes(me.direction())
```

The direction quantity values should be longitude(angle) and latitude(angle) (none needed for planets: the frame epoch defines coordinates). See quantity for possible angle formats.

Arguments
Inputs

- **rf**
  - reference code
  - allowed: string
  - Default: J2000

- **v0**
  - longitude
  - allowed: any
  - Default: variant

- **v1**
  - latitude
  - allowed: any
  - Default: variant

- **off**
  - optional offset direction measure
  - allowed: record
  - Default: 

Returns

record

Example

```python
print "\t----\t direction Ex 2 \t----"
print me.direction('j2000','30deg','40deg')
#{'m0': {'value': 0.52359877559829882, 'unit': 'rad'},
 # 'm1': {'value': 0.69813170079773168, 'unit': 'rad'},
 # 'refer': 'J2000',
 # 'type': 'direction'}
#
print me.direction('mars')
#{'m0': {'value': 0.0, 'unit': 'rad'},
 # 'm1': {'value': 1.5707963267948966, 'unit': 'rad'},
 # 'refer': 'MARS',
 # 'type': 'direction'}
```
**measures.getvalue.html**

**measures.getvalue - Function**

1.4.1 get the value of a measure

**Description**

getvalue gets the actual implementation value of the measure.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>v</code> measure (array of measures)</td>
<td>allowed: record</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
print "\t----\t getvalue Ex 1 \t----"
b=me.direction('j2000','0deg','80deg')
print me.getvalue(b)
#{'m0': {'value': 0.0, 'unit': 'rad'},
 # 'm1': {'value': 1.3962634015954634, 'unit': 'rad'}}
```
measures.gettype.html

**measures.gettype - Function**

1.4.1 get the type of a measure

**Description**

getype gets the actual type of the measure.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>v measure (array of measures)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>allowed: record</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

string

**Example**

```plaintext
print "\t----\t gettype Ex 1 \t----"
b=me.direction('j2000','0deg','80deg')
print me.getvalue(b)
#{'m0': {'value': 0.0, 'unit': 'rad'},
# 'm1': {'value': 1.3962634015954634, 'unit': 'rad'}}
print me.gettype(b)
# 'Direction'
```

1001
measures.getref.html

**measures.getref - Function**

1.4.1 get the reference code of a measure

**Description**

gettype gets the actual reference code of the measure.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>measure (array of measures)</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

string

**Example**

```python
print "\t----\t getref Ex 1 \t----"
b=me.direction('j2000','0deg','80deg')
print me.getvalue(b)
#{'m0': {'value': 0.0, 'unit': 'rad'},
#  'm1': {'value': 1.3962634015954634, 'unit': 'rad'}}
print me.gettype(b)
#'Direction'
print me.getref(b)
#'J2000'
```
measures.getoffset.html

**measures.getoffset - Function**

1.4.1 get the offset of a measure

**Description**

goffset gets the actual offset of the measure (as a measure) or F if no offset given.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v measure</td>
<td>array of measures</td>
</tr>
<tr>
<td>allowed: record</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
print "\t--\t getoffset Ex 1 \t--"
b=me.direction('j2000','0deg','80deg')
print me.getvalue(b)
#{'m0': {'value': 0.0, 'unit': 'rad'},
 # 'm1': {'value': 1.3962634015954634, 'unit': 'rad'}}
print me.gettype(b)
#'Direction'
print me.getref(b)
#'J2000'
print me.getoffset(b)
#{}
```
measures.cometname.html

**measures.cometname - Function**

1.4.1 get the current comet name

**Description**

cometname gets the name of the current comet (if any).

**Arguments**

**Inputs**

**Returns**

string

**Example**

```plaintext
print "\t----\t cometname Ex 1 \t----"
print me.cometname()
#Thu Nov 9 21:27:25 2006 WARN :
#Method cometname fails! No Comet table present
#'
```
measures.comettype - Function

get the current comet table type

Description

comettype gets the comet table type (apparent or topocentric)

Arguments

Inputs

Returns

string

Example

```plaintext
print "\t----\t comettype Ex 1 \t----"
print me.comettype()
# 'none'
```
measures.cometdist - Function

1.4.1 get the distance of the current comet in the current frame

Description

cometdist returns the distance in AU of the current comet in the current frame, as a quantity. It will return -1 AU on failure!

Arguments

Inputs

Returns

record

Example

print "\t----\t cometdist Ex 1 \t----"
# Directory with several Solar System ephemerides for setjy.
cometdir = os.getenv("CASAPATH").split()[0] + "/data/ephemerides/JPL-Horizons/"
me.framecomet(cometdir + "Ganymede_55438-56292dUTC.tab")
# Out[5]: True
me.doframe(me.epoch("utc", "2011/01/03/17:00:00"))
me.doframe(me.observatory("ALMA"))
gandist = me.cometdist()
print gandist
# {'value': 5.1241088343892631, 'unit': 'AU'}
measures.cometangdiam.html

**measures.cometangdiam - Function**

1.4.1 get the angular diameter of the current comet in the current frame

**Description**

cometdist returns the angular diameter (as seen from Earth) in AU of the current comet in the current frame, as a quantity. It will return -1 radians on failure!

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Returns</th>
</tr>
</thead>
</table>

| record |

**Example**

```python
print "\t\t cometangdiam Ex 1 \t---
# Directory with several Solar System ephemerides for setjy.
cometdir = os.getenv("CASAPATH").split()[0] + "/data/ephemerides/JPL-Horizons/
me.framecomet(cometdir + "Ganymede_55438-56292dUTC.tab")
# Out[5]: True
me.doframe(me.epoch("utc", "2011/01/03/17:00:00"))
me.doframe(me.observatory("ALMA"))
gad = me.cometangdiam()
print gad
# {‘unit’: ’rad’, ‘value’: 6.8679673431729014e-06}
```
measures.comettopo.html

**measures.comettopo - Function**

1.4.1 get the current comet table coordinates

**Description**

comettopo gets the comet table’s topographic coordinates used.

**Arguments**

**Inputs**

**Returns**

record

**Example**

```python
print "\t----\t comettopo Ex 1 \t----"
print me.comettopo()
#Thu Nov 9 21:45:40 2006 WARN :
#Method comettopo fails! No Topocentric Comet table present
#{'value': [0.0], 'unit': '}
```
measures.framecomet.html

**measures.framecomet - Function**

1.4.1 set the current comet table

**Description**

framecomet will put the specified comet table in the frame.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>name of a table</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
print "\t==== framecomet Ex 1 \t===="
print me.framecomet('VGEO')
#True
print me.showframe()
# Frame: VENUS comet between MJD 50802.7 and 50803.1
print me.cometname()
# 'VENUS'
print me.comettype()
# 'APP'
print me.doframe(me.epoch('et',qa.quantity('1997/12/20/17:30:0')))
#True
print me.measure(me.direction('comet'),'app')
# {'m0': {'value': -0.94936485919663083, 'unit': 'rad'},
# 'm1': {'value': -0.34710256485894436, 'unit': 'rad'},
# 'refer': 'APP'},
```
# 'type': 'direction'}

1010
measures.position.html

**measures.position - Function**

1.4.1 define a position measure

**Description**

position defines a position measure from the CLI. It has to specify a reference code, position quantity values (see introduction for the action on a scalar quantity with either a vector or scalar value), and optionally it can specify an offset, which in itself has to be a position. Allowable reference codes are: *WGS84* *ITRF* (World Geodetic System and International Terrestrial Reference Frame).

Note that additional ones may become available. Check in CASA with:

```python
print "\t----\t position Ex 1 \t----"
print me.listcodes(me.position())
#{'normal': ['ITRF', 'WGS84'], 'extra': []}
```

The position quantity values should be either longitude (angle), latitude(angle) and height(length); or x,y,z (length). See quantity for possible angle formats.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>rf</td>
<td>reference code</td>
<td>string</td>
<td>WGS84</td>
</tr>
<tr>
<td>v0</td>
<td>longitude or x</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>v1</td>
<td>latitude or y</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>v2</td>
<td>height or z</td>
<td>any</td>
<td>variant</td>
</tr>
</tbody>
</table>

off optional offset position measure

<table>
<thead>
<tr>
<th>allowed</th>
<th>record</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td></td>
</tr>
</tbody>
</table>
Returns
record

Example

print "\t\t\t\t\t\t position Ex 2 \t\t\t\t\t\t"
print me.position('wgs84','30deg','40deg','10m')
#{'m0': {'value': 0.52359877559829882, 'unit': 'rad'},
 # 'm1': {'value': 0.6981317007977319, 'unit': 'rad'},
 # 'm2': {'value': 9.9999999999999982, 'unit': 'm'},
 # 'refer': 'WGS84',
 # 'type': 'position'}
print me.observatory('ATCA')
#{'m0': {'value': 2.6101423190348916, 'unit': 'rad'},
 # 'm1': {'value': -0.5261379196128062, 'unit': 'rad'},
 # 'm2': {'value': 6372960.2577234386, 'unit': 'm'},
 # 'refer': 'ITRF',
 # 'type': 'position'}

###One can use a quantity-vectors especially when dealing with multiple antenna positions

ants=me.position('itrf',qa.quantity([3828763.11, 3828746.55, 3828727.43],'m'), qa.quantity([442449.106, 442592.14, 442580.12],'m'), qa.quantity([5064923.01, 5064923.01, 5064923.51],'m'))
print ants
#{'m0': {'unit': 'rad',
 # 'value': array([ 0.11504897, 0.11508633, 0.1150838 ])},
 # 'm1': {'unit': 'rad',
 # 'value': array([ 0.92031276, 0.92031276, 0.92031535])},
 # 'm2': {'unit': 'm',
 # 'value': array([ 6364639.28758924, 6364639.27051283, 6364627.33064587])},
 # 'refer': 'ITRF',
 # 'type': 'position'}
measures.observatory.html

measures.observatory - Function

1.4.1 get position of an observatory

Description

observatory will give you the position of an observatory as given in the system. At the time of writing the following observatories are recognised (but check e.g. the position GUI for currently known ones, or the me.obslist() tool function): 'ALMA' 'ARECIBO' 'ATCA' 'BIMA' 'CLRO' 'DRAO' 'DWL' 'GB' 'GBT' 'GMRT' 'IRAM PDB' 'IRAM_PDB' 'JCMT' 'MOPRA' 'MOST' 'NRAO12M' 'NRAO_GBT' 'PKS' 'SAO SMA' 'SMA' 'VLA' 'VLBA' 'WSRT' 'ATF' 'ATA' 'CARMA' 'ACA' 'OSF' 'OVRO_MMA' 'EVL'A' 'ASKAP' 'APEX' 'SMT' 'NRO' 'ASTE' 'LOFAR' 'MeerKAT' 'KAT-7' 'EVN' 'LWA1' 'PAPER_SA' 'PAPER_GB' 'e-MERLIN' 'MERLIN2' 'Effelsberg' 'MWA32T'.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>observatory name - case insensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: ALMA</td>
</tr>
</tbody>
</table>

Returns

record

Example

print "\t----\t observatory Ex 1 \t----"
print me.observatory('ATCA')
#{'m0': {'value': 2.6101423190348916, 'unit': 'rad'},
# 'm1': {'value': -0.5261379196128062, 'unit': 'rad'},
# 'm2': {'value': 6372960.2577234386, 'unit': 'm'},
# 'refer': 'ITRF',
# 'type': 'position'}
measures.obslist.html

measures.obslist - Function

1.4.1 get a list of known observatories

Description

obslist will give you an array of strings of the observatories known in the Observatories table.

Arguments

Returns

stringArray

Example

print "\t----\tobslist Ex 1 \t----"
print me.obslist()

#['ALMA' 'ARECIBO' 'ATCA' 'BIMA' 'CLRO' 'DRAO' 'DWL' 'GB' 'GBT' 'GMRT' 'IRAM_PDB' 'IRAM_PDB' 'JCMT' 'MOPRA' 'MOST' 'NRAO12M' 'NRAO_GBT' 'PKS' 'SAO_SMA' 'SMA' 'VLA' 'VLBA' 'WSRT' 'ATF' 'ATA' 'CARMA' 'ACA' 'OSF' 'OVRO_MMA' 'EVLA' 'ASKAP' 'APEX' 'SMT' 'NRO' 'ASTE' 'LOFAR' 'MeerKAT' 'KAT-7' 'EVEN' 'LWA1' 'PAPER_SA' 'PAPER_GB' 'e-MERLIN' 'MERLIN2' 'Effelsberg' 'MWA32T']
measures.linelist.html

**measures.linelist - Function**

1.4.1 get a list of known spectral lines

**Description**

linelist will give you a string with a space separated list of spectral lines known in the Lines table. A number of lines are available now, but tables with many lines are already online, and will be interfaced once a nomenclature can be defined for the tens of thousands of lines.

**Arguments**

**Returns**

string

**Example**

```
print "\t----\t linelist Ex 1 \t----"
print me.linelist()
#'C109A CI CII166A DI H107A H110A H138B H166A H240A H272A
# H2CO HE110A HE138B HI OH1612 OH1665 OH1667 OH1720'
```
measures.spectralline.html

**measures.spectralline - Function**

1.4.1 get frequency of a spectral line

**Description**

spectralline will give you the frequency of a spectral line. The known list can be obtained by me.linelist().

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>name</th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name</td>
<td>string</td>
<td>HI</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
print "\t----\t spectralline Ex 1 \t----"
print me.spectralline('HI')
#{'m0': {'value': 1420405751.786, 'unit': 'Hz'},
  # 'refer': 'REST',
  # 'type': 'frequency'}
```
measures.sourcelist.html

**measures.sourcelist - Function**

1.4.1 get a list of known sources

**Description**

sourcelist will give you a string with the space separated list of sources known in the Sources table.

**Arguments**

**Returns**

string

**Example**

```plaintext
print "\t----\t sourcelist Ex 1 \t----"
print me.sourcelist()[0:62]
# '0002-478 0003+380 0003-066 0007+106 0007+171 0008-264 0008-421'
# ......
```
### measures.source - Function

1.4.1 get direction of a source

#### Description

`source` will give you the direction of a source. The known list can be obtained by `me.sourcelist()`.

#### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>name</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td>variant 1934-638</td>
</tr>
</tbody>
</table>

#### Returns

record

#### Example

```python
print "\t----\t source Ex 1 \t----"
print me.source()
print me.source('1934-638')
# Out[19]:
#{'m0': {'value': -1.1370073467795063, 'unit': 'rad'},
# 'm1': {'value': -1.1119959323803881, 'unit': 'rad'},
# 'refer': 'ICRS',
# 'type': 'direction'}
```
measures.frequency.html

measures.frequency - Function

1.4.1 define a frequency measure

Description

frequency defines a frequency measure from the CLI. It has to specify a reference code, frequency quantity value (see introduction for the action on a scalar quantity with either a vector or scalar value), and optionally it can specify an offset, which in itself has to be a frequency. Allowable reference codes are: REST LSRK LSRD BARY GEO TOPO GALACTO LGROUP CMB.

Note that additional ones may become available. Check in CASA with:

```python
print "\t\t frequency Ex 1 \t\t"
print me.listcodes(me.frequency())
#('normal': ['REST', 'LSRK', 'LSRD', 'BARY', 'GEO', 'TOPO',
# 'GALACTO', 'LGROUP', 'CMB'], 'extra': [])
```

The frequency quantity values should be in one of the recognised units (examples all give same frequency):

- value with time units: a period (0.5s)
- value as frequency: 2Hz
- value in angular frequency: 720deg/s
- value as length: 149896km
- value as wave number: 4.19169e-8m-1
- value as energy (h.nu): 8.27134e-9ueV
- value as momentum: 4.42044e-42kg.m

Arguments
## Inputs

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>rf</strong></td>
<td>reference code</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>LSRK</td>
</tr>
<tr>
<td><strong>v0</strong></td>
<td>frequency/wavelength/...</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
<tr>
<td><strong>off</strong></td>
<td>optional offset frequency measure</td>
</tr>
<tr>
<td>allowed:</td>
<td>record</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

## Returns

record

## Example

```python
print "\t----\t frequency Ex 2 \t----"
print me.frequency('lsrk','5GHz')
#{'m0': {'value': 5000000000.0, 'unit': 'Hz'},
  # 'refer': 'LSRK',
  # 'type': 'frequency'}
print me.frequency('lsrk','21cm')
#{'m0': {'value': 1427583133.3333333, 'unit': 'Hz'},
  # 'refer': 'LSRK',
  # 'type': 'frequency'}
```
measures.doppler.html

measures.doppler - Function

1.4.1 define a doppler measure

Description

doppler defines a doppler measure from the CLI. It has to specify a reference code, doppler quantity value (see introduction for the action on a scalar quantity with either a vector or scalar value), and optionally it can specify an offset, which in itself has to be a doppler. Allowable reference codes are: RADIO Z RATIO BETA GAMMA OPTICAL TRUE RELATIVISTIC.

Note that additional ones may become available. Check in CASA with:

```python
print "\t----\t doppler Ex 1 \t----"
print me.listcodes(me.doppler())
#{'normal': ['RADIO', 'Z', 'RATIO', 'BETA', 'GAMMA', 'OPTICAL', 
 'TRUE', 'RELATIVISTIC'], 'extra': []}
```

The doppler quantity values should be either non-dimensioned to specify a ratio of the light velocity, or in velocity.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>reference code</th>
</tr>
</thead>
<tbody>
<tr>
<td>rf</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: RADIO</td>
</tr>
<tr>
<td>v0</td>
<td>doppler ratio/velocity</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
</tbody>
</table>

| off      | optional offset doppler measure |
|          | allowed: record               |
|          | Default:                     |

Returns

record
Example

Examples both give same doppler:

```python
print "\t--\tdoppler Ex 2 \t--"
print me.doppler('radio','0.4')
#{'m0': {'value': 119916983.2, 'unit': 'm/s'},
  # 'refer': 'RADIO',
  # 'type': 'doppler'}
print me.doppler('radio',qa.mul(qa.quantity('0.4'),qa.constants('c')))
#{'m0': {'value': 119916983.2, 'unit': 'm/s'},
  # 'refer': 'RADIO',
  # 'type': 'doppler'}
```
measures.radialvelocity.html

**measures.radialvelocity - Function**

1.4.1 define a radialvelocity measure

**Description**

radialvelocity defines a radialvelocity measure from the CLI. It has to specify a reference code, radialvelocity quantity value (see introduction for the action on a scalar quantity with either a vector or scalar value), and optionally it can specify an offset, which in itself has to be a radialvelocity. Allowable reference codes are: *LSRK LSRD BARY GEO TOPO GALACTO LGROUP CMB*.

Note that additional ones may become available. Check in CASA with:

```python
print "\t----\t radialvelocity Ex 1 \t----"
print me.listcodes(me.radialvelocity())
```

```python
# Out[17]:
#{'extra': [],
# 'normal': ['LSRK', 'LSRD', 'BARY', 'GEO', 'TOPO', 'GALACTO',
# 'LGROUP', 'CMB']}
```

The radialvelocity quantity values should be given as velocity.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>rf</td>
<td>reference code</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>LSRK</td>
</tr>
<tr>
<td>v0</td>
<td>radial velocity</td>
<td>any</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>variant</td>
</tr>
<tr>
<td>off</td>
<td>optional offset radialvelocity measure</td>
<td>record</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

record
Example

```
print "\t----\t radialvelocity Ex 2 \t----"
print me.radialvelocity('lsrk','20km/s')
# Out[18]:
#{'m0': {'value': 20000.0, 'unit': 'm/s'},
# 'refer': 'LSRK',
# 'type': 'radialvelocity'}
```
measures.shift.html

**measures.shift - Function**

1.4.1 Shift a direction measure by an offset angle at a position angle.

**Description**

This method calculates the direction measure located at the specified offset angular amount along the specified position angle from the specified direction measure.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>The direction measure to shift, represented as a record. allowed: record Default:</td>
</tr>
<tr>
<td>offset</td>
<td>The angular offset, represented as a quantity record or string. allowed: any Default: variant 0deg</td>
</tr>
<tr>
<td>pa</td>
<td>Position angle of the offset, measured from the positive latitude axis through the positive longitude axis. allowed: any Default: variant 0deg</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```plaintext
v = me.direction("J2000", "13:22:44", ";50.20.20")
# shift along 4 arcminutes at a pa of 30 degrees.
offset = me.shift(v, offset="4arcmin", pa="30deg")
```
measures.uvw.html

**measures.uvw - Function**

**[1.4.1]** define a uvw measure

**Description**

uvw defines a uvw measure from the CLI. It has to specify a reference code, uvw quantity values (see introduction for the action on a scalar quantity with either a vector or scalar value), and optionally it can specify an offset, which in itself has to be a uvw. Allowable reference codes are ITRF and the direction ones. Note that additional ones may become available. Check in CASA with:

```python
print "\t----	 uvw Ex 1 \t----"
print me.listcodes(me.uvw())
```

The uvw quantity values should be either longitude (angle), latitude(angle) and height(length); or x,y,z (length). See quantity for possible angle formats.

**Arguments**
Inputs

rf  reference code
    allowed: string
    Default: ITRF
v0  longitude or x
    allowed: any
    Default: variant
v1  latitude or y
    allowed: any
    Default: variant
v2  height or z
    allowed: any
    Default: variant
off optional offset uvw measure
    allowed: record
    Default:

Returns

record

Example

print "\t----\t uvw Ex 2 \t----"
print me.uvw('itrf','30deg','40deg','10m')
#{'m0': {'value': 0.52359877559829882, 'unit': 'rad'},
  # 'm1': {'value': 0.6981317007977319, 'unit': 'rad'},
  # 'm2': {'value': 9.9999999999999982, 'unit': 'm'},
  # 'refer': 'ITRF',
  # 'type': 'uvw'}
print me.doframe(me.epoch('utc','today'))
#True
print me.doframe(me.observatory('ALMA'))
#True
print me.doframe(me.direction('mars'))
#True
print me.measure(me.uvw('itrf','30deg','40deg','10m'), 'j2000')
#{'m0': {'value': 0.52321924738347259, 'unit': 'rad'},
   # 'm1': {'value': 0.6981317007977319, 'unit': 'rad'},
   # 'm2': {'value': 9.9999999999999982, 'unit': 'm'},
   # 'refer': 'ITRF',
   # 'type': 'uvw'}
# 'm1': {'value': 0.69813169995801672, 'unit': 'rad'},
# 'm2': {'value': 10.0, 'unit': 'm'},
# 'refer': 'J2000',
# 'type': 'uvw
measures.touvw.html

**measures.touvw - Function**

1.4.1 calculate a uvw measure from a baseline

**Description**

touvw calculates a uvw measure from a baseline. Note that the baseline does not have to be a proper *baseline*, but can be a series of positions (to call positions baselines see asbaseline ) for speed reasons: operations are linear and can be done on positions, which are converted to baseline values at the end (with expand ).

Whatever the reference code of the baseline, the returned *uvw* will be given in J2000. If the *dot* argument is given, that variable will be filled with a quantity array consisting of the time derivative of the uvw (note that only the sidereal rate is taken into account; not precession, earth tides and similar variations, which are much smaller). If the *xyz* variable is given, it will be filled with the quantity values of the uvw measure.

The values of the input baselines can be given as a quantity vector per x, y or z value.

*uvw* coordinates are calculated for a certain direction in the sky; hence the frame has to contain the direction for the calculation to work. Since the baseline and the sky rotate with respect of each other, the time should be specified as well.

**Arguments**

<table>
<thead>
<tr>
<th>Outputs</th>
<th>uvw-dot (quantity array)</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>dot</em></td>
<td>allowed: record</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td><em>xyz</em></td>
<td>uvw (quantity array)</td>
</tr>
<tr>
<td></td>
<td>allowed: record</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inputs</th>
<th>baseline measure</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>v</em></td>
<td>allowed: record</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

1030
Example

print "\t----\t touvw Ex 1 \t----"
print me.doframe(me.observatory('atca'))
#True
print me.doframe(me.source('1934-638'))
#True
print me.doframe(me.epoch('utc', qa.unit('today')))
#True
b=me.baseline('itrf','10m','20m','30m')
print me.touvw(b)
#{'dot': {'unit': 'm/s',
    # 'value': [-0.0011912452908351659,
    # -0.00098731747136827593,
    # -0.00048769097314181744]},
    # 'return': {'m0': {'value': -0.094777304811312649, 'unit': 'rad'},
    # 'm1': {'value': -1.1509286139398101, 'unit': 'rad'},
    # 'm2': {'value': 37.416573867739416, 'unit': 'm'},
    # 'refer': 'J2000',
    # 'type': 'uvw'},
    # 'xyz': {'unit': 'm',
    # 'value': [15.184026188402472,
    # -1.4434256399579168,
    # -34.166677788919138]}}
print me.getvalue(me.touvw(b))
#{'m0': {'value': -0.094777304811312649, 'unit': 'rad'},
    # 'm1': {'value': -1.1509286139398101, 'unit': 'rad'},
    # 'm2': {'value': 37.416573867739416, 'unit': 'm'}}
print me.getvalue(me.touvw(b))['m0']
#{'value': -0.094777304811312649, 'unit': 'rad'}

###Or when you are dealing with multiple antennas
###set the frame...i,e where, direction and when.
me.doframe(me.observatory('VLA'))
me.doframe(me.direction('J2000', '19h20m00', '20d10m00'))
me.doframe(me.epoch('utc', '2007/07/08/20:30:00'))
###antenna positions
ants=me.position('itrf', qa.quantity([3828763.11, 3828746.55], [3828727.43]), qa.quantity([442449.106, 442592.14, 442580.12], [442580.12]), qa.quantity([5064923.01, 5064923.01, 5064923.51], [5064923.01]))
###convert to baseline measures
bl=me.asbaseline(ants)
###convert to uvw
me.touvw(bl)

#{'dot': {'unit': 'm/s',
  'value': array([ 181.25190155, -73.29924893, 199.57974846, 181.25985238,
    -73.29691498, 199.57339353, 181.2583565 , -73.29668498,
    199.57276731])},

'return': {'m0': {'unit': 'rad',
  'value': array([ 2.21611194, 2.21610131, 2.21609887])},

'm1': {'unit': 'rad',
  'value': array([ 0.6984441 , 0.69846521, 0.69846285])},

'm2': {'unit': 'm',
  'value': array([ 6364639.28758924, 6364639.27051283, 6364627.33064587]),

'refer': 'J2000',
'type': 'uvw'},

'xyz': {'unit': 'm',
  'value': array([-2931661.69632123, 3894141.52172208, 4092639.28758924,
    -2931568.34776551, 3894103.64373003, 4092737.08879791,
    -2931559.14911939, 3894111.22249941, 4092717.89890567])}}}

###print the (n-1)n/2 baselines(u,v,w)
me.expand(me.touvw(bl)['return'])['xyz']

#{'unit': 'm',
  'value': array([ 93.34855573, -37.87799205, 102.8798504 , 102.54720184,
    -30.29922267, 83.68995815, 9.19864612, 7.57876938,
    -19.18989224])}
measures.expand.html

**measures.expand - Function**

1.4.1 expand n positions to n*(n-1)/2 baselines

**Description**

expand calculates the differences between a series of given measure values: it calculates baseline values from position values. The returned value is a measure, but the value of the optional output variable *xyz* will be set to an array of values.

**Arguments**

<table>
<thead>
<tr>
<th>Outputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><em>xyz</em></td>
<td><em>uvw</em> (quantity array)</td>
</tr>
<tr>
<td></td>
<td>allowed: record</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Inputs**

<table>
<thead>
<tr>
<th>v</th>
<th>measure (baseline, position or uvw measure)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>allowed: record</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
print "\t\t\t expand Ex 1 \t\t"
b=me.baseline('itrf', qa.quantity([10, 20, 30], 'm'), qa.quantity([10, 20, 30], 'm'), qa.quantity([0, 0, 0], 'm'))
print me.expand(b)
me.expand(b)

#{'return': {'m0': {'unit': 'rad',
#                  'value': array([ 0.78539816, 0.78539816, 0.78539816])},
# 'm1': {'unit': 'rad', 'value': array([ 0., 0., 0.])},
```

1033
print me.expand(b)['xyz']['value']

# [ 10. 10. 0. 20. 20. 0. 10. 10. 0.]
measures.earthmagnetic.html

**measures.earthmagnetic - Function**

1.4.1 define an earthmagnetic measure

**Description**

earthmagnetic defines an earthmagnetic measure from the CLI. It needs a reference code, earthmagnetic quantity values (see introduction for the action on a scalar quantity with either a vector or scalar value) if the reference code is not for a model, and optionally it can specify an offset, which in itself has to be a earthmagnetic. In general you specify a model (IGRF is the default and the only one known) and convert it to an explicit field. (See http://fdd.gsfc.nasa.gov/IGRF.html for information on the International Geomagnetic Reference Field). The earthmagnetic quantity values should be either longitude (angle), latitude(angle) and length(field strength); or x,y,z (field). See quantity for possible angle formats.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>reference code</th>
</tr>
</thead>
<tbody>
<tr>
<td>rf</td>
<td>reference code</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: IGRF</td>
</tr>
<tr>
<td>v0</td>
<td>Field strength</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
<tr>
<td>v1</td>
<td>longitude</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
<tr>
<td>v2</td>
<td>latitude</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
<tr>
<td>off</td>
<td>optional offset earthmagnetic measure</td>
</tr>
<tr>
<td></td>
<td>allowed: record</td>
</tr>
<tr>
<td></td>
<td>Default: 1035</td>
</tr>
</tbody>
</table>
Returns
record

Example

print "\t----\t earthmagnetic Ex 1 \t----"
print me.earthmagnetic('igrf')
#{'type': 'earthmagnetic', 'refer': 'IGRF', 'm1': {'value': 0.0, 'unit': 'nT'},
  # 'm0': {'value': 6.1230317691118855e-23, 'unit': 'nT'},
  # 'm2': {'value': 9.999999999999995e-07, 'unit': 'nT'}}
print me.doframe(me.observatory('atca'))
print me.doframe(me.source('1934-638'))
print me.doframe(me.epoch('utc', qa.unit('today')))
print me.measure(me.earthmagnetic('igrf'), 'j2000')
#{'type': 'earthmagnetic', 'refer': 'J2000',
  # 'm1': {'value': -8664.8767628222304, 'unit': 'nT'},
  # 'm0': {'value': 50544.054410564473, 'unit': 'nT'},
  # 'm2': {'value': 1799.5131920958615, 'unit': 'nT'}}
measures.baseline.html

measures.baseline - Function

1.4.1 define a baseline measure

Description

baseline defines a baseline measure from the CLI. It has to specify a reference code, baseline quantity values (see introduction for the action on a scalar quantity with either a vector or scalar value, and when a vector of quantities is given), and optionally it can specify an offset, which in itself has to be a baseline. Allowable reference codes are ITRF and the direction ones. Note that additional ones may become available. Check in CASA with:

```
print "\t----\t baseline Ex 1 \t----"
print me.listcodes(me.baseline())
#{'normal': ['J2000', 'JMEAN', 'JTRUE', 'APP', 'B1950', 'BMEAN', 'BTRUE',
# 'GALACTIC', 'HADEC', 'AZEL', 'AZELSW', 'AZELNE', 'AZELGEO', 'AZELSWGEO',
# 'AZELNEGEO', 'JNAT', 'ECLIPTIC', 'MECLIPTIC', 'TECLIPTIC', 'SUPERGAL',
# 'ITRF', 'TOPO', 'ICRS'], 'extra': []}
```

The baseline quantity values should be either longitude (angle), latitude(angle) and height(length); or x,y,z (length). See quantity for possible angle formats.

Arguments
Inputs

rf reference code
  allowed: string
  Default: ITRF

v0 longitude or x
  allowed: any
  Default: variant

v1 latitude or y
  allowed: any
  Default: variant

v2 height or z
  allowed: any
  Default: variant

off optional offset baseline measure
  allowed: record
  Default: 

Returns

record

Example

print "\t---\t Ex 2 \t---"
print me.baseline('itrf','30deg','40deg','10m')
#{'m0': {'value': 0.52359877559829882, 'unit': 'rad'},
  # 'm1': {'value': 0.6981317007977319, 'unit': 'rad'},
  # 'm2': {'value': 9.9999999999999982, 'unit': 'm'},
  # 'refer': 'ITRF',
  # 'type': 'baseline'}
print me.doframe(me.observatory('atca'))
print me.doframe(me.source('1934-638'))
print me.doframe(me.epoch('utc',qa.unit('today')))
print me.measure(me.baseline('itrf','30deg','40deg','10m'), 'J2000')
#{'m0': {'value': 0.58375325605991979, 'unit': 'rad'},
  # 'm1': {'value': 0.69758519780286155, 'unit': 'rad'},
  # 'm2': {'value': 9.9999999999999964, 'unit': 'm'},
  # 'refer': 'J2000',
# 'type': 'baseline'}
measures.asbaseline.html

**measures.asbaseline - Function**

1.4.1 define a baseline from a position measure

**Description**

asbaseline converts a position measure into a baseline measure. No actual baseline is calculated, since operations can be done on positions, with subtractions to obtain baselines at a later stage.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
</tr>
<tr>
<td>allowed:</td>
</tr>
<tr>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```
print "\t----\t asbaseline Ex 1 \t----"

###An example of getting baselines with 3 antenna positions
### Define the frame; where, which-direction and when
me.doframe(me.observatory('VLA'))
me.doframe(me.direction('J2000', '19h20m00', '20d10m00'))
me.doframe(me.epoch('utc', '2007/07/08/20:30:00'))

##antenna position
ants=me.position('itrf',qa.quantity([3828763.11,3828746.55, 3828727.43], 'm'), qa.quantity([442449.106,442592.14, 442580.12], 'm'), qa.quantity([5064923.01, 5064923.01, 5064923.51], 'm'))
```

1040
bl = me.asbaseline(ants)
print bl

me.expand(bl)

{'return': {'m0': {'value': array([-0.51637894, -0.36575235, 1.50036599])},
            'm1': {'value': array([-0.00060966, 0.00302388, 0.02206414])},
            'm2': {'value': array([143.98943974, 135.78652583, 22.58992696])},
            'refer': 'J2000',
            'type': 'baseline'},
            'xyz': {'value': array([1.25215025e+02, -7.10925354e+01, -8.77850493e-02,
                                    1.26804339e+02, -4.85640980e+01, 4.10601842e-01,
                                    1.58931410e+00, 2.25284374e+01, 4.98386892e-01])}}
measures.listcodes.html

**measures.listcodes - Function**

| 1.4.1 get known reference code names (list indices do not necessarily correspond to enumeration indices) |

**Description**

listcodes will produce the known reference codes for a specified measure type. It will return a record with two entries. The first is a string vector of all normal codes; the second a string vector (maybe empty) with all extra codes (like planets). NOTE: Synonyms and different code groups may be present in the code name lists. The indices in these lists therefore do not necessarily correspond to the internal CASA enumeration indices.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ms</td>
<td>the measure type for which to list</td>
</tr>
<tr>
<td>allowed:</td>
<td>record</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```plaintext
print \"\\t\\t\\tlistcodes Ex 1 \"\\t\\t\\t\\n# Generate some direction
# Note that an empty or non-specified reference code will produce the
# measure with the default code for that measure type
a=me.direction()
print me.getref(a)
#'J2000'
print me.ismeasure(a)
#True
```

1042
# Get the known reference codes for direction
print me.listcodes(a)
#{'normal': ['J2000', 'JMEAN', 'JTRUE', 'APP', 'B1950', 'BMEAN',
# 'BTRUE', 'GALACTIC', 'HADEC', 'AZEL', 'AZELSW', 'AZELNE', 'AZELGEO',
# 'AZELSWGEO', 'AZELNEGEO', 'JNAT', 'ECLIPTIC', 'MECLIPTIC',
# 'TECLIPTIC', 'SUPERGAL', 'ITRF', 'TOPO', 'ICRS'],
# 'extra': ['MERCURY', 'VENUS', 'MARS', 'JUPITER', 'SATURN', 'URANUS',
# 'NEPTUNE', 'PLUTO', 'SUN', 'MOON', 'COMET']}
measures.measure.html

**measures.measure - Function**

convert a measure to another reference

**Description**

measure converts measures (epoch, direction etc.) from one reference to another. It will, for instance, convert a direction from J2000 to AZEL representation.

Its arguments are a measure, an output reference code (see the individual measures for the allowable codes (direction, position, epoch, frequency, doppler, radialvelocity, baseline, uvw, earthmagnetic)), and an optional offset of the same type as the main measure. The offset will be subtracted from the result before it is returned.

In some cases (see the individual measures for when), more information than just a reference code is necessary. E.g. the above example of a conversion to AZEL, needs to know for when, and where on Earth we want it. This information is stored in a reference frame. Measures are set in the reference frame with the doframe function. The frame is tool wide.

**IMPORTANT NOTE:**

To get an accurate conversion of solar system objects direction to a celestial frame, one should convert to AZEL or HADEC before to get parallax accounted for. Thus if you want to get the moon’s position in J2000..one would do it in 2 stages

i.e (after setting the appropriate frames)

moonazel=me.measure(me.direction('moon'), 'AZELGEO')

moonJ2000=me.measure(moonazel, 'J2000')

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>measure to be converted</td>
</tr>
<tr>
<td>allowed: record</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>rf</td>
<td>output reference code</td>
</tr>
<tr>
<td>allowed: string</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>off</td>
<td>optional output offset measure</td>
</tr>
<tr>
<td>allowed: record</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>
print "\t----\\t measure Ex 1 \t----"
a = me.epoch('utc','today') # a time
print a
#{'m0': {'value': 54054.872957673608, 'unit': 'd'},
  # 'refer': 'UTC',
  # 'type': 'epoch'}
print me.doframe(me.source('1934-638'))
print me.measure(a, 'tai') # convert to IAT
#{'m0': {'value': 54054.873339618054, 'unit': 'd'},
  # 'refer': 'TAI',
  # 'type': 'epoch'}
print me.doframe(a) # set time in frame
True
print me.doframe(me.observatory('ALMA')) # set position in frame
True
b=me.direction('j2000', qa.toangle('0h'), '-30deg') # a direction
print b
#{'m0': {'value': 0.0, 'unit': 'rad'},
  # 'm1': {'value': -0.52359877559829882, 'unit': 'rad'},
  # 'refer': 'J2000',
  # 'type': 'direction'}
print me.measure(b, 'azel') # convert to AZEL
#{'m0': {'value': 1.92440968108222324, 'unit': 'rad'},
  # 'm1': {'value': 0.76465385681363052, 'unit': 'rad'},
  # 'refer': 'AZEL',
  # 'type': 'direction'}
print qa.angle(me.getvalue(me.measure(b,'azel'))['m0']) # show as angles
[+'110.15.38']
print qa.angle(me.getvalue(me.measure(b,'azel'))['m1'])
[+'043.48.41']

Another example:

print "\t----\\t measure Ex 2 \t----"
# Fill the frame with necessary information

1045
print me.doframe(me.epoch('utc','today'))
#True
print me.doframe(me.observatory('ALMA'))
#True
print me.doframe(me.direction('mars'))
#True
a=qa.unit('1GHz')
print a
#{'value': 1.0, 'unit': 'GHz'}
m=me.frequency('lsrk',qa.quantity(qa.getvalue(a),qa.getunit(a)))
print m
#{'m0': {'value': 1000000000.0, 'unit': 'Hz'},
  # 'refer': 'LSRK',
  # 'type': 'frequency'}
print me.measure(m,'lsrd')
#{'m0': {'value': 1000001766.3928765, 'unit': 'Hz'},
  # 'refer': 'LSRD',
  # 'type': 'frequency'}
measures.doframe.html

**measures.doframe - Function**

1.4.1 save a measure as frame reference

**Description**

doframe will set the measure specified as part of a frame.
If conversion from one type to another is necessary, with the measure function, the following frames should be set if one of the reference types involved in the conversion is as in the following lists.

*Epoch*
- UTC
- TAI
- LAST position
- LMST position
- GMST1
- GAST
- UT1
- UT2
- TDT
- TCG
- TDB
- TCD

*Direction*
- J2000
- JMEAN epoch
- JTRUE epoch
- APP epoch
- B1950
- BMEAN epoch
- BTRUE epoch
- GALACTIC
- HADEC epoch
- position
- AZEL epoch
- position
- SUPERGALACTIC
- ECLIPTIC
- MECLIPTIC epoch
- TECLIPTIC epoch
- PLANET epoch [position]

*Position*
- WGS84
- ITRF

*Radial Velocity*
- LSRK direction
- LSRD direction
- BARY direction
- GEO direction
- epoch
- TOPO direction
- epoch
- position
- GALACTO direction

*Doppler*
- RADIO
- OPTICAL
- Z
- RATIO
- RELATIVISTIC
- BETA
- GAMMA

*Frequency*
- REST direction
- radialvelocity
- LSRK direction
- LSRD direction
- BARY direction
- GEO direction
- epoch
- TOPO direction
- epoch
- position
- GALACTO

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>measure to be set in frame</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: record</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool
Example

print "\t----\t doframe Ex 1 \t----"

a = me.epoch('utc', 'today') # a time

print a

#{'m0': {'value': 54054.91671484954, 'unit': 'd'},
# 'refer': 'UTC',
# 'type': 'epoch'}

print me.doframe(a) # set time in frame

#True
measures.framenow.html

**measures.framenow - Function**

1.4.1 set the active frame time at now

**Description**

framenow will fill the active frame time with the current date and time. The different frame values necessary are described in the doframe function.

**Arguments**

- **Inputs**

**Returns**

- **bool**

**Example**

```
print \"\t----\t framenow Ex 1 \t----\" 
print me.framenow()  # specify now as frame reference 
#True
print me.showframe()  # and show the current frame 
#\'Frame: Epoch: 54054::22:01:42.2880\'
```
measures.showframe.html

**measures.showframe - Function**

1.4.1 show the currently active frame reference

**Description**

showframe will display the currently active reference frame values on the terminal. The different frame values necessary are described in the doframe function. The frame is displayed on the terminal using the formatting as done for the show function.

**Arguments**

**Inputs**

**Returns**

string

**Example**

```plaintext
print "\t----\t showframe Ex 1 \t----"
print me.doframe(me.epoch('utc','today')) # specify now as frame reference
#T
print me.showframe() # and show the current frame
#'Frame: Epoch: 54054::22:01:42.2880'
```
measures.toradialvelocity.html

**measures.toradialvelocity - Function**

1.4.1 convert a doppler type value to a real radial velocity

**Description**

toradialvelocity will convert a Doppler type value (e.g. in radio mode) to a real radial velocity. The type of velocity (e.g. LSRK) should be specified

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>rf</td>
<td>radial velocity reference type</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>v0</td>
<td>doppler value measure</td>
<td>record</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
print "\t\t\toradialvelocity Ex 1 \t----"

a = me.doppler('radio','0.4')
print a
# Out[4]:
#{'m0': {'value': 119916983.2, 'unit': 'm/s'},
# 'refer': 'RADIO',
# 'type': 'doppler'}

print me.toradialvelocity('topo',a)
#{'m0': {'value': 141078803.7647059, 'unit': 'm/s'},
# 'refer': 'TOPO',
# 'type': 'radialvelocity'}
```

1051
measures.tofrequency.html

**measures.tofrequency - Function**

1.4.1 convert a doppler type value to a frequency

**Description**

tofrequency will convert a Doppler type value (e.g. in radio mode) to a frequency. The type of frequency (e.g. LSRK) and a rest frequency (either as a frequency quantity (e.g. qa.constants('HI')) or a frequency measure (e.g. me.frequency('rest','5100MHz')) should be specified

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>rf</td>
<td>frequency reference type</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>v0</td>
<td>doppler measure value</td>
<td>record</td>
<td></td>
</tr>
<tr>
<td>rfq</td>
<td>rest frequency (frequency measure or frequency quantity)</td>
<td>record</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
print "\t\t tofrequency Ex 1 \t---"
a=me.doppler('radio','0.4')
print a
#{'m0': {'value': 119916983.2, 'unit': 'm/s'},
#  'refer': 'RADIO',
#  'type': 'doppler'}
```

1053
print me.tofrequency('lsrk',a,qa.constants('HI'))
#{'m0': {'value': 852243451.07159996, 'unit': 'Hz'},
# 'refer': 'LSRK',
# 'type': 'frequency'}
measures.todoppler - Function

1.4.1 convert a frequency or radial velocity measure to a doppler measure

Description

todoppler will convert a radial velocity measure or a frequency measure to a doppler measure. In the case of a frequency, a rest frequency has to be specified. The type of doppler wanted (e.g. RADIO) has to be specified.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>rf</td>
<td>doppler reference type</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>v0</td>
<td>radial velocity or frequency measure</td>
<td>record</td>
<td></td>
</tr>
<tr>
<td>rfq</td>
<td>rest frequency (frequency measure or frequency quantity)</td>
<td>any</td>
<td>variant</td>
</tr>
</tbody>
</table>

Returns

record

Example

```
print "\t\t\t todoppler Ex 1 \t\t\t"

f = me.frequency('lsrk','1410MHz')  # specify a frequency
print f
#{'m0': {'value': 1410000000.0, 'unit': 'Hz'},
# 'refer': 'LSRK',
```
# 'type': 'frequency'}
print me.todoppler('radio', f, qa.constants('HI')) # give doppler, using HI rest
#{'m0': {'value': 2196249.8401180855, 'unit': 'm/s'},
# 'refer': 'RADIO',
# 'type': 'doppler'}

1056
measures.torestfrequency.html

**measures.torestfrequency - Function**

1.4.1 convert a frequency and doppler measure to a rest frequency

### Description

torestfrequency will convert a frequency measure and a doppler measure (e.g. obtained from another spectral line with a known rest frequency) to a rest frequency.

### Arguments

| Inputs | | | |
|--------|--------|--------|
| v0     | frequency reference type | allowed: record |
| d0     | doppler measure value     | allowed: record |

### Returns

record

### Example

```plaintext
print "\t\t torestfrequency Ex 1 \t\t"

dp = me.doppler('radio', '2196.24984km/s')  # a measured doppler speed
print dp
#{'m0': {'value': 2196249.8399999999, 'unit': 'm/s'},
# 'refer': 'RADIO'},
# 'type': 'doppler'}

f = me.frequency('lsrk','1410MHz')  # a measured frequency
print f
#{'m0': {'value': 1410000000.0, 'unit': 'Hz'},
# 'refer': 'LSRK'},
```
# 'type': 'frequency'}
print me.torestfrequency(f, dp)  # the corresponding rest frequency
#{'m0': {'value': 1420405751.7854364, 'unit': 'Hz'},
# 'refer': 'REST',
# 'type': 'frequency'}
measures.rise.html

**measures.rise - Function**

1.4.1 get rise and set sidereal time

**Description**

rise will give the rise/set hour-angles of a source. It needs the position in the frame, and a time. If the latter is not set, the current time will be used.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>direction of source (direction measure)</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ev</th>
<th>elevation angle limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant 0.0deg</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
# NOT IMPLEMENTED
print "\t\t\t\t\t\t rise Ex 1 \t----"
print me.rise(me.direction('sun'))
# [rise=[value=267.12445, unit=deg], set=[value=439.029964, unit=deg]]
print qa.form.long(me.rise(me.direction('sun')).rise)
#17:48:29.868
```
measures.riseset.html

**measures.riseset - Function**

1.4.1 get rise and set times

**Description**

rise will give the rise/set times of a source. It needs the position in the frame, and a time. If the latter is not set, the current time will be used. The returned value is a record with a ‘solved’ field, which is F if the source is always below or above the horizon. In that case the rise and set fields will all have a string value. The record also returns a rise and set record, with 'last' and 'utc' fields showing the rise and set times as epochs.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>crd</td>
<td>direction of source (direction measure)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
<td></td>
</tr>
<tr>
<td>ev</td>
<td>elevation limit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: variant 0.0deg</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```
# NOT IMPLEMENTED
print "\t---\t riseset Ex 1 \t----"
print me.riseset(me.direction('sun'))
#[solved=T,  
# rise=[last=[type=epoch, refer=LAST, m0=[value=0.0731388605, unit=d]],  
# utc=[type=epoch, refer=UTC, m0=[value=52085.8964, unit=d]]],  
# set=[last=[type=epoch, refer=LAST, m0=[value=0.455732593, unit=d]],
```

1060
# utc=[type=epoch, refer=UTC, m0=[value=52086.2779, unit=d]]]
print me.riseset(me.direction('sun'), qa.unit('80deg'))
# [solved=F,
# rise=[last=below, utc=below],
# set=[last=below, utc=below]]
print qa.form.long(me.riseset(me.direction('sun')).rise.utc.m0)
#21:30:47.439
#
measures.posangle.html

measures.posangle - Function

1.4.1 get position angle of two directions

Description

posangle will give the position angle from a direction to another. I.e. the angle in a direction between the direction to the North pole and the other direction. The position angle is calculated in the frame of the first argument. m2 is thus converted to the frame of m1 before calculating the position angle.

Arguments

Inputs

m1 direction of source (direction measure)
   allowed: record
   Default:

m2 direction of other source (direction measure)
   allowed: record
   Default:

Returns

record

Example

```python
print "\t----\t posangle Ex 1 \t----"
a=me.direction('j2000','0deg','70deg')
b=me.direction('j2000','0deg','80deg')
print me.posangle(a,b)
#{'value': -0.0, 'unit': 'deg'}
print me.separation(a,b)
#{'value': 9.9999999999999893, 'unit': 'deg'}
tim=me.epoch('utc','today')
print me.doframe(tim)
```
# True
pos = me.observatory('ATCA')
print me.doframe(pos)
# True
print me.posangle(a, b)
#{'value': -0.0, 'unit': 'deg'}

### Example of how to calculate the parallactic angle of a given direction on the sky.

### set the frames and epoch
measures.separation.html

**measures.separation - Function**

1.4.1 get separation angle between two directions

**Description**

separation will give the separation of a direction from another as an angle.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>direction of source (direction measure)</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td>allowed: record</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>m2</td>
<td>direction of other source (direction measure)</td>
</tr>
<tr>
<td></td>
<td>allowed: record</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
print "\t\t separation Ex 1 \t---"
me.direction('j2000','0deg','70deg')
me.direction('j2000','0deg','80deg')
me.separation(a,b)
#{'value': 9.999999999999993, 'unit': 'deg'}
tim = me.epoch('utc','today') # set the time
me.doframe(tim)
#True
me.observatory('ATCA') # set where
me.doframe(pos)
#True
c=me.measure(b,'azel') # try with different type
```

1064
print me.separation(a,c)
#{'value': 10.000000000062277, 'unit': 'deg'}

### the example below is how to calculate
### the parallactic angle
me.doframe(me.epoch('utc','2015/06/30/19:30:40'))
me.doframe(me.observatory('ALMA'))
mydir = me.direction('J2000','17h28m00','-28d00m00' )
#convert direction to AZEL
mydirazel=me.measure(mydir, 'AZEL')
hadecpol=me.direction('HADEC', '00h00m00', '90d00m00')
### no need to convert north pole direction to AZEL
### as it will converted to the frame of mydirazel
parAngle=me.posangle(mydirazel, hadecpol)
measures.addxvalue.html

**measures.addxvalue - Function**

1.4.1 get some additional measure information

**Description**

addxvalue will give some additional information about some measures as a vector of quantities. It is used internally to get the rectangular coordinates of measures that are normally given in angles. The casual user will probably in general not interested in this function.

**Arguments**

**Inputs**

<table>
<thead>
<tr>
<th>measures for which extra information is to be gotten</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed: record</td>
</tr>
<tr>
<td>Default: record</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```plaintext
print "\t----\t addxvalue Ex 1 \t----"
a=me.observatory('atca')
print a
# {'m0': {'value': 2.6101423190348916, 'unit': 'rad'},
# 'm1': {'value': -0.5261379196128062, 'unit': 'rad'},
# 'm2': {'value': 6372960.2577234386, 'unit': 'm'},
# 'refer': 'ITRF',
# 'type': 'position'}
print me.addxvalue(a)
# {'value': [-4750915.8370000012, 2792906.1819999996, -3200483.747], 'unit': 'm'}
print me.addxvalue(me.epoch('utc','today'))
#{}
```

1066
measures.type.html

**measures.type** - Function

1.4.1 type of tool

Description

type will return the tool name.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
</table>

Returns

string

Example

```python
print "\t----\t type Ex 1 \t----"
print me.type()
# 'measures'
```

1068
measures.done.html

**measures.done - Function**

1.4.1 Free resources used by tool.

**Description**

In general you will not want to call this method. It removes and then recreates the default measures tool.

**Arguments**

**Inputs**

**Returns**

bool

**Example**

```python
print "\t----\t done Ex 1 \t----"
print me.done() #True
```

1069
measures.ismeasure.html

measures.ismeasure - Function

1.4.1 Check if measure

Description

Checks if the operand is a correct measure

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>value to be tested</td>
</tr>
<tr>
<td>allowed: record</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool

Example

print "\t----\t ismeasure Ex 1 \t----"
x=me.epoch('utc','today')
print x
#{'m0': {'value': 54056.043754386577, 'unit': 'd'},
# 'refer': 'UTC',
# 'type': 'epoch'}
print me.ismeasure(x)
#True
y=me.getvalue(x)
print y
#{'m0': {'value': 54056.043754386577, 'unit': 'd'}}
print me.ismeasure(y)
#False
print "Last example, exiting!"
exit()
1.5 quanta - Module

Units and quantities handling

**Description**  
*Introduction*

A quantity is a value with a unit. For example, '5km/s', or '20Jy/pc2'. This module (the `quanta` module) enables you to create and manipulate such quantities. The types of functionality provided are:

- **Conversion** - Conversion of quantities to different units
- **Calculation** - Calculations with quantities

The Quanta tool manipulates quantities. A quantity is stored as a record with two fields. These fields are named 'value' and 'unit'. As well as simple scalar quantities, one can also create quantities as vectors or arrays. For example, you may have a vector of values, which all have the same unit - there is no need to store a copy of the unit for each value. Access to the individual fields of a quantum should always be by using the `getvalue` and `getunit` methods, especially since the internal names can change or be not accessible at some stage.

**Example**

```
#"
#
print "\t----\t Module Ex 1 \t----"
print qa.quantity(5.4, 'km/s')
#{'value': 5.4000000000000004, 'unit': 'km/s'}
q1 = qa.quantity([8.57132661e+09, 1.71426532e+10], 'km/s')
print qa.convert(q1, 'pc/h');
#{'value': array([ 1., 2.]), 'unit': 'pc/h'}
#"
```

In the first example, we make a simple scalar quantity. You can see that the quantity (which is actually a record) has fields 'value' and 'unit'.

In the second example, we make a vector quantity and then convert it from units of km/s to pc/h.
Example

""
#
print "\t----\t Module Ex 2 \t----"
p1 = qa.quantity('5.4km/s')
#{'value': 5.4000000000000004, 'unit': 'km/s'}
p2 = qa.quantity(qa.unit('5.4km/s'))
#{'value': 5.4000000000000004, 'unit': 'km/s'}
#
""

In the first example, the value and unit were combined into one string (just
saves a bit of typing). The second example shows that the function unit is an
alias for quantity, and that you can create a quantity from another quantity.

Example

""
#
print "\t----\t Module Ex 3 \t----"
q1 = qa.unit("5s 5.4km/s")
print len(q1)
#2
print q1["*0"]
#{'unit': 's', 'value': 5.0}
print q1["*1"]
#{'unit': 'km/s', 'value': 5.4000000000000004}
#

Here we make a vector quantity by using the string vector. You can see that
the resultant quantity record is of length 2 and that each field of that vector
quantity is a scalar quantity. So you see that 'q1' itself does not have fields
'value' and 'unit', only the elements of 'q1' have that.

Example

""
#
print "\t----\t Module Ex 4 \t----"
q1 = qa.unit('5km');
q2 = qa.unit('200m');
print qa.canonical(qa.add(q1,q2))
#{'value': 5200.0, 'unit': 'm'}
#
""

Here we make two quantities with consistent but different units, add them
together and then convert the result to canonical units.
Example

```python
#
print "\t----\t Module Ex 5 \t----"
q1 = qa.quantity('6rad');
q2 = qa.quantity('3deg');
print qa.compare(q1,q2)
#True
print qa.compare(q1,qa.unit('3km'))
#False
#
```

Here we compare the dimensionality of the units of two quantities.

Constants, time and angle formatting

If you would like to see all the possible constants known to the Quanta tool you can issue the command `print qa.map('const')`. You can get the value of any constant in that list with a command such as

```python
#
print "\t----\t Module Ex 6 \t----"
boltzmann = qa.constants('k')
print 'Boltzmann constant is ', boltzmann
#Boltzmann constant is {'value': 1.3806577987510647e-23, 'unit': 'J/K'}
#
```

There are some extra handy ways you can manipulate strings when you are dealing with times or angles. The following list shows special strings and string formats which you can input to the `quantity` function. Something in square brackets is optional. There are examples after the list.

- time: `[+-]hh:mm:ss.t...` – This is the preferred time format (trailing fields can be omitted)
- time: `[+-]hhH:mm:ss.S` – This is an alternative time format (HMS case insensitive, trailing second fields can be omitted)
- angle: `[+-]dd:mm:ss.t...` – This is the preferred angle format (trailing fields after second period can be omitted; dd.. is valid)
- angle: `[+-]ddD:mm:ss.S` – This is an alternative angle format (DMS case insensitive, trailing fields can be omitted after M)
- today – The special string “today” gives the UTC time at the instant the command was issued.
• today/time – The special string “today” plus the specified time string gives the UTC time at the specified instant

• yyyy/mm/dd[time] – gives the UTC time at the specified instant

• dd[-]mmm[-]ccyy[/time] – gives the UTC time at the specified instant in calendar style notation (23-jun-1999)

Note that the standard unit for degrees is ’deg’, and for days ’d’. Formatting is done in such a way that it interprets a ’d’ as degrees if preceded by a value without a period and if any value following it is terminated with an ’m’. In other cases 'days' are assumed. Here are some examples.

```python
# print "\t----\t Module Ex 7 \t----"
print qa.quantity('today')
#{'value': 54178.87156457176, 'unit': 'd'}
print qa.quantity('5jul1998')
#{'value': 50999.0, 'unit': 'd'}
print qa.quantity('5jul1998/12:')
#{'value': 50999.5, 'unit': 'd'}
print qa.quantity('-30.12.2')
#{'value': -30.200555555555557, 'unit': 'deg'}
print qa.quantity('2:2:10')
#{'value': 30.541666666666668, 'unit': 'deg'}
print qa.unit('23h3m2.2s')
#{'value': 345.75916666666666, 'unit': 'deg'}
#
```

Angles and times can often be used interchangeably. Special functions (qa.totime() and qa.toangle()) are available to make them in the right units for the purpose. E.g. qa.sin(time) gives an error, whereas qa.sin(qa.toangle(time)) works ok. See the [map] function for pre-defined units.

```python
# print "\t----\t Module Ex 8 \t----"
a = qa.quantity('today'); # 1
print a
#{'value': 54178.871564641202, 'unit': 'd'}
b = qa.toangle(a); # 2
print b
#{'value': 340415.88977452344, 'unit': 'rad'}
print qa.angle(qa.norm(qa.toangle(a))); # 3
#-046.14.12
1074
```
print qa.angle(qa.norm(qa.toangle(a), 0));  # 4
#313.45.48
print qa.sub('today', a);  # 5
#{'value': 1.1576048564165831e-08, 'unit': 'd'}
#
print "Last example! Exiting ..."
exit()
1.5.1 quanta - Tool

quanta tool handles units and quantities

Requires:

Synopsis

Methods

- `convertfreq` convert a frequency quantity to another unit
- `convertdop` convert a doppler velocity quantity to another unit
- `quantity` make a quantity from a string or from a numeric value and a unit string
- `getvalue` get the internal value of a quantity
- `getunit` get the internal unit of a quantity
- `canonical` get canonical value of quantity
- `canon` get canonical value of quantity
- `convert` convert a quantity to another unit
- `define` define a new unit name
- `map` list known unit names and constants
- `maprec` create record containing list of known unit names and constants
- `fits` define some FITS units
- `angle` show an angle as a formatted string
- `time` show a time (or date) as a formatted string
- `add` add quantities
- `sub` subtract quantities
- `mul` multiply quantities
- `div` divides quantities
- `neg` negate quantities
- `norm` normalise angle
- `le` compare quantities
- `lt` compare quantities
- `eq` compare quantities
- `ne` compare quantities
- `gt` compare quantities
- `ge` compare quantities
- `sin` sine of quantity
- `cos` cosine of quantity
- `tan` tangent of quantity
- `asin` arcsine of quantity
- `acos` arccosine of quantity
- `atan` arctangent of quantity
- `atan2` arctangent of two quantity
- `abs` absolute value of quantity
- `ceil` ceil value of quantity
floor  floor value of quantity
log   logarithm of quantity
log10 logarithm of quantity
exp   exponential of quantity
sqrt  square root of quantity
compare compare dimensionality of units
check check for proper unit string
checkfreq check for proper frequency unit
pow   raise quantity to power
constants get a constant
isangle check if valid angle or time quantity
totime convert an angle (or a time) to a time
toangle convert a time (or an angle) to an angle
splitdate split a date/time into a record
tos   convert quantity to string
type  type of tool
done Free resources used by tool. Current implementation ignores input parameter, does nothing and returns true
unit
isquantity Check if quantity
setformat set format for output of numbers. (NOT IMPLEMENTED YET!)
getformat get current output format (NOT IMPLEMENTED YET!)
formxxx Format a quantity using given format, allowed are hms, dms, deg, rad, +deg.
convertfreq converts a frequency quantity to another unit.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>quantity to convert</td>
<td>variant</td>
<td>1.0</td>
</tr>
<tr>
<td>outunit</td>
<td>unit to convert to</td>
<td>string</td>
<td>Hz</td>
</tr>
</tbody>
</table>

Returns
record

Example

```
# print "\t----\t convertfreq Ex 1 \t----"
print qa.convertfreq('5GHz','cm')
#{'value': 5.9958491599999997, 'unit': 'cm'}
print qa.convertfreq('5cm','GHz')
#{'value': 5.9958491599999997, 'unit': 'GHz'}
#```
**quanta.convertdop - Function**

1.5.1 convert a doppler velocity quantity to another unit

**Description**

`convertfreq` converts a velocity quantity to another unit. Units are either velocity or dimensionless.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>quantity to convert</td>
</tr>
<tr>
<td></td>
<td>allowed: variant</td>
</tr>
<tr>
<td></td>
<td>Default: 0.0</td>
</tr>
<tr>
<td>outunit</td>
<td>unit to convert to</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: km/s</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python

# print "\t----\t convertdop Ex 1 \t----"
print qa.convertdop('1','km/s')
#{"value": 299792.45799999998, 'unit': 'km/s'}
print qa.convertdop('10km/s','1')
#{"value": 3.3356409519815205e-05, 'unit': '1'}
```

1079
**quanta.quantity** - Function

1.5.1 make a quantity from a string or from a numeric value and a unit string

**Description**

quantity makes a quantity from a string, or from a value and a string. Note that a function unit exists which is a synonym for quantity. If only a string is given, it can be a scalar string. The result will be a scalar quantity. If a numeric value and a unit string are given, the numeric value can be any numeric type, and can also be a vector of numeric values. print qa.map() to get a list of recognized units. ‘d’ is usually days, but can be degrees (see example).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>quantity or numeric or string to convert to quantity</td>
</tr>
<tr>
<td>unitname</td>
<td>unit string if v numeric</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>allowed</th>
<th>variant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>allowed</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```
# print "\t----\t quantity Ex 1 \t----"

# make quantity

print tu # also accepts a quantity

print qa.quantity(tu) # also accepts a quantity
```
tu = qa.quantity('1Jy') # make quantity with synonym
print tu
print qa.quantity(-1.3, 'Jy') # make quantity with separate value
q1 = qa.quantity([8.57132661e+09, 1.71426532e+10], 'km/s') # Composite unit
print q1
q = qa.quantity('5d'); print q
q = qa.quantity('5 d'); print q # even if there’s a space, as of 5/28/09
q = qa.quantity('5d30m'); print q
q = qa.quantity('5d30s'); print q # Unless followed by an m!
qa.quantity('5d30s') # WRONG
# {'unit': 'd30s', 'value': 5.0} # I told you...
qa.quantity('5d0m30s') # OK
# {'unit': 'deg', 'value': 5.0083333333333337}

""

1081
**quanta.getvalue - Function**

get the internal value of a quantity

---

**Description**

getvalue returns the internal value of a quantity. It also can handle an array of quantities.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>quantity</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>variant</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

doubleArray

---

**Example**

```
# getvalue Ex 1
print "\t----\t getvalue Ex 1 \t----"
tu = qa.quantity(-1.3, 'Jy')   # make quantity
print tu
#{'value': -1.3, 'unit': 'Jy'}
print qa.getvalue(tu)
-1.3
print qa.getunit(tu)
#Jy
a = qa.quantity([3,5],'cm')   
print a
#{'value': array([ 3., 5.]), 'unit': 'cm'}
print qa.getvalue(a)
```

1082
# [3.0, 5.0]
#
##
quanta.getunit.html

**quanta.getunit - Function**

1.5.1 get the internal unit of a quantity

**Description**

getunit returns the internal unit string of a quantity

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>quantity</td>
</tr>
<tr>
<td>allowed:</td>
<td>variant</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

string

**Example**

```
#  print "\t----\t getunit Ex 1 \t----"
tu = qa.quantity(-1.3, 'Jy')    # make quantity
print tu
#{'value': -1.3, 'unit': 'Jy'}
print qa.getvalue(tu)
#-1.3
print qa.getunit(tu)
#Jy
#  
```
**quanta.canonical - Function**

1.5.1 get canonical value of quantity

**Description**

canonical (with alias canon) gets the canonical value of a quantity

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>v value to convert</td>
</tr>
<tr>
<td>allowed: variant</td>
</tr>
<tr>
<td>Default: 1.0</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
print "\t----\t canonical Ex 1 \t----"
print qa.canonical('1Jy') # canonical value of a string
#{'value': 1e-26, 'unit': 'kg.s-2'}
print qa.canon(qa.quantity('1Jy')) # canonical value of a unit
#{'value': 1e-26, 'unit': 'kg.s-2'}
#
```
quanta.canon.html

**quanta.canon** - Function

1.5.1 get canonical value of quantity

Description

canon gets the canonical value of a quantity

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>value to convert</td>
</tr>
</tbody>
</table>

allowed: variant

Default: 

Returns

record

Example

```python
""
#
print "\t----\t canon Ex 1 \t----"
print qa.canon('1Jy') # canonical value of a string
#{'value': 1e-26, 'unit': 'kg.s-2'}
print qa.canonical(qa.quantity('1Jy')) # canonical value of a unit
#{'value': 1e-26, 'unit': 'kg.s-2'}
#
""
```
quanta.convert.html

quanta.convert - Function
1.5.1 convert a quantity to another unit

Description
convert converts a quantity to another unit. If no output unit given, conversion is to canonical units

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>quantity to convert</td>
</tr>
<tr>
<td></td>
<td>allowed: variant</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>outunit</td>
<td>unit to convert to</td>
</tr>
<tr>
<td></td>
<td>allowed: variant</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns
record

Example

```python
""
#
print "\t\tconvert Ex 1 \t----"
tu = qa.quantity('5Mm/s') # specify a quantity
print tu
#{'value': 5.0, 'unit': 'Mm/s'}
print qa.convert(tu, 'pc/a') # convert it to parsec per year
#{'value': 0.0051135608266237404, 'unit': 'pc/a'}
print qa.convert(tu) # convert to canonical units
#{'value': 5000000.0, 'unit': 'm.s-1'}
# ```

1087
Description

define defines the name and value of a user defined unit

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>name of unit to define</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
</tr>
<tr>
<td>v</td>
<td>variant</td>
</tr>
</tbody>
</table>

Returns

bool

Example

"""
#
print "\t----\t define Ex 1 \t----"
print qa.define('JY','1Jy') # your misspelling #True
print qa.define('VLAunit', '0.898 JY') # a special unit using it #True
print qa.quantity('5 VLAunit') # check its use #{'value': 5.0, 'unit': 'VLAunit'}
print qa.convert('5 VLAunit','Jy') #{'value': 4.490000000000002, 'unit': 'Jy'}
#
quanta.map.html

quanta.map - Function

|1.5.1| list known unit names and constants

Description

map lists the known mapping of units and constants. It has a single argument, which can be a coded string (no-case, minimax match):

**all** all of the following units (not constants): also the default

**Prefix** known decimal prefixes

**SI** known SI units

**Customary** a set of customary units known to programs

**User** units defined by the user

**Constants** known constants (note: only 'const', 'Const', 'constants' and 'Constants' recognised).

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
</table>
| v | type of information to list - coded string  
|    | allowed: string  
|    | Default: all  

Returns

string

Example

""

#

print "\t----\t map Ex 1 \t----"

1091
print qa.map('pre') # list decimal prefixes
#
# == Prefix ===  20 ====
#
# E  (exa)     1e+18
# G  (giga)    1000000000
# M  (mega)    1000000
# P  (peta)    1e+15
# T  (tera)    1e+12
# Y  (yotta)   1e+24
# Z  (zetta)   1e+21
# a  (atto)    1e-18
# c  (centi)   0.01
# d  (deci)    0.1
# da (deka)    10
# f  (femto)   1e-15
# h  (hecto)   100
# k  (kilo)    1000
# m  (milli)   0.001
# n  (nano)    1e-09
# p  (pico)    1e-12
# u  (micro)   1e-06
# y  (yocto)   1e-24
# z  (zepto)   1e-21

print qa.map('Constants') # list known constants
#
# == Constants ====
#
# pi  3.14159
# ee  2.71828
# c  light vel.  2.99792e+08 m/s
# G  grav. const  6.67259e-11 N.m2/kg2
# h  Planck const  6.62608e-34 J.s
# HI HI line     1420.41 MHz
# R  gas const    8.31451 J/K/mol
# NA Avogadro #  6.02214e+23 mol-1
# e  electron charge  1.60218e-19 C
# mp proton mass   1.67262e-27 kg
# mp_me mp/me     1836.15
# mu0 permeability vac.  1.25664e-06 H/m
# eps0 permittivity vac.  1.60218e-19 C
# k  Boltzmann const  1.38066e-23 J/K
# F  Faraday const   96485.3 C/mol
# me electron mass    9.10939e-31 kg
# re electron radius   2.8179e-15 m
# a0 Bohr's radius     5.2918e-11 m
# R0 solar radius      6.9599e+08 m
# k2 IAU grav. const^2  0.000295912 AU3/d2/S0
quanta.maprec.html

**quanta.maprec - Function**

1.5.1 create record containing list of known unit names and constants

**Description**

maprec returns a record with the known mapping of units and constants. It has a single argument, which can be a coded string (no-case, minimax match):

**all** all of the following units (not constants): also the default

**Prefix** known decimal prefixes

**SI** known SI units

**Customary** a set of customary units known to programs

**User** units defined by the user

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>type of information to list - coded string</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: all</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
""" # print "\t----\t maprec Ex 1 \t----"
p = qa.maprec('pre') # list decimal prefixes
print p['Prefix_G']
"""
```

1094
# G (giga) 1000000000
s = qa.maprec('SI')  # list SI units
print s['SI_Jy']
# Jy (jansky) 1e-26 kg.s-2
#
"""
quanta.fits.html

**quanta.fits - Function**

1.5.1 define some FITS units

**Description**

fits defines some unit names used in reading and writing FITS files.

**Arguments**

**Inputs**

**Returns**

bool

**Example**

"""
#
print "\t----\t fits Ex 1 \t----"
print qa.fits()
#True
print qa.map('user')
#    == User ====
#    BEAM   (dimensionless beam) 1 _
#    DAYS   (day) 86400 s
#    DEG    (degree) 0.0174532925199 rad
#    DEGREES (degree) 0.0174532925199 rad
#    HZ     (hertz) 1 s-1
#    JY     (jansky) 1e-26 kg.s-2
#    KELVIN (kelvin) 1 K
#    KELVINS (kelvin) 1 K
#    KM     (km) 1000 m
#    M      (meter) 1 m
"""
# METERS (meter) 1 m
# PASCAL (pascal) 1 m⁻¹.kg.s⁻²
# PIXEL (dimensionless pixel) 1 _
# S (second) 1 s
# SEC (second) 1 s
# SECONDS (second) 1 s
# VOLTS (volt) 1 m².kg.s⁻³.A⁻¹
# YEAR (year) 31557600 s
# YEARS (year) 31557600 s
#
quanta.angle.html

**quanta.angle - Function**

1.5.1 show an angle as a formatted string

**Description**

angle converts an angle quantity to a formatted string. The formatting information is a precision (0 is default, 6 includes +-ddd.mm.ss) and a string array of codes (no-case, minimax match): Codes include:

- **clean** delete leading/trailing superfluous separators
- **no_d** do not show degrees part
- **no_dm** do not show degrees and minutes part
- **dig2** show only 2 digits of degrees in angle format
- **time** show as time (hh:mm:ss.ttt) rather than as angle

If a multi-dimensional value is given for the value $v$, the returned value is a string vector of a length equal to last dimension. Each string has a number of fields equal to the number of elements in all earlier dimensions. If the $showform$ is $T$, each vector element is surrounded by a pair of square brackets if there is more than one entry, and fields are separated by a ‘,’.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>v</strong></td>
<td>angle quantity value to output</td>
</tr>
<tr>
<td><strong>prec</strong></td>
<td>number of digits shown</td>
</tr>
<tr>
<td><strong>form</strong></td>
<td>formatting information in coded string array</td>
</tr>
<tr>
<td><strong>showform</strong></td>
<td>show square brackets and separating ,</td>
</tr>
<tr>
<td></td>
<td>allowed: variant</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
<tr>
<td></td>
<td>allowed: stringArray</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: false</td>
</tr>
</tbody>
</table>
Returns
stringArray

Example

""
#
print "\t----\t angle Ex 1 \t----"
tu = qa.quantity('5.7.12.345678')  # define an angle
print tu
#{'value': 5.1200960216666669, 'unit': 'deg'}
print qa.angle(tu)  # default output
#+005.07.12
print qa.angle(tu, prec=7)  # 7 digits
#+005.07.12.3
print qa.angle(tu, prec=4)  # 4 digits
#+005.07.
print qa.angle(tu, form=['tim','no_d'])  # as time, no hours shown
#:20:29
#
""
quanta.time.html

quanta.time - Function

1.5.1 show a time (or date) as a formatted string

Description

time converts a time quantity to a formatted string. The formatting information is a precision (0 is default, 6 includes hh:mm:ss) and a string array of codes (no-case, minimax match): Codes include:

clean delete leading/trailing superfluous separators

no_d do not show hours part

no_dm do not show hours and minutes part

ymd include a date as yyyy/mm/dd (date is by default not shown)

dmy include a date as ddMMMyyyy (date is by default not shown)

mjd include a date as Modified Julian Day (date is by default not shown)

fits include a date and show time in FITS format: le from OS

angle show in angle (dd.mm.ss.ttt) rather than time format

day prefix day-of-week to output

local show local time rather than UTC (add timezone offset)

no_time suppress printing of time part

If a multi-dimensional value is given for the value v, the returned value is a string vector of a length equal to last dimension. Each string has a number of fields equal to the number of elements in all earlier dimensions. If the showform is T, each vector element is surrounded by a pair of square brackets if there is more than one entry, and fields are separated by a ‘,’.

Arguments

1100
Inputs

v 
  time quantity value to output 
  allowed: variant 
  Default: 

prec 
  number of digits shown 
  allowed: int 
  Default: 0 

form 
  formatting information in coded string array 
  allowed: stringArray 
  Default: 

showform 
  show square brackets and separating , 
  allowed: bool 
  Default: false 

Returns

stringArray

Example

"""
#
print "\t----\t time Ex 1 \t----"
tu = qa.quantity('today')  # a time
print tu
#{'value': 54175.708981504627, 'unit': 'd'}
print qa.time(tu)  # default format
#17:00:56
print qa.time(tu,form="dmy")  # show date
#16-Mar-2007/17:00:56
print qa.time(tu,form=["ymd","day"])  # and day
#Fri-2007/03/16/17:00:56
print qa.time(tu,form="fits")  # FITS format
#2007-03-16T17:00:56
print qa.time(tu,form=["fits","local"])  # local FITS format
#2007-03-16T10:00:56-07:00
print qa.time(tu,form=["ymd","local"])  # local time
#2007/03/16/10:00:56
#
"""
quanta.add.html

**quanta.add - Function**

1.5.1 add quantities

### Description

add adds two quantities

### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>allowed: variant</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>value</td>
</tr>
<tr>
<td>allowed: variant</td>
<td></td>
</tr>
<tr>
<td>Default: 0</td>
<td></td>
</tr>
</tbody>
</table>

### Returns

record

### Example

```ruby
###
#
print "\t----\t add Ex 1 \t----"
print qa.add('5m', '2yd')
#{'value': 6.8288000000000002, 'unit': 'm'}
#
###
```
quanta.sub.html

quanta.sub - Function

1.5.1 subtract quantities

Description

sub subtracts two quantities

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>value</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

allowed: variant

Default: variant 0

Returns

record

Example

```
###
#
print "\t----\t sub Ex 1 \t----"
print qa.sub('5m', '2yd')
#{'value': 3.1712000000000002, 'unit': 'm'}
#
###
```
quanta.mul.html

**quanta.mul - Function**

1.5.1 multiply quantities

**Description**

mul multiplies two quantities

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>value</th>
<th>allowed: variant</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>value</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
""
#
print "\t---\t mul Ex 1 \t---"
print qa.mul('5m', '3s')
#{'value': 15.0, 'unit': 'm.s'}
#
""
```
quanta.div.html

**quanta.div - Function**

1.5.1 divides quantities

### Description

div divides two quantities

### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
</tr>
<tr>
<td>a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>allowed</th>
<th>variant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>allowed</th>
<th>variant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>1</td>
</tr>
</tbody>
</table>

### Returns

record

### Example

```python
#
print "\t----\t div Ex 1 \t----"
print qa.div('5m', '3s')
#{'value': 1.6666666666666667, 'unit': 'm/(s)'}
#
```
Negates a quantity

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>v value</td>
</tr>
<tr>
<td>allowed: variant</td>
</tr>
<tr>
<td>Default: 1</td>
</tr>
</tbody>
</table>

Returns

record

Example

```python

#
print "\t----\t neg Ex 1 \t----"
print qa.neg('5m')
#{'value': -5.0, 'unit': 'm'}
#
```
quanta.norm.html

**quanta.norm - Function**

1.5.1 normalise angle

**Description**

norm normalise angles in interval of $2\pi$ radians. The default interval is from -0.5 to +0.5 of a full interval (i.e. from -180 to +180 degrees). The lower end of the interval can be set as a fraction of $2\pi$

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>angle</td>
<td>quantity</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>variant</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>lower</td>
<td>interval boundary</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td>-0.5</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
# print "\t----\t norm Ex 1 \t----"
print qa.norm('713deg') #default normalisation
#{'value': -6.9999999999999716, 'unit': 'deg'}
print qa.norm('713deg', -2.5) # normalise to interval -900 - -540 deg
#{'value': -727.0, 'unit': 'deg'}
#```

1108
quanta.le.html

**quanta.le - Function**

1.5.1 compare quantities

**Description**

le compares two quantities for less than or equal.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>v</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>allowed: any</td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
<td>Default: variant 0</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```
"
#
print "\t----\t le Ex 1 \t----"
print qa.le('5m', '2yd')
#False
#"
```
**quanta.lt.html**

**quanta.lt - Function**

1.5.1 compare quantities

**Description**

lt compares two quantities for less than.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>v value</th>
<th>allowed: any</th>
<th>Default: variant</th>
</tr>
</thead>
<tbody>
<tr>
<td>a value</td>
<td></td>
<td>allowed: any</td>
<td>Default: variant 0</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```
###
#
print "\t----\t lt Ex 1 \t----"
print qa.lt('5m', '2yd')
#False
#
###
```
**quanta.eq - Function**

Description

`eq` compares two quantities for equality.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>value</th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td></td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>a</td>
<td></td>
<td>any</td>
<td>variant 0</td>
</tr>
</tbody>
</table>

Returns

`bool`

Example

```python
"""
#
print "\t----\t eq Ex 1 \t----"
print qa.eq('5m', '2yd')
#False
#"""
quanta.ne.html

**quanta.ne - Function**

1.5.1 compare quantities

**Description**

ne compares two quantities for non equality.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **value**
  - allowed: any
  - Default: variant
  - allowed: any
  - Default: variant 0

**Returns**

bool

**Example**

```plaintext
###
#
print "\t----\t ne Ex 1 \t----"
print qa.ne('5m', '2yd')
#True
#
###
```
quanta.gt.html

**quanta.gt - Function**

1.5.1 compare quantities

**Description**

gt compares two quantities for greater than.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>any</td>
</tr>
<tr>
<td>a</td>
<td>any</td>
</tr>
</tbody>
</table>

Default:

- variant
- variant 0

**Returns**

bool

**Example**

```python
###
#
print "\t----\t gt Ex 1 \t----"
print qa.gt('5m', '2yd')
#True
#
###
```
**quanta.ge.html**

**quanta.ge - Function**

1.5.1 compare quantities

---

**Description**

ge compares two quantities for greater than or equal.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>value</td>
</tr>
<tr>
<td>a</td>
<td>value</td>
</tr>
</tbody>
</table>

- **allowed:** any
- **Default:** variant
- **allowed:** any
- **Default:** variant 0

**Returns**

bool

**Example**

```python
""
#
print "\t----\t ge Ex 1 \t----"
print qa.ge('5m', '2yd')
#True
#
""
```
quanta.sin.html

quanta.sin - Function

1.5.1 sine of quantity

Description

sin gives sine of angle quantity

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>angle quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
</tbody>
</table>

Returns

record

Example

""
#
print "\t----\t sin Ex 1 \t----"
print qa.sin('7deg')
#{'value': 0.12186934340514748, 'unit': ''}
#
"""
**quanta.cos** - Function

1.5.1 cosine of quantity

**Description**

cos gives cosine of angle quantity

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>angle quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```
print "\t----\t cos Ex 1 \t----"
print qa.cos('7deg')
#{'value': 0.99254615164132198, 'unit': ''}
```

1117
**quanta.tan - Function**

1.5.1 tangent of quantity

**Description**

tan gives tangent of angle quantity

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>angle quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
###
#
print "\t----\t tan Ex 1 \t----"
print qa.tan('7deg')
#{'value': 0.1227845609029046, 'unit': ''}
#
###
```

```html

```
**quanta.asin - Function**

1.5.1 arcsine of quantity

**Description**

asin gives arcsine of non-dimensioned quantity

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>non-dimensioned quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>any</td>
</tr>
<tr>
<td></td>
<td>variant</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
""
#
print "\t----\t asin Ex 1 \t----"
print qa.convert(qa.asin(qa.sin('7deg')), 'deg')
#{'value': 7.0, 'unit': 'deg'}
#
"""
```
**quanta.acos.html**

**quanta.acos - Function**

1.5.1 arccosine of quantity

**Description**

acos gives arccosine of non-dimensioned quantity

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>non-dimensioned quantity</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python

# """
# print "\t----\t acos Ex 1 \t----"
print qa.convert(qa.acos(qa.cos('7deg')), 'deg')
#{'value': 7.0000000000000249, 'unit': 'deg'}
# """
```

1120
quanta.atan.html

**quanta.atan - Function**

1.5.1 arctangent of quantity

**Description**

atan gives arctangent of non-dimensioned quantity

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
</tr>
<tr>
<td>allowed:</td>
</tr>
<tr>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```
""
#
print "\t----\t atan Ex 1 \t----"
print qa.convert(qa.atan(qa.tan('7deg')), 'deg')
#{'value': 7.0, 'unit': 'deg'}
#
"""
```
quanta.atan2.html

**quanta.atan2 - Function**

1.5.1 arctangent of two quantity

**Description**

atan gives arctangent of two non-dimensioned quantity

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>non-dimensioned quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>

- **allowed:** any
- **Default:** variant

**Returns**

record

**Example**

```
""
#
print "\t----\t atan2 Ex 1 \t----"
print qa.convert(qa.atan2(qa.sin('7deg'), qa.cos('7deg')), 'deg')
#{'value': 7.0, 'unit': 'deg'}
#
""
```
quanta.abs.html

**quanta.abs - Function**

1.5.1 absolute value of quantity

**Description**

abs gives absolute value of quantity

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```plaintext
""
#
print "\t----\t abs Ex 1 \t----"
print qa.abs('-5km/s')
#{'value': 5.0, 'unit': 'km/s'}
#
""
```
**quanta.ceil - Function**

**1.5.1 ceil value of quantity**

**Description**

ceil gives ceiling value of quantity

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>value</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td>variant</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```
###
#
print "\t----\t ceil Ex 1 \t----"
print qa.ceil('5.1AU')
#{'value': 6.0, 'unit': 'AU'}
#
###
```
quanta.floor.html

quanta.floor - Function

1.5.1 floor value of quantity

Description

floor gives flooring value of quantity

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
</tbody>
</table>

Returns

record

Example

```python
"""
#
print "\t----\t floor Ex 1 \t----"
print qa.floor('-5.1AU')
#{'value': -6.0, 'unit': 'AU'}
#
"""
```

1125
quanta.log.html

**quanta.log - Function**

1.5.1 logarithm of quantity

**Description**

log gives natural logarithm of dimensionless quantity

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>dimensionless quantity</td>
<td></td>
</tr>
<tr>
<td>allowed: any</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default: variant</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```

# print "\t----\t log Ex 1 \t----"
print qa.log('2')
# {'value': 0.69314718055994529, 'unit': ''}
#
```

```
quanta.log10.html

**quanta.log10 - Function**

1.5.1 logarithm of quantity

**Description**

log10 gives logarithm of dimensionless quantity

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>v</th>
<th>dimensionless quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: variant</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```
###
#
print "\t----\t log10 Ex 1 \t----"
print qa.log10('2')
#{'value': 0.3010299956639812, 'unit': ''}
#
###
```
quanta.exp.html

**quanta.exp - Function**

L.5.1 exponential of quantity

**Description**

exp gives exponential value of dimensionless quantity

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>dimensionless quantity</td>
<td></td>
</tr>
</tbody>
</table>

allowed: any
Default: variant

**Returns**

record

**Example**

```python

"""
#
print "\t----\t exp Ex 1 \t----"
print qa.exp('2')
#{'value': 7.3890560989306504, 'unit': ''}
try:
    print qa.exp('2m')
except Exception, e:
    print "Caught an expected exception", e
#Caught an expected exception Quantum::exp illegal unit type 'm'
#"""
```

1128
**quanta.sqrt.html**

**quanta.sqrt - Function**

1.5.1 square root of quantity

**Description**

sqrt gives square root of quantity with only even powered dimensions

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>dimensionless quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td></td>
</tr>
</tbody>
</table>

allowed: any

Default: variant

**Returns**

record

**Example**

```python
""
#
print "\t----\t sqrt Ex 1 \t----"
print qa.sqrt('2m2')
#{'value': 1.4142135623730951, 'unit': 'm'}
try:
    print qa.sqrt('2s')
except Exception, e:
    print "Caught an expected exception", e
#Caught an expected exception UnitVal::UnitVal Illegal unit dimensions for root
#
""
```

1129
**quanta.compare** - Function

compare dimensionality of units

**Description**

compare compares the dimensionality of units of two quantities

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>value</th>
<th>allowed:</th>
<th>any</th>
<th>Default:</th>
<th>variant</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```markdown
###
# print "\t---\t compare Ex 1 \t---"
print qa.compare('5yd/a', '6m/s')  # equal dimensions
#True
print qa.compare('5yd', '5s')  # unequal dimensions
#False
#
```

1130
quanta.check.html

quanta.check - Function

1.5.1 check for proper unit string

Description

check checks if the argument has a properly defined unit string

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>string</td>
</tr>
<tr>
<td>allowed:</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool

Example

"""
#
print "\t----\t check Ex 1 \t----"
print qa.check('5AE/Jy.pc5/s')
#True
print qa.check('7MYs')
#False
#
"""
quanta.checkfreq.html

**quanta.checkfreq - Function**

1.5.1 check for proper frequency unit

**Description**

checkfreq checks if the argument has a properly defined frequency interpretable unit string

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>cm value</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

````
#!
print "\t----\t checkfreq Ex 1 \t----"
print qa.checkfreq('5GHz')
#True
print qa.checkfreq('5cm')
#True
print qa.checkfreq('5cm/s2')
#False
#````
quanta.pow.html

**quanta.pow - Function**

1.5.1 raise quantity to power

**Description**

pow raises a quantity to an integer power

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
<tr>
<td>a</td>
<td>power</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 1</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
"""
#
print "\t----\t pow Ex 1 \t----"
print qa.pow('7.2km/s', -3)
#{'value': 0.0026791838134430724, 'unit': '(km/s)-3'}
#
"""
```

1133
quanta.constants.html

**quanta.constants - Function**

1.5.1 get a constant

**Description**

constants gets a named constant quantity. Names (no-case, minimax) are:
pi 3.14.. 3.14159 ee 2.71828 c light vel. 2.99792e+08 m/s G grav. const
6.67259e-11 N.m2/kg2 h Planck const 6.62608e-34 J.s HI HI line 1420.41 MHz
R gas const 8.31451 J/K/mol NA Avogadro number 6.02214e+23 mol-1 e
electron charge 1.60218e-19 C mp proton mass 1.67262e-27 kg mp_me mp/me
1836.15 mu0 permeability vac. 1.25664e-06 H/m eps0 permittivity vac.
1.60218e-19 C k Boltzmann const 1.38066e-23 J/K F Faraday const 96485.3
C/mol me electron mass 9.10939e-31 kg re electron radius 2.8179e-15 m a0
Bohr’s radius 5.2918e-11 m R0 solar radius 6.9599e+08 m k2 IAU grav. const²
0.000295912 AU3/d2/S0

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>v</th>
<th>allowed</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td></td>
<td>Default</td>
<td>pi</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
# print "\t----\t constants Ex 1 \t----"
print qa.constants()
#{'unit': ' ', 'value': 3.1415926535897931}
#`
```

1134
quanta.isangle.html

**quanta.isangle - Function**

L.5.1 check if valid angle or time quantity

**Description**

isangle checks if the argument is a valid angle/time quantity.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>angle/time quantity</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
'''
#
print "\t----\t isangle Ex 1 \t----"
print qa.isangle(qa.constants('pi'))
#False
#'''
'''
```
quanta.totime.html

quanta.totime - Function

1.5.1 convert an angle (or a time) to a time

Description

totime converts an angle quantity (or a time) to a time quantity

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>angle/time quantity</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
</tbody>
</table>

Returns

record

Example

"""
#
print "\t----\t totime Ex 1 \t----"
print qa.totime('2d5m')
#{'value': 0.0057870370370376, 'unit': 'd'}
#
"""
quanta.toangle.html

**quanta.toangle - Function**

1.5.1 convert a time (or an angle) to an angle

**Description**

toangle converts a time quantity (or an angle) to an angle quantity

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>angle/time quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```
#
print "\t----\t toangle Ex 1 \t----"
print qa.toangle(’5h30m12.6’)  #
#{’value’: 82.55249999999995, ’unit’: ’deg’}#
```

1138
quanta.splitdate.html

quanta.splitdate - Function

1.5.1 split a date/time into a record

Description

splitdate splits a date/time quantity into a record with constituent fields like year, yearday, month etc. All fields will be integer (to enable use as index and easy personal formatting), with the exception of the $s$ field which is a double float. See the example for the fields returned.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>angle/time quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
</tbody>
</table>

Returns

record

Example

```python
# print "\t----\t splitdate Ex 1 \t----"
print qa.splitdate('today')
print qa.splitdate('183.33333333deg')
#{'mjd': 0.50925925925000004, 'week': 46, 'usec': 999999, 'hour': 12, 'min': 13, 'yearday': 321, 'msec': 999, 'month': 11, 's': 19.999999200003487, 'sec': 19, 'weekday': 3, 'year': 1858,
```
# 'monthday': 17}
#
"""
quanta.tos.html

**quanta.tos - Function**

1.5.1 convert quantity to string

**Description**

tos converts a quantity to a string with the precision defined with the `setformat('prec')` (which defaults to 9). If the optional `prec` argument is set to an integer value greater than 1, that precision is used in the conversion.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>value</td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
<td></td>
</tr>
<tr>
<td>prec</td>
<td>convert precision of value</td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 9</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

string

**Example**

```python
# print "\t----\t tos Ex 1 \t----"
a = qa.quantity('2.56 yd/s')
print a
#{'value': 2.5600000000000001, 'unit': 'yd/s'}
print qa.tos(a)
#2.560000000000000yd/s
a=qa.quantity(1./7, 'km/s')
print qa.tos(a)
```
#0.142857143km/s
print qa.tos(a,2)
#0.14km/s
print qa.tos(a,20)
#0.14285714285714284921km/s
print qa.tos(a)
#0.142857143km/s
#
""
quanta.type.html

**quanta.type - Function**

1.5.1 type of tool

**Description**

Type will return the tool name.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
</table>

**Returns**

string

**Example**

```python
""
#
# print "\t----\t type Ex 1 \t----"
print qa.type()
#quanta
#
""
```
**quanta.done** - Function

Free resources used by tool. Current implementation ignores input parameter, does nothing and returns true

### Description

Currently, this method is an NOP.

### Arguments

#### Inputs

<table>
<thead>
<tr>
<th>kill</th>
<th>force kill of the default tool (ignored)</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>

#### Returns

bool

### Example

```python
# ""
# print "\t----\t done Ex 1 \t----"
print qa.done()
#True
print qa.done()
#True
print qa.done(kill=T)
#True
# ""
```
quanta.unit.html

quanta.unit - Function

1.5.1 quantity from value v and unit string

Description

unit makes a quantity from a string, or from a value and a string. Note that
unit is a synonym for quantity (see example there).

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>unitname</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

Returns

record
**quanta.isquantity** - **Function**

Check if quantity

**Description**

Checks if the operand is a correct quantity

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>v</code></td>
<td>value to be tested</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
</tbody>
</table>

**Returns**

`bool`

**Example**

```python
""
#
print "\t----\t isQuantity Ex 1 \t----"
a = qa.quantity("5Jy")  # make a quantity
print a
#{'value': 5.0, 'unit': 'Jy'}
print qa.isquantity(a)  # is it one?
#True
print qa.isquantity("5Jy")  # and this string?
#True
#
""
```
quanta.setformat - Function

1.5.1  set format for output of numbers. (NOT IMPLEMENTED YET!)

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>type-coded string indicating which format parameter to set</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>v</td>
<td>format parameter value - numeric or coded string, depending on format type</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: F</td>
</tr>
</tbody>
</table>

Returns

bool
quanta.getformat.html

**quanta.getformat - Function**

1.5.1 get current output format (NOT IMPLEMENTED YET!)

**Description**

getformat returns the current format value set for the different format possibilities. See the setformat function for the different format type descriptions. The known types are:
prec, aprec, tprec, long, lat, len, dtime, elev, auto, vel, freq, dop, unit.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>t</th>
<th>type - coded string</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

string

**Example**

```python

"""
#
print "\t-------\t getformat Ex 1 \t-------"
print qa.getformat('prec')
#6
#setformat is NOT IMPLEMENTED YET!
#qa.setformat('prec', 12) # set precision to 12 significant digits
#T
#print qa.getformat('prec')
#12
print qa.getformat('long')
#hms

1148```
quanta.formxxx.html

**quanta.formxxx - Function**

1.5.1 Format a quantity using given format, allowed are hms, dms, deg, rad, +deg.

**Description**

form.xxx (xxx can be lat, long, len, vel, freq, dtime, unit) will format the input into a string using the global format information set by setformat().

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>value to be converted</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
<tr>
<td>format</td>
<td>xxx can be hms, dms, deg, rad or +deg</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: dms</td>
</tr>
<tr>
<td>prec</td>
<td>digits in fractional part of output string for dms,hms</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 2</td>
</tr>
</tbody>
</table>

**Returns**

string

**Example**

```
# form_xxx NOT IMPLEMENTED YET!
```

1150
#5.99584916 cm
print "Last example, exiting! ..."
exits()#
"

1.6 spectralline - Module

Spectral line search from Splatalogue

Description
This module contains functionality to access spectral-line Splatalogue line list.
The available tools in this module are

- **spectralline** - In order for the spectralline tool to be able to search Splatalogue a splatalogue line list, an ascii version of a line list must be exported using the splatalogue web interface and then converted to a CASA table. The method splattotable does this conversion.

Images

---

SynthesisPackage.html

1151
Chapter 2

Package Synthesis

The synthesis package contains modules needed for processing synthesis data. calibrater-Module.html

2.1 calibrater - Module

Module for synthesis calibration
include calibrater.g

**Description**  The calibrater module provides synthesis calibration capabilities within CASA. The primary purpose of this module is to solve for calibration components, and to optionally apply these corrections to the observed data. The calibration module is designed to be used in conjunction with the imager module which provides support for synthesis imaging.

The calibration model adopted by calibrater is that of the Hamaker-Bregman-Sault measurement equation for synthesis radio telescopes (see CASANote [189]). This represents calibration corrections as matrices acting on 4-vectors representing the four possible correlations measured by an interferometer in full polarization. The calibration matrices cover a diversity of instrumental effects, including: parallactic angle and feed configuration (P,C), atmospheric phase (T), electronic gain (G), bandpass (B), instrumental polarization (D), baseline-based (correlator) corrections (M, MF), and baseline-based fringe-fitting (K).

The calibration data are stored in CASA tables and can be directly examined, manipulated or edited in the Glish command line interpreter (CLI) via the table tool. The calibration tables may be interpolated when applied to the observed uv-data.

The solver allows the calibration components to be determined over different time intervals, thus allowing, as an example, the solution for atmospheric phase effects (T) over a much shorter interval than electronic gain terms (G). This also allows polarization self-calibration for time variable instrumental polarization corrections.

1152
The measurement equation is designed to model calibration effects for a
generic radio telescope and the calibration and synthesis modules are, in general,
not instrument specific.

Each calibrater tool created acts on a specified Measurement Set (MS),
containing the observed uv-data. The Measurement Set format is described in
(see CASANote 191). The interaction of the calibrater tool with specific MS
data columns is important. The observed data, as recorded by the instrument,
are stored in the DATA column of the MS, and are referred to as the observed
data. If calibration corrections are applied by calibrater, the resulting cali-
brated data are stored in a separate CORRECTED_DATA column in the MS.
These columns can be selected when imaging the data using the imager tool.
A further MS column is used by calibrater, namely the MODEL_DATA
column. The difference between the model data and corrected data columns is
used to form $\chi^2$, when solving for individual calibration components. It is im-
portant to set the MODEL_DATA column before using calibrater to solve
for calibration. This can be done using the imager functions setjy or ft.

The capabilities of the calibrater module are made available by including
the associated Glish initialization script for the module, as:

```
- include 'calibrater.g'
```

where a hyphen precedes user input. The Glish response is indicated without
the prompt.

A calibrater tool is created and attached to a specified measurement set
as indicated in the following example:

```
- c:=calibrater('3C273XC1.MS');
```

A variety of functions can be invoked for any given calibrater tool. These
functions fall broadly into two categories: i) functions which set parameters to
be used by the calibrater; and ii) the execution of explicit calibration procedures
such as solving for, or applying calibration corrections.

Option (i) may equivalently be viewed as setting the state of the calibrater
tool. These functions are named with the prefix set, such as in setdata and
setapply. When created, the calibrater tool sets default internal information
for each of the calibration components (measurement equation correction mat-
rices). This information is modified using the setapply and setsolve functions
as shown in the following example:

```
# Set the solution interval for the electronic gain matrix (G) to
# 300 seconds, specify input and output calibration table names,
# and enable this component for phase and amplitude solution.
# Use antenna number 3 as the reference for the solutions.
# ```
- c.setapply ("G", 0.0, "gcal_in", ");
- c.setsolve ("G", 300, F, 3, "gcal_out", F);

Once the state of the **calibrater** tool has been set, explicit calibration functions, as outlined in option (ii) above, are executed as follows:

```
# Solve for the selected calibration components
- c.solve()
```

```
# Apply the calibration components to the measurement set data
- c.correct()
```

**Example**  The following example illustrates the quickest way to perform simple self-calibration, starting from an input FITS file in the local area. The **imager** module should be consulted for detailed information on the imaging functions.

```
# Include the synthesis scripts
include 'imager.g';
include 'calibrater.g';
include 'ms.g';

# Construct a measurement set from the input FITS file
m:=fitstoms (msfile='3C273XC1.MS', fitsfile='3C273XC1.FITS');
m.close();
m.done();

# Create an imager tool
sk:=imager('3C273XC1.MS');

# Set image parameters
sk.setimage (nx=256, ny=256, cellx='0.7arcsec', celly='0.7arcsec');

# Make a dirty image and deconvolve using CLEAN
```
# sk.image ('observed', image='3C273XC1.dirty');
sk.clean (niter=1000, threshold='3mJy', model='3C273XC1.clean.model');
#
# Fourier transform the model to the uv-plane
#
sk.ft (model='3C273XC1.clean.model');
#
# Close the imager tool
#
sk.close();
sk.done();
#
# Create a calibrater tool
#
c:=calibrater('3C273XC1.MS');
#
# Select solution for electronic gain (G) and atmospheric phase (T)
#
c.setsolve ("G", 300.0, F, 3, "gcal_out", F);
c.setsolve ("T", 30.0, T, 3, "tcal_out", F);
#
# Solve for the selected G and T components
#
c.solve();
#
# Close the calibrater tool
#
c.close();
2.1.1 calibrater - Tool

Synthesis calibration (self- and cross-)

Requires:

**Synopsis**

**Description**

The calibrater tool (cb) provides for synthesis calibration operations within CASA.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calibrater</td>
<td>Construct a calibrater tool</td>
</tr>
<tr>
<td>open</td>
<td>Attach MeasurementSet to the calibrater tool</td>
</tr>
<tr>
<td>selectvis</td>
<td>Set the data selection for subsequent processing</td>
</tr>
<tr>
<td>setmodel</td>
<td>Set the sky model used to compute the model visibilities</td>
</tr>
<tr>
<td>setptmodel</td>
<td>Set the point source model Stokes parameters to be used to compute the model visibilities</td>
</tr>
<tr>
<td>setapply</td>
<td>Arrange to apply calibration</td>
</tr>
<tr>
<td>setcallib</td>
<td>Arrange to apply calibration via a Cal Library</td>
</tr>
<tr>
<td>validatecallib</td>
<td>Validate a Cal Library record</td>
</tr>
<tr>
<td>setsolve</td>
<td>Arrange to solve for calibration</td>
</tr>
<tr>
<td>setsolvegainspline</td>
<td>Specialization of setsolve for cubic spline G (time-dependent gain) solving</td>
</tr>
<tr>
<td>setsolvebandpoly</td>
<td>Specialization of setsolve for polynomial B (bandpass) solving</td>
</tr>
<tr>
<td>state</td>
<td>Request the apply/solve state of the calibrater tool</td>
</tr>
<tr>
<td>reset</td>
<td>Reset the selected apply and/or solve components</td>
</tr>
<tr>
<td>initcalset</td>
<td>Re-initialize the calibration scratch columns.</td>
</tr>
<tr>
<td>delmod</td>
<td>Delete model data representations in the MS.</td>
</tr>
<tr>
<td>solve</td>
<td>Solve for the selected calibration components</td>
</tr>
<tr>
<td>correct</td>
<td>Apply calibration information</td>
</tr>
<tr>
<td>corrupt</td>
<td>Corrupt model with calibration tables</td>
</tr>
<tr>
<td>initweights</td>
<td>Initialize MS weight info in various ways.</td>
</tr>
<tr>
<td>fluxscale</td>
<td>Bootstrap the flux density scale from standard calibrators</td>
</tr>
<tr>
<td>accumulate</td>
<td>Accumulate incremental calibration solutions into a cumulative calibration table</td>
</tr>
<tr>
<td>activityrec</td>
<td>Returns a record containing properties of recent activity</td>
</tr>
<tr>
<td>specifical</td>
<td>Externally specify calibration of various types</td>
</tr>
<tr>
<td>smooth</td>
<td>Produce a smoothed calibration table</td>
</tr>
<tr>
<td>listcal</td>
<td>List the contents of a calibration table</td>
</tr>
<tr>
<td>posangcal</td>
<td>Apply position angle calibration to an existing cal table</td>
</tr>
</tbody>
</table>
linpolcor  Correct the gain table for linear polarization of the calibrator
plotcal   Plot a calibration table
modelfit  Model fitting
createcaltable Create an empty calibration table
updatecaltable Caltable modernizer.
close     Close the calibrator tool
done      Destroy the calibrator tool
calibrater.calibrater.html

**calibrater.calibrater - Function**

2.1.1 Construct a calibrater tool

**Description**

Create a `calibrater` tool. The casapy environment provides a standard calibrater tool for general use (cb), but additional calibrater tools may be created if needed. Calibrater tools created in this way are independent of the standard calibrater tool.

**Arguments**

**Returns**

`calibrater`

**Example**

```python
cb2=calibrater.create()
```
calibrater.open.html

**calibrater.open - Function**

2.1.1 Attach MeasurementSet to the calibrater tool

**Description**

Attaches a MeasurementSet to the calibrater tool for further processing with other methods.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>MeasurementSet file name. No default allowed: string Default:</td>
</tr>
<tr>
<td>compress</td>
<td>Compress calibration columns? allowed: bool Default: false</td>
</tr>
<tr>
<td>addcorr</td>
<td>Add scratch columns? allowed: bool Default: true</td>
</tr>
<tr>
<td>addmodel</td>
<td>Add MODEL_DATA column along with CORRECTED_DATA? allowed: bool Default: true</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```plaintext
cb.open('ngc5921.ms');
```
calibrater.selectvis.html

**calibrater.selectvis - Function**

2.1.1 Set the data selection for subsequent processing

**Description**

This function provides for selection of the visibility data from the MS which will be treated by subsequent execution of the `solve` and `correct` functions. Note that data selection is not cumulative, i.e., any selection made in a previous call to `selectvis` will be overridden by the current call. Most of the `selectvis` parameters use the standardized MS Selection syntax. The parameters are described below. The selected data will satisfy the logical AND of all non-trivially specified parameters. Note that the old-fashioned strided channel selection parameters are deprecated (and will soon be removed); use `spw` instead. Running `selectvis` with no specified parameters restores selection of the entire MS.

- **time** is used to specify time ranges in a standard format
- **spw** is used to specify spectral window and channel selection. Currently, only a single channel range can be specified per `spw`.
- **scan** is used to specify scan numbers and ranges
- **observation** is used to specify observation ID(s).
- **field** is used to specify field names or indices
- **baseline** is used to specify antenna and baseline combinations
- **uvrange** is used to specify baseline length ranges
- **chanmode** is deprecated (use `spw`)
- **nchan** is deprecated (use `spw`)
- **start** is deprecated (use `spw`)
- **step** is deprecated (use `spw`)
- **mstart** is deprecated (use `spw`)
- **mstep** is deprecated (use `spw`
**msselect** is used to specify a subselection of data according to Measurement Set columns in conditional combinations not possible with the standard parameters above. This parameter should be specified as a valid [TaQL] expression. If both `msselect` and the standard selection parameter are used together, they are combined with a logical AND, i.e., the data must jointly satisfy all `selectvis` parameters.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Select on</th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>on time</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>spw</td>
<td>on spectral window</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>scan</td>
<td>on scan</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>field</td>
<td>on field</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>intent</td>
<td>on intent or state</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>observation</td>
<td>by observation ID(s)</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>baseline</td>
<td>on antennas/baselines</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>uvrange</td>
<td>by uvrange</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>chanmode</td>
<td>Type of data selection: channel or velocity</td>
<td>string</td>
<td>channel</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>velocity</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>none</td>
</tr>
<tr>
<td>nchan</td>
<td>Number of channels to select (mode='channel')</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>start</td>
<td>Start channel (0-relative) (mode='channel')</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>step</td>
<td>Step in channel number (mode='channel')</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>mstart</td>
<td>km/s Start velocity (e.g. '20Km/s')</td>
<td>doublekm/s</td>
<td>0.0</td>
</tr>
<tr>
<td>mstep</td>
<td>km/s Step in velocity (e.g. '100m/s')</td>
<td>doublekm/s</td>
<td>0.0</td>
</tr>
<tr>
<td>msselect</td>
<td>TAQL selection string. Default (empty) is no specific selection.</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>
Returns
bool

Example

Open and select a field:

cb.open('ngc5921.ms');
cb.selectvis(field='N5921_2');  # by complete name
cb.selectvis(field='N5921*');    # with wildcard
cb.selectvis(field='2');        # by index

Select a field and a channel range:

cb.selectvis(spw='0:10~40', field='N5921*');

Select using all MS Selection parameters (these parameters are over-specified somewhat, i.e., scan 6 contains only field N5921_2, etc.):

cb.selectvis(time='>1995/04/13/10:40:00', # times greater than this
             spw='0:20~40',    # channels 20-40 in spw 0
             scan='6',        # scan 6 only
             field='N59*',    # fields matching N59*
             baseline='1 \& *',  # baselines to antenna 1
             uvrange='>0.0klambda')  # baselines greater than zero length

Reset selection to the entire dataset

cb.selectvis()
calibrater.setmodel.html

**calibrater.setmodel - Function**

2.1.1 Set the sky model used to compute the model visibilities

**Description**

Name of the model image to be used as a sky model for model visibility computations. For now, this is used only by EP-Jones solver.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelimage</td>
<td>Name of the model image.</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```plaintext
cb.setmodel("mymodel");
```
calibrater.setptmodel.html

**calibrater.setptmodel - Function**

Set the point source model Stokes parameters to be used to compute the model visibilities

**Description**

Set a global point source model Stokes parameters to use in solving operations.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stokes</td>
<td>Vector of Stokes parameters.</td>
</tr>
<tr>
<td>allowed:</td>
<td>doubleArray</td>
</tr>
<tr>
<td>Default:</td>
<td>0.0 0.0 0.0 0.0</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```c
cb.setmodel([1,1,0,0]);
```
**calibrater.setapply** - Function

2.1.1 Arrange to apply calibration

**Description**

This function is used to specify the calibration components which should be applied during subsequent execution of the `solve` and `correct` functions. This function should be executed as many times as necessary to specify all desired calibration components.

Each calibration component represents a separate calibration matrix correction included in the measurement equation. The different types correspond to different instrumental and atmospheric effects. Calibration components are available as calibration tables generated by previous `solve` executions (types ‘B’, ‘BPOLY’, ‘G’, ‘GSPLINE’, ‘D’, ‘DF’, ‘T’, ‘M’, ‘MF’, ‘X’), or are calculated analytically on the fly (types ‘P’, ‘TOPAC’, ‘GAINCURVE’). Upon execution of `solve` or `correct`, the group of specified calibration components will be applied in the order prescribed by the Measurement Equation formalism.

The parameters are as follows:

- **type** The calibration type being specified. This is only required for analytic types ('P', 'TOPAC', 'GAINCURVE'). When specifying an existing pre-solved calibration table, it is not necessary to explicitly specify the type; this will be discerned from the table. (Specifying the type as well as the table will force a check that the table contains solutions of the specified type.

For **type='GAINCURVE'**, an elevation-dependent correction will be applied using parameters read from the data repository. Currently, this is only supported for the VLA.

- **t** This parameter will be used in a future release to control the range of applicability of the specified calibration. Currently, it is ignored.

- **table** For pre-solved calibration, the file name of the table to apply.

- **field** The fields to select from the specified table, using MS Selection syntax (as in selectvis).

- **interp** The desired type of time-dependent interpolation. Use `interp='nearest'` to calibrate each datum with the calibration value nearest in time. Use `interp='linear'` to calibrate each datum with calibration phases and amplitudes linearly interpolated from neighboring
(in time) values. In the case of phase, this mode will assume that phase jumps greater than 180 degrees between neighboring points indicate a cycle slip, and the interpolated value will follow this change in cycle accordingly (i.e., the implied rate will always be less than 180 degrees per sample). Use `interp='aipslin'` to emulate the basic interpolation mode used in classic AIPS, i.e., linearly interpolated amplitudes, with phases derived from linear interpolation of the complex calibration values. While this method avoids having to track cycle slips (which is unstable for solutions with very low SNR), it will yield a phase interpolation which becomes increasingly non-linear as the spanned phase difference increases. The non-linearity mimics the behavior of `interp='nearest'` as the spanned phase difference approaches 180 degrees (the phase of the interpolated complex calibration value initially changes very slowly, then rapidly jumps to the second value at the midpoint of the interval). If the uncalibrated phase is changing this rapidly, a 'nearest' interpolation is not desirable. Usually, `interp='linear'` is the best choice. The `interp` parameter is applicable to any calibration type, as long as there are sufficient solutions available to perform the interpolation. Note that calibration solutions which have been determined for only one timestamp will default to 'nearest'. More interpolation options (e.g., 'cubic') will be added in the future.

**select** Used to specify general selection of a subset of calibration measurements from the table to be applied to the visibility data. Arbitrary cross-calibration is possible by combining this function with the `setdata` function. The string specified must be a valid TaQL expression.

**spwmap** This parameter is used to indicate how solutions derived from different spectral windows should be applied to other spectral windows. Nominally, data in each spectral window will be corrected by solutions derived from the same spectral window. This is the default behavior of `spwmap`, i.e., if `spwmap` is not specified, calibrator will insist that data be corrected by solutions from the same spw. Otherwise, `spwmap` takes a vector of integers indicating which spectral window solutions to apply to which spectral window data, such that `spwmap[j]=i` causes solutions derived from the i-th spectral window to be used to correct the j-th spectral window. For example, if (say) bandpass solutions are available for spws 0 & 2, and it is desired that these be applied to spws 1 & 3 (as well as 0 & 2), respectively, use `spwmap=[0,0,2,2]`. Even if some spws do not require an explicit `spwmap` setting, yet one or more does, it is safest to specify it explicitly for all, e.g., `spwmap=[0,1,3,3]` indicates that spw 2 will be corrected with solutions from spw 3, and the others will behave nominally. Note that if no solutions exist for any of the spws specified in `spwmap`, an error message will result.

**calwt** If set True, the data weights will be calibrated along with the data.
This is usually desirable.

**opacity**  For **type=’TOPAC’**, an elevation-dependent opacity correction will be applied according to the zenith opacity value supplied in the **opacity** parameter. Currently, only one zenith opacity value can be supplied, and it is used for all antennas.

Use the **state** function to review the list of calibration components that have been set for application.

Pending improvements:

- Enable variety of interpolation modes and timescales
- Allow for antenna- and time-dependent opacities

**Arguments**
Inputs

<table>
<thead>
<tr>
<th>type</th>
<th>Component type</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>BPOLY</td>
</tr>
<tr>
<td></td>
<td>G</td>
</tr>
<tr>
<td></td>
<td>GSPLINE</td>
</tr>
<tr>
<td></td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>P</td>
</tr>
<tr>
<td></td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>TOPAC</td>
</tr>
<tr>
<td></td>
<td>GAINCURVE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>t</th>
<th>Interpolation interval (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed</td>
<td>double</td>
</tr>
<tr>
<td>Default</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>table</th>
<th>Calibration table name</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>field</th>
<th>Select on field</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed</td>
<td>any</td>
</tr>
<tr>
<td>Default</td>
<td>variant</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>interp</th>
<th>Interpolation type (in time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>aipslin</td>
</tr>
<tr>
<td></td>
<td>nearest</td>
</tr>
<tr>
<td></td>
<td>linear</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>select</th>
<th>TAQL selection string. Default is no selection.</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>calwt</th>
<th>Calibrate weights?</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>spwmap</th>
<th>Spectral windows to apply</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed</td>
<td>intArray</td>
</tr>
<tr>
<td>Default</td>
<td>-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>opacity</th>
<th>Array-wide zenith opacity per antenna (for type='TOPAC')</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed</td>
<td>doubleArray</td>
</tr>
<tr>
<td>Default</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Returns

| bool | |
|------| |

1170
Example

```python
cb.open('ngc5921.ms')
cb.selectvis(field='N5921*')
cb.setapply (type='G', table='gcal', field='1445*')
cb.setapply (type='P')
cb.correct();
cb.close();
```

In this example, we apply parallactic angle corrections and a gain calibration derived from a field whose name matches '1445*' in a caltable called 'gcal' to data for a field matching 'N5921*'.
calibrater.setcallib.html

**calibrater.setcallib - Function**

2.1.1 Arrange to apply calibration via a Cal Library

**Description**

TBD

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>callib</td>
</tr>
<tr>
<td>allowed:</td>
</tr>
<tr>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

TBD
calibrater.validatecallib.html

**calibrater.validatecallib - Function**

2.1.1 Validate a Cal Library record

**Description**

TBD

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>callib</td>
<td>A calibration library record</td>
</tr>
</tbody>
</table>

allowed: record

Default: 

**Returns**

bool

**Example**

TBD

1173
calibrater.setsolve.html

**calibrater.setsolve** - Function

2.1.1 Arrange to solve for calibration

**Description**

This function specifies the calibration component that will be solved for by the `solve` function. Currently, only one type can be solved for at one time. Each calibration component represents a separate calibration matrix correction included in the measurement equation. The different types correspond to different instrumental and atmospheric effects. Currently, the solvable calibration components are types 'G', 'T', 'B', 'D' and 'DF', which are antenna-based, and, 'M' and 'MF', which are baseline-based. Arrange to pre-apply any existing calibration components (of types other than the solved-for one) using the `setapply` function.

The parameters are:

**type** Specify the calibration type you want to solve for, from 'G', 'T', 'B', 'D', 'DF', 'M', 'MF'.

**t** Specify the solution interval. This can be specified as an integer (units of seconds assumed) or as a string containing a value and units (e.g., '30s', '45min', '2h') or 'inf' (infinite) or 'int' (per data integration). A solution interval of 0 (with or without units) is the same as 'int' (per integration), and negative solution intervals are treated as 'inf' (infinite).

**table** Specify the output calibration table name in which to store the calibration solve result. Existing tables will be deleted and replaced.

**append** Append the solutions to an existing table.

**preavg** Specify the amount of pre-average (in time) within the solution interval. By default, data are averaged up to the solution interval (or up to 5 minutes for 'D' solving).

**phaseonly** This parameter is deprecated, use `apmode`.

**apmode** Control generation of amplitude-only ('a'), phase-only ('p'), or amplitude-and-phase ('ap', the default) solutions.

**refant** Specify an antenna (using data selection syntax) for referencing the solutions.
solnorm Normalize the solutions by their mean post-solve. For 'B', and 'MF', this is a complex normalization per solution spectrum. For other types, this is a global (per-spw) normalization of the amplitudes only.

minsnr Specify the SNR below which solution are rejected.

combine Specify which data axes (spw, field, scan, or some combination) on which the data should be combined to generate a single solution. E.g., combine='spw' will force combination of many spws to form a single solution (per solution interval). Similarly, combine='scan' with a long solution interval will force the combination of scans to yield individual solutions (per field and spw). Ordinarily, solutions are always broken at scans boundaries. Separate multiple combine options with commas.

fillgaps For 'B' solutions, specify the largest solution channel gap (which arise due to flagged data) that will be filled post-solve via interpolation. Such solution gaps remain flagged by default.

Pending improvements:
- Change t to solint?
- Permit flexible specification of preavg (as for t)

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Component type</th>
<th>allowed:</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td></td>
<td>Default:</td>
<td>G</td>
</tr>
<tr>
<td>solution interval (units optional)</td>
<td></td>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>table</td>
<td>Output calibration table name</td>
<td>Default:</td>
<td>variant</td>
</tr>
<tr>
<td>append</td>
<td>Append to existing table?</td>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>preavg</td>
<td>Pre-averaging interval (in sec)</td>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>phaseonly</td>
<td>Solve only for phase?</td>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>apmode</td>
<td>Solve for 'AP', 'A' (amp-only) or 'P' (phase-only)</td>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>refant</td>
<td>Reference antenna. Default is none.</td>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>minblperant</td>
<td>Minimum number of baselines per ant for solving</td>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>solnorm</td>
<td>Normalize solution after solve</td>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>minsnr</td>
<td>SNR threshold for accepting solutions</td>
<td>allowed:</td>
<td>float</td>
</tr>
<tr>
<td>combine</td>
<td>Data axes on which to combine solving (scan, spw, and/or field)</td>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>fillgaps</td>
<td></td>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>cfcache</td>
<td>Name of the directory to be used for convolution function disk cache. This is used when type=EP.</td>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>painc</td>
<td>Parallactic Angle increment used to trigger computation of a new convolution function. This is used when type=EP. Default value implies that only one convolution function will be computed for the entire range of observation.</td>
<td>allowed:</td>
<td>int</td>
</tr>
</tbody>
</table>
Returns
bool

Example

cb.open('ngc5921.ms');
cb.setapply (type='P');
cb.setsolve (type='G', t='300s', refant=3, table='gcal');
cb.solve();
cb.close();

In this example, analytic (non-solvable) parallactic angle corrections are pre-applied before G solutions are obtained on a timescale of 300 seconds. The resulting solutions are phase-referenced to antenna 3, and stored in a calibration table called 'gcal'.

cb.reset();
cb.setapply (type='P', t=5.0);
cb.setapply (type='G', table='gcal');
cb.setsolve (type='D', t=86400.0, preavg=60.0, refant=3, table='dcal');
cb.solve();
cb.close();

In this example, the solve/apply state of the calibrator tool is reset and then the P and G corrections (from above) are applied before solving for D solutions on a diurnal timescale. Note that the data will be averaged only to 60 seconds before the solution. The resulting D solutions are stored in a table called 'dcal'.

1177
calibrater.setsolvegainspline.html

**calibrater.setsolvegainspline - Function**

**2.1.1 Specialization of setsolve for cubic spline G (time-dependent gain) solving**

**Description**

This function is a specialization of the `setsolve` method which should be used when cubic spline G solutions are desired, e.g., when SNR on calibrators is very low. Currently, this solving mode treats dual polarization data on a per-polarization basis. The option to obtain a joint solution (à la 'T') will be provided in the future.

The visibility data are averaged in frequency (for multi-channel data) prior to the solution.

This method uses many of the basic parameters as the generic `setsolve`. Parameters unique to the spline solver are:

- **mode** For phase solutions only, use `mode='PHAS'`. For amplitude solutions only, use `mode='AMP'`. If both are desired, use `mode='PHASAMP'`, and both will be solved for using the same spline timescale (this mode also assumes that all calibrators have the correct relative flux densities). If solving for phase and amplitude separately (usually in this order), it is usually desirable to apply the first one when solving for the second one. Spline solution so obtained will be stored in separate calibration tables. In the near future, the `mode` parameter will be consolidated with the generic `apmode` parameter.

- **splinetime** The spline timescale (time between knots) is specified here. The default is 10800 seconds (3 hours). In future this parameter will be consolidated with the generic `t` parameter. The `preavg` parameter should be set to a value at least 4X shorter than the spline time (an error will occur if there is insufficient sampling within the `splinetime` timescale), and consistent with the expected coherence. Consistent with these constraints, use the largest possible value for `preavg` to optimize the SNR of the pre-solve phase-tracking algorithm.

- **npointaver and phasewrap** These parameters tune the phase-unwrapping algorithm when `mode = 'PHAS'`. Cycle slips are detected (and removed before the spline solve) when the median phase a sequence of length `npointaver` (in integrations) differs by more than `phasewrap` degrees from the previous sequence.

Pending improvements:
• Consolidate more parameters with the generic `setsolve`
• Introduce the generic combine options
• Improve phase-tracking algorithm

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output calibration table name</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>table</code></td>
<td>allowed: string</td>
</tr>
<tr>
<td><code>append</code></td>
<td>Append to existing table?</td>
</tr>
<tr>
<td><code>allowed</code></td>
<td>allowed: bool</td>
</tr>
<tr>
<td><code>mode</code></td>
<td>Phase or Amplitude mode?</td>
</tr>
<tr>
<td><code>allowed</code></td>
<td>allowed: string</td>
</tr>
<tr>
<td><code>splinetime</code></td>
<td>Spline timescale (sec)</td>
</tr>
<tr>
<td><code>allowed</code></td>
<td>allowed: double</td>
</tr>
<tr>
<td><code>preavg</code></td>
<td>Pre-averaging interval (in sec)</td>
</tr>
<tr>
<td><code>allowed</code></td>
<td>allowed: double</td>
</tr>
<tr>
<td><code>npointaver</code></td>
<td>allowed: int</td>
</tr>
<tr>
<td><code>phaseshift</code></td>
<td>allowed: double</td>
</tr>
<tr>
<td><code>refant</code></td>
<td>Reference antenna. Default is none.</td>
</tr>
<tr>
<td><code>allowed</code></td>
<td>allowed: any</td>
</tr>
</tbody>
</table>

**Returns**

`bool`

**Example**

```
1179
```
cb.open('ngc5921.ms')
cb.selectvis(field='1445*')
cb.setsolvegainspline (table='gcalph', mode='PHAS', splinetime=3600.0, preavg=60.0) cb.solve()

cb.setsolvegainspline (table='gcalamp', mode='AMP', splinetime=10800.0);
cb.solve();
cb.close();

In this example, a spline solution is first found for phase on a hourly timescale, then for
calibrater.setsolvebandpoly.html

**calibrater.setsolvebandpoly - Function**

2.1.1 Specialization of setsolve for polynomial B (bandpass) solving

**Description**

This function is a specialization of the `setsolve` method which should be used to arrange for bandpass solving when polynomial solutions for B are desired, e.g., when per-channel SNR on calibrators is too low to obtain a useful sampled bandpass.

Prior to the solution, the visibility data are averaged in time, and the solution is performed for both phase and amplitude.

This method uses most of the same parameters as the generic `setsolve`, with a few unique additions:

**degamp and degphase** The parameters permit specification of the polynomial order to use in amp and phase. Specifying 0 (zero) yields constant solutions.

**visnorm** This parameter is used to normalize the assembled spectral data, in a per baseline manner. If set True, this will have the effect of removing any non-frequency-dependent closure errors (e.g., as caused by source structure, or introduced by the instrument) from the data, and should be used with caution. The resulting solutions will be effectively normalized as well. When `visnorm=False` is used, closure errors in the data (as supplied to the solver) may be visible in the form of offsets between the data and solutions. For bandpass calibration, this is usually ok, as the shape of the bandpass is the most important aspect of the solution. In future this parameter will be generalized and made available for other solve types. 

(NB: Use of `solnorm=True` still provides for post-solve normalization of the solutions.)

**maskcenter and maskedge** These parameters control how many channels are ignored on-the-fly, at the center and edges of each input spectral window, respectively. To avoid edge channels, it is almost always better to flag these channels directly, or select against them in `setdata`. Aggressive use of maskedge (large values), will yield polynomial solutions which will tend to diverge at the edges (especially when the polynomial degree is also high), because maskedge does not change the frequency domain of the solutions. Such solutions should be used with caution in subsequent operations. (It is best to avoid use of maskedge.)
The BPOLY solution is performed for both phase and amplitude, and the result will be stored in the same table. The frequency domain of the solutions is limited to only the range of frequencies selected in `selectvis`. When correcting data with these solutions (for other solves or with `correct`), only data within this domain will be corrected. Data outside (e.g., edge channels avoided in `setdata` for the solve), will not be corrected. Therefore, the same (or narrower) channel selection is recommended for all operations using solutions produced by this function and `solve()`.

Note that the `combine` parameter can be used meaningfully with the BPOLY solver. When `combine='spw'`, the data from multiple spws will be combined on a common frequency axis, and a single polynomial will be determined spanning them all. This is different than for ordinary sampled 'B' solutions, for which `combine='spw'` causes the bandpass to be combined on a common channel axis, effectively yielding a mean bandpass for the set of spws.

**Arguments**
Returns
bool

Example

1183
cb.open('ngc5921.ms');
cb.selectvis(field='1331*')
cb.setsolvebandpoly(table='bpoly',degamp=5,degphase=7);
cb.solve();
cb.close();

In this example, amplitude (degree 5) and phase (degree 7) Chebychev polynomial bandpasses are determined using the default parameters.
calibrater.state.html

**calibrater.state - Function**

2.1.1 Request the apply/solve state of the calibrater tool

### Description

Request the apply/solve state of the calibrater tool. A listing of all calibration components that have been set for application or solving is written to the logger.

### Arguments

### Returns

bool

### Example

```csharp
cb.open('ngc5921.ms');
cb.setapply ('P', 5.0);
cb.setsolve ('G', 300.0, F, 3, 'gcal_1', T);
cb.state();
```
calibrater.reset.html

**calibrater.reset - Function**

**2.1.1** Reset the selected apply and/or solve components

**Description**

Resets the apply and/or solve components previously set by setapply and setsolve.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply</td>
<td>If true, unset all apply settings</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>solve</td>
<td>If true, unset all solve settings</td>
<td>bool</td>
<td>true</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
cb.open('ngc5921.ms')
cb.setapply ('P', 5.0)
cb.setsolve ('G', 300.0, F, 3, 'gcal_1', T)
cb.state()
cb.reset(apply=T,solve=F);
cb.state()
cb.reset()
```
**calibrater.initcalset - Function**

2.1.1 Re-initialize the calibration scratch columns.

**Description**

This function re-initializes the calibration scratch columns: MODEL\_DATA to unity (in total intensity, and unpolarized), and CORRECTED\_DATA to (observed) DATA. Optionally if calset is set to 1 any model saved in the MS header to for calibration purposes is deleted.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calset</td>
<td>if it set to 1 the model saved in the header is removed</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```
cb.open('ngc5921.ms');
cb.initcalset();
cb.solve();
```
calibrater.delmod.html

calibrater.delmod - Function

2.1.1 Delete model data representations in the MS.

Description

This method can be used to delete the model visibility data representations in the MS. The 'otf' representation is the new (as of v3.4) 'scratch-less' model data, stored as keywords in the MS header containing model data formation instructions. It is generated by the im tool (setjy, ft, and clean methods; usescratch=F in im.open), and if present, overrides the old-fashioned MODEL_DATA column (if present). If a user wishes to use the MODEL_DATA column after having operated with the 'otf' representation, this method can be used to delete the 'otf' representation to make the MODEL_DATA column visible. (Create the MODEL_DATA column by using usescratch=T in the im tool, or by running the cb.open with addmodel=T.) If otf=T, the user may selectively remove only a selection of fields model from the MS by specifying the field parameter. Similarly if the field parameter is specified, selected spws model for those fields may be deleted by specifying the spw.

For convenience, this method also provides a means for deleting the MODEL_DATA column by setting scr=T.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>otf</td>
<td>If T, delete the otf model data keywords</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: false</td>
</tr>
<tr>
<td>field</td>
<td>Select on field</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
<tr>
<td>spw</td>
<td>Select spw only if field is defined</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
<tr>
<td>scr</td>
<td>If T, delete the MODEL_DATA column</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: false</td>
</tr>
</tbody>
</table>
Returns
_bool_

Example

```python
cb.open('ngc5921.ms');
cb.delmod(otf=T,scr=F);  # delete only the otf model for all fields
cb.solve();

cb.open('n4826.ms')
cb.delmod(otf=T, field='1')
# delete otf model of field 1 only, all other fields model are untouched
#if present
   cb.open('n4826.ms')
cb.delmod(otf=T, field='1', spw='2')
# delete otf model of field 1 and spectralwindow 2 only.

### NOTE doing:
   cb.delmod(otf=T, field='', spw='2')

# will delete all otf models and spw will be ignored
```
calibrater.solve.html

calibrater.solve - Function

2.1.1 Solve for the selected calibration components

Description

Execution of this function initiates a solve for the calibration component specified in a previous setsolve execution. Existing calibration components (as specified in one or more setapply executions) will be appropriately applied to the observed and model data according to their position in the Measurement Equation, and their commutation properties.

Arguments

Returns

bool

Example

cb.open(‘ngc5921.ms’);
cb.setapply (‘P’, t=10)
cb.setsolve (‘G’, 300.0, F, 3, ’gcal_1’, T);
cb.solve();
cb.close();
calibrater.correct.html

**calibrater.correct** - Function

2.1.1 Apply calibration information

**Description**

This function applies the calibration components specified via one or more invocations of the `setapply` function to the observed visibility data and writes the result to the CORRECTED_DATA column of the Measurement Set.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>applymode</code></td>
<td>Correction cal/flag mode: <code>&quot;=calflag&quot;,&quot;cal&quot;,&quot;flag&quot;,&quot;trial&quot;</code> allowed: string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```
cb.open('ngc5921.ms');
cb.selectvis(field='1445*')
cb.setapply ('G', 10.0, 'gcal_1')
cb.correct();
cb.close();
```
calibrater.corrupt - Function

Description

This function applies the calibration components specified via one or more invocations of the setapply function to the model visibility data and (over-)writes the result to the MODEL_DATA column of the Measurement Set.

Arguments

Returns

bool

Example

```python
cb.open('ngc5921.ms')
cb.selectvis(field='1445*')
cb.setapply ('G', 10.0, 'gcal_1')
cb.corrupt()
cb.close()
```
calibrater.initweights.html

**calibrater.initweights - Function**

2.1.1 Initialize MS weight info in various ways.

**Description**

This function initializes the MS weight info in various ways.  
If wtmode='ones', SIGMA and WEIGHT will be initialized with 1.0, globally.  
If wtmode='nyq' (the default), SIGMA and WEIGHT will be initialized  
according to bandwidth and integration time. This is the theoretically correct  
mode for raw normalized visibilities.  
If wtmode='sigma', WEIGHT will be initialized according to the existing  
SIGMA column.  
If mode='weight', WEIGHT_SPECTRUM will be initialized according to the  
existing WEIGHT column; dowtspec=T must be specified in this case.  
For the above wtmodes, if dowtspec=T (or if the WEIGHT_SPECTRUM  
column already exists), the WEIGHT_SPECTRUM column will be initialized  
(uniformly in channel), in a manner consistent with the WEIGHT column. If  
the WEIGHT_SPECTRUM column does not exist, dowtsp=T will force its  
creation.  
The follow modes should be used with extreme care: If wtmode='delwtsp', the  
WEIGHT_SPECTRUM column will be deleted (if it exists). If  
wtmode='delsigsp', the SIGMA_SPECTRUM column will be deleted (if it  
exists). Note that creation of SIGMA_SPECTRUM is not supported via this  
method.  
Note that this method does not support any prior selection. Initialization  
of the weight information must currently be done globally or not at all. This is to  
maintain consistency.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>wtmode</td>
<td>Initialization mode</td>
<td>string</td>
<td>nyq</td>
</tr>
<tr>
<td>dowtsp</td>
<td>Initialize WEIGHT,SPECTRUM column</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>tsystable</td>
<td>Tsys calibration table to apply on the fly</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>gainfield</td>
<td>Select a subset of calibrators from Tsys caltable</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>interp</td>
<td>Interp type in time[,freq]. default=linear,linear</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>spwmap</td>
<td>Spectral windows combinations to form for gaintables(s)</td>
<td>intArray</td>
<td></td>
</tr>
</tbody>
</table>

| Returns                     | bool                                                                         |

| Example                     |                                                                              |

cb.open('ngc5921.ms')
cb.initweights()
cb.close()
calibrater.fluxscale - Function

2.1.1 Bootstrap the flux density scale from standard calibrators

Description

This function is used to bootstrap the amplitude scale the calibration solutions according to specified reference calibrator(s) of known flux density. This is necessary when the flux densities of some of your calibrators were unknown (and thus were assumed to be 1 Jy) during G solving. The bootstrapping is achieved by comparing the median gain norm of the calibration solutions derived for the calibrators specified in reference (one or more sources with known flux densities at the time of G solving) with that of the calibrators specified in transfer, and enforcing the assumption that the antenna gains are constant, on average. The gain solutions for the transfer sources are then re-scaled accordingly. The reference and transfer parameters may be specified using the general field selection syntax (as in field in selectvis).

If no transfer fields are specified, then the solutions for all non-reference fields in tablein will be re-scaled. If no tableout is specified the input table will be overwritten with the scaled solutions. Note that the resulting table will only contain solutions for those fields implicit in the reference and transfer specifications. Use append=T to append the scaled solutions to an existing table.

Use the refspwmap parameter to indicate how data for different spectral windows should be matched in calculating the flux density scale factor for transfer fields. The default behavior for refspwmap is to insist on precisely matching spectral windows for reference and transfer fields. When specified, the refspwmap parameter takes a vector of integers indicating which spectral window solutions to use as the reference for others, such that refspwmap[j]=i causes solutions (from reference fields) observed in the i-th spectral window to be used to reference solutions (from transfer fields) observed in the j-th spectral window. For example, for the case of a total of 4 spectral windows: if the reference fields were observed only in spw=2 & 4, and the transfer fields were observed variously in all 4 spws, specify refspwmap=[2,2,4,4]. This will ensure that transfer fields observed in spws 1,2,3,4 will be referenced to reference field data from spws 2,2,4,4, respectively. Note that if the transfer fields were observed only in spws 1 & 3, the same specification would work, but refspwmap=[2,2,4] would suffice. In this case, nothing need be specified for the 4th spw (there are no transfer fields
there), and specifying 2 for the 2nd spw is actually inconsequential (though required so that the specification of 4 for spw 3 is properly interpreted).
The gain values used in the flux scaling determination skewed by outliers. The parameters, gainthreshold and antenna can be used to limit the input gain solutions to be included in the flux scale determination. Use the gainthreshold is a threshold in % from the median values of the gain solutions to be used. Use the antenna to select or de-select (using the MSSelection syntax) antenna(s). Further refinements on the selection based on timerange and scan are possible.
The derived flux densities for the transfer fields will be reported in the logger, and returned to the Python dictionary specified in fluxd. This will be an 2D array of shape [number-of-spectral-windows X number-of-fields]. When multiple spectral windows are involved the spectral index will also be reported by fitting the determined flux densities across the frequencies. The order of a polynomial to be fitted can be specified with fitorder.
Note that elevation-dependent gain effects may render the basic assumption used here invalid, and so should be corrected for prior to solving for G, using types 'TOPAC' or 'GAINCURVE' in setapply.
Note that the visibility data itself is not used directly by this function.
Pending improvements:

- Allow antenna and uv-distance selection to improve results for resolved calibrators
- Set the visibility model according to the flux density results
- An option to use the data to derive the relative flux densities

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>tablein</strong></td>
<td>Input calibration table name</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><strong>reference</strong></td>
<td>Reference calibrator field names (comma-separated)</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
<tr>
<td><strong>tableout</strong></td>
<td>Output calibration table name. Default is input calibration table name.</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><strong>transfer</strong></td>
<td>Transfer source field names (comma-separated). Default is all other fields.</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
<tr>
<td><strong>listfile</strong></td>
<td>Name of listfile that contains the fit information. Default is &quot;&quot; (no file).</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><strong>append</strong></td>
<td>Append to existing table?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td><strong>refspwmap</strong></td>
<td>List of alternate spw for referencing</td>
</tr>
<tr>
<td>allowed:</td>
<td>intArray</td>
</tr>
<tr>
<td>Default:</td>
<td>-1</td>
</tr>
<tr>
<td><strong>gainthreshold</strong></td>
<td>Threshold of gain amplitudes with respect to the median value to be used in flux scale calculation. Default: -1.0 (no threshold)</td>
</tr>
<tr>
<td>allowed:</td>
<td>float</td>
</tr>
<tr>
<td>Default:</td>
<td>-1.0</td>
</tr>
<tr>
<td><strong>antenna</strong></td>
<td>antenna selection/de-selection in flux scale calculation. Default: &quot;&quot; (include all antennas)</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><strong>timerange</strong></td>
<td>timerange sub-selection with antenna selection in flux scale calculation. Default: &quot;&quot; (include all)</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><strong>scan</strong></td>
<td>scan sub-selection with antenna selection in flux scale calculation. Default: &quot;&quot; (include all)</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><strong>incremental</strong></td>
<td>create a incremental caltable</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td><strong>fitorder</strong></td>
<td>order for spectral fitting for multiple spws</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>1</td>
</tr>
<tr>
<td><strong>display</strong></td>
<td>display statistics of the flux ratios</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>
Returns
record

Example

cb.open('ngc5921.ms')
cb.selectvis(field='1331*,1445*')
cb.setsolve(type='G',table='gcal',t='inf')
cb.solve()
cb.fluxscale (tablein='gcal', tableout='flxcal',
        reference='1331*', transfer='1445*');
cb.close();

This example generates a calibration table containing {\tt G} solutions ('gcal') and then writes a re-scaled version, using 1335+305 as the reference calibrator, to derive properly scaled amplitude calibration for the transfer source, 1445+099. We have assumed that 1331+305 has already had its MODEL\_DATA set to the correct flux density.
**calibrater.accumulate - Function**

2.1.1 Accumulate incremental calibration solutions into a cumulative calibration table

**Description**

This function enables cumulative calibration using `calibrater`. It is the analog of the task “CLCAL” in classic AIPS. The `accumulate` function is useful when:

- a calibration solution of a particular type already exists,
- an incremental calibration solution of the same type is desired (an incremental solution in this context means derived independently from, or determined with respect to, the first)
- the first calibration cannot be implicitly recovered in the course of obtaining the incremental solution

For example, a phase-only “G” self-calibration on a target source may be desired to tweak the full amplitude and phase “G” calibration already obtained from a calibrator. The initial calibration (from the calibrator) contains amplitude information, and so must be carried forward, yet the phase-only solution itself cannot (by definition) recover this information, as a full amplitude and phase self-calibration would. In this case, the initial solution must be applied while solving for the phase-only solution, then the two solutions combined to form a cumulative calibration embodying the net effect of both. In terms of the Measurement Equation, the net calibration is the product of the initial and incremental solutions.

The analog of `accumulate` in classic AIPS is the use of CLCAL to combine a series of (incremental) SN calibration tables to form successive (cumulative) CL calibration tables.

Cumulative calibration tables also provide a means of generating carefully interpolated calibration, on variable user-defined timescales, that can be examined prior to application to the data with `setapply` and `correct`. The solutions for different fields and/or spectral windows can be interpolated in different ways, with all solutions stored in the same table.

The only difference between incremental and cumulative calibration tables is that incremental tables are generated directly from the data via `solve` or (in the near future) from other ancilliary data (e.g. weather information), and cumulative tables are generated from other cumulative and incremental tables.
via **accumulate**. In all other respects (internal format, application to data via **setapply** and **correct**, plotting with **plotcal**, etc.), they are the same, and therefore interchangeable. Thus, **accumulate** and cumulative calibration tables need only be used when circumstances require it.

The **accumulate** function represents a generalization on the classic AIPS CLCAL model of cumulative calibration in that its application is not limited to accumulation of “G” solutions (SN/CL tables classic AIPS are the analog of “G” (and, implicitly, “T”) in **aips++**). In principle, any basic calibration type can be accumulated (onto itself), as long as the result of the accumulation (matrix product) is of the same type. This is true of all the basic types, except “D”. Accumulation is currently supported for “B”, “G”, and “T”, and, in future, “F” (ionospheric Faraday rotation), “J” (generic full-polarization calibration), fringe-fitting, and perhaps others. Accumulation of certain specialized types (e.g., “GSPLINE”, “TOPAC”, etc.) onto the basic types will be supported in the near future. The treatment of various calibration from ancilliary data (e.g., system temperatures, weather data, WVR, etc.), as they become available, will also make use of **accumulate** to achieve the net calibration.

Note that accumulation only makes sense if treatment of a uniquely incremental solution is required (as described above), or if a careful interpolation or sampling of a solution is desired. In all other cases, re-solving for the type in question will suffice to form the net calibration of that type. For example, the product of an existing “G” solution and an amplitude and phase “G” self-cal (solved with the existing solution applied), is equivalent to full amplitude and phase “G” selfcal (with no prior solution applied), as long as the timescale of this solution is at least as short as that of the existing solution.

Use of **accumulate** is straightforward:

The **tablein** parameter is used to specify the existing cumulative calibration table to which an incremental table is to be applied. Initially, no such table exists, and **accumulate** will generate one from scratch (on-the-fly), using the timescale (in seconds) specified by the parameter **t**. These nominal solutions will be unit-amplitude, zero-phase (i.e., unit matrix) calibration, ready to be adjusted by accumulation. When **t** is negative (the default), the table name specified in **tablein** must exist and will be used.

The **incrtable** parameter is used to specify the incremental table that should be applied to **tablein**. The calibration type of **incrtable** sets the type assumed in the operation, so **tablein** must be of the same type. If it is not, **accumulate** will exit with an error message. (Certain combinations of types and subtypes will be supported by **accumulate** in the future.)

The **tableout** parameter is used to specify the name of the output table to write. If un-specified (or “”), then **tablein** will be overwritten. Use this feature with care, since an error here will require building up the cumulative table from the most recent distinct version (if any).

The **field** parameter specifies those field names (standard selection syntax) in **tablein** to which the incremental solution should be applied. The solutions
for other fields will be passed to `tableout` unaltered. If the cumulative table was created from scratch in this run of `accumulate`, then these solutions will be unit-amplitude, zero-phase, as described above.

The `calfield` parameter is used to specify the fields (standard selection syntax) to select from `incrtable` to use when applying to `tablein`. Together, use of `field` and `calfield` permit completely flexible combinations of calibration accumulation with respect to fields. Multiple runs of `accumulate` can be used to generate a single table with many combinations. In future, a “self” mode will be enabled that will simplify the accumulation of field-specific solutions.

The `interp` parameter is used to specify the interpolation type to use on the incremental solutions, as in `setapply`. The currently available interpolation types are “nearest”, “linear”, and “aipslin”. See the `setapply` URM documentation for more details.

The `spwmap` parameter enables accumulating solutions from differing spectral windows. See `setapply` for details on how `spwmap` works.

Pending improvements:

- Implement a “self” mode (independent of interpolation type), to simplify or eliminate use of the `field` and `calfield` parameters in some contexts (e.g., self-cal)
- More interpolation modes, e.g., “cubic”, and interpolation timescale (timerange to permit interpolation)
- Handle propagation (or not) of bad/flagged solutions
- Support of specialized types (e.g., TOPAC) onto the basic types
- Smoothing (probably a separate function)

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablein</td>
<td>Input cumulative calibration table name</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>incrtable</td>
<td>Input incremental calibration table name</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>tableout</td>
<td>Output cumulative calibration table name. Default is input table name.</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>field</td>
<td>List of fields (names) to update in input cumulative table. Default is all.</td>
<td>Any</td>
<td>Variant</td>
</tr>
<tr>
<td>calfield</td>
<td>List of fields (names) in incremental table to use. Default is use all.</td>
<td>Any</td>
<td>Variant</td>
</tr>
<tr>
<td>interp</td>
<td>Interpolation mode to use on incremental solutions</td>
<td>String</td>
<td>Linear</td>
</tr>
<tr>
<td>t</td>
<td>Cumulative table timescale when creating from scratch</td>
<td>Double</td>
<td>-1.0</td>
</tr>
<tr>
<td>spwmap</td>
<td>Spectral windows to apply</td>
<td>IntArray</td>
<td>-1</td>
</tr>
</tbody>
</table>

| Returns         | bool                                                                        |

**Example**

cb.open('ap366.sim');

# obtain G solutions from calibrator
cb.selectvis(msselect='FIELD_ID IN [9,11]');
cb.setsolve(type='G',table='cal.G0', t=300);
cb.solve()

# obtain proper flux density scale
cb.fluxscale (tablein='cal.G0', tableout='cal.G1',
                   reference='1328+307', transfer="0917+624");

# generate cumulative table for target source on 20s timescale
cb.accumulate(tablein='', incrtable='cal.G1', tableout='cal.cG0',
                   field='0957+561', calfield='0917+624',
                   interp='linear', t=20);

# apply this calibration to target
cb.selectvis(msselect='FIELD_ID==10');
cb.setapply(type='G', table='cal.cG0', interp='linear')
cb.correct();

# (image target with imager tool)

# phase-selfcal target on 60s timescale
cb.selectvis(msselect='FIELD_ID==10');
cb.setapply(type='G', table='cal.cG0', interp='linear')
cb.setsolve(type='G', table='cal.G2', t=60, phaseonly=T);
cb.solve();

# accumulate new solution onto existing one
cb.accumulate(tablein='cal.cG0', incrtable='cal.G2', tableout='cal.cG1',
                   field='0957+561', calfield='0957+561',
                   interp='linear');

# apply new cumulative solution to data
cb.setapply(type='G', table='cal.cG1', interp='linear')
cb.correct();

# (another round of imaging, etc.)

cb.close();
calibrater.activityrec.html

**calibrater.activityrec - Function**

2.1.1 Returns a record containing properties of recent activity

**Description**

This function enables returning generic information about recent activity.

Pending improvements:

- ??

**Arguments**

**Returns**

record

**Example**

TBD
calibrater.specifycal.html

**calibrater.specifycal - Function**

2.1.1 Externally specify calibration of various types

**Description**

This function enables specifying calibration parameters externally.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>caltable</td>
<td>The calibration table name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>Calibration timestamp</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>spw</td>
<td>Calibration spw(s)</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>antenna</td>
<td>Calibration antenna(s)</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>pol</td>
<td>Calibration polarization</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>caltype</td>
<td>Calibration timestamp</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>parameter</td>
<td>Calibration parameters</td>
<td>doubleArray</td>
<td></td>
</tr>
<tr>
<td>infile</td>
<td>Ancillary input file</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

1205
Example

cb.open('ap366.sim');

(TBD)

cb.close();
2.1.1 Produce a smoothed calibration table

Description

This function provides for time-dependent smoothing of sampled calibration solutions. Currently supported types are 'G', 'B', and 'T'. (Smoothing on the frequency axis for 'B' will be supported in the near future.)

Two (sliding) smoothing types are currently supported: 'median' or 'mean', one of these options should be specified in smoothtype. The full width (in seconds) of the smoothing filter should be specified in smoothtime. Amplitude and (ambiguity-corrected) phase are smoothed separately.

Use field to limit the smoothing operation to a subset of the fields (standard selection syntax) found in the calibration table (other fields will pass to the output table unsmoothed). If field is left blank, all fields in the table will be smoothed.

The smoothing is always done independently for each field, but scan boundaries are not observed. Thus, if the smoothtime is large enough, smoothing may occur over many boundaries.

Flagged solutions in the input table will not participate in the smoothing calculation, but will be replaced with smoothed values if the smoothing window covers one or more unflagged solutions when centered on the flagged point.

Pending improvements:

- Add other smoothtypes?
- Add spw and other selection on input table
- Add A/P toggle

Arguments
### Inputs

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tablein</code></td>
<td>Input calibration table</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td><code>tableout</code></td>
<td>Output calibration table</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td><code>field</code></td>
<td>Limit smoothing to these fields (default is all fields)</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td><code>smoohtype</code></td>
<td>The smoothing type: 'mean' or 'median'</td>
<td>string</td>
<td>mean</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>median</td>
</tr>
<tr>
<td><code>smoothtime</code></td>
<td>Smoothing filter time constant (sec)</td>
<td>double</td>
<td>60.0</td>
</tr>
</tbody>
</table>

### Returns

bool

### Example

```python
cb.open('ngc5921.ms');
cb.smooth(tablein='in.gcal',tableout='out.gcal',
          smoohtype='median',smoothtime=60);
cb.close();
```

In this example, 'G' solutions for all fields in the table 'in.gcal' are smoothed using a median filter with a full-width of 60 seconds, and the result written to 'out.gcal'.
calibrator.listcal.html

**calibrator.listcal - Function**

2.1.1 List the contents of a calibration table

**Description**

calibrator.listcal() lists antenna gain solutions in tabular form. The table is organized as follows. Solutions are output by

1. Spectral window,
2. Antenna,
3. Time,
4. Channel,
5. and Polarization.

The inner-most loop is over polarization. A “Spw Header” row is printed each time the spectral window ID (SpwID), the Spw Header also lists the date of observation (Date), the calibration table name (CalTable), and the measurement set name (MS name).

A lower-level “antenna header” is printed each time the antenna names change or every ‘pagerows’ of output, whichever comes first. The antenna header column are described here:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ant</td>
<td>Antenna name</td>
</tr>
<tr>
<td>Time</td>
<td>Visibility timestamp corresponding to gain solution</td>
</tr>
<tr>
<td>Field</td>
<td>Field name</td>
</tr>
<tr>
<td>Chn</td>
<td>Channel number</td>
</tr>
<tr>
<td>Amp</td>
<td>Complex solution amplitude</td>
</tr>
<tr>
<td>Phs</td>
<td>Complex solution phase</td>
</tr>
<tr>
<td>F</td>
<td>Flag</td>
</tr>
</tbody>
</table>

Elements of the “F” column contain an ‘F’ when the datum is flagged, and ‘ ‘ (whitespace) when the datum is not flagged.

Presently, the polarization mode names (for example: R, L) are not given, but the ordering of the polarization modes (left-to-right) is equivalent to the order output by task listobs (see “Feeds” in listobs output).

**Arguments**
Inputs

caltablename Calibration table to list
  allowed: string
  Default:
field Field names or indices to list: "" => all
  allowed: any
  Default: variant
antenna Antenna/Baseline to list: "" => all
  allowed: any
  Default: variant
spw Spectral windows and channels: "" => all,
  spw=’10:8~20’
  allowed: any
  Default: variant
listfile Send output to file: "" => send to terminal)
  allowed: string
  Default:
pagerows Rows per page
  allowed: int
  Default: 50

Returns

bool

Example

Input:

The following example imports a UVFITS file, performs a bandpass calibration, and displays a subset of the resulting calibration table.

pathname=os.environ.get(‘CASAPATH’).split()[0] # Get path to CASA home dir
fitsdata=pathname+/data/demo/NGC5921.fits’ # Select uv-data (FITS) file
msdata=’NGC5921.ms’ # MS name; write to current directory
importuvfits(fitsfile=fitsdata, vis=msdata) # import FITS data to MS
setjy(vis=msdata) # Create model data for flux calibrator
caldata=msdata+'.bcal' # Calibration table name
bandpass(vis=msdata, caltable=caldata) # Bandpass calibration
cb.open(msdata) # Open MS in cb
cb.listcal(caltable=caldata, field='N5921_2, 0, 1', antenna='1~5;10~13;20~22', spw='0:4~6', pagerows=0)

Output:

SpwID = 0, Date = 1995/04/13, CalTable = NGC5921.ms.bcal (B Jones), MS name = /users/jcrossle/NRAO/casa/NGC5921.ms

<table>
<thead>
<tr>
<th>Time</th>
<th>Field</th>
<th>Chn</th>
<th>Ant = 1</th>
<th>Ant = 2</th>
<th>Ant = 3</th>
<th>Ant = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:21:46.0</td>
<td>1331+30500002_0</td>
<td>4</td>
<td>0.294  3.5</td>
<td>0.296  105.9</td>
<td>0.287 -111.9</td>
<td>0.276 -66.0</td>
</tr>
<tr>
<td>09:21:46.0</td>
<td>1331+30500002_0</td>
<td>5</td>
<td>0.303  0.6</td>
<td>0.305  107.0</td>
<td>0.298 -111.6</td>
<td>0.280 -24.5</td>
</tr>
<tr>
<td>10:05:27.9</td>
<td>1445+09900002_0</td>
<td>4</td>
<td>0.467  7.6</td>
<td>0.473  107.7</td>
<td>0.455 -112.3</td>
<td>0.427 165.8</td>
</tr>
<tr>
<td>10:05:27.9</td>
<td>1445+09900002_0</td>
<td>5</td>
<td>0.472  0.0</td>
<td>0.486  109.1</td>
<td>0.471 -111.8</td>
<td>0.420 166.8</td>
</tr>
<tr>
<td>10:05:27.9</td>
<td>1445+09900002_0</td>
<td>6</td>
<td>0.486  8.4</td>
<td>0.482  110.0</td>
<td>0.478 -111.4</td>
<td>0.439 167.8</td>
</tr>
<tr>
<td>09:21:46.0</td>
<td>1331+30500002_0</td>
<td>4</td>
<td>0.261 -26.0</td>
<td>0.285 -107.1</td>
<td>0.279 -149.5</td>
<td>0.263 2.9</td>
</tr>
<tr>
<td>09:21:46.0</td>
<td>1331+30500002_0</td>
<td>5</td>
<td>0.269 -26.1</td>
<td>0.295 -107.2</td>
<td>0.288 -148.8</td>
<td>0.274 2.3</td>
</tr>
<tr>
<td>10:05:27.9</td>
<td>1445+09900002_0</td>
<td>4</td>
<td>0.416 -24.0</td>
<td>0.450 -106.4</td>
<td>0.437 -147.3</td>
<td>0.414 3.2</td>
</tr>
<tr>
<td>10:05:27.9</td>
<td>1445+09900002_0</td>
<td>5</td>
<td>0.421 -22.6</td>
<td>0.478 -106.1</td>
<td>0.463 -147.4</td>
<td>0.443 2.0</td>
</tr>
<tr>
<td>10:05:27.9</td>
<td>1445+09900002_0</td>
<td>6</td>
<td>0.436 -22.7</td>
<td>0.478 -106.7</td>
<td>0.459 -146.6</td>
<td>0.443 2.4</td>
</tr>
<tr>
<td>09:21:46.0</td>
<td>1331+30500002_0</td>
<td>4</td>
<td>0.261 -26.0</td>
<td>0.285 -107.1</td>
<td>0.279 -149.5</td>
<td>0.263 2.9</td>
</tr>
<tr>
<td>09:21:46.0</td>
<td>1331+30500002_0</td>
<td>5</td>
<td>0.269 -26.1</td>
<td>0.295 -107.2</td>
<td>0.288 -148.8</td>
<td>0.274 2.3</td>
</tr>
<tr>
<td>10:05:27.9</td>
<td>1445+09900002_0</td>
<td>4</td>
<td>0.416 -24.0</td>
<td>0.450 -106.4</td>
<td>0.437 -147.3</td>
<td>0.414 3.2</td>
</tr>
<tr>
<td>10:05:27.9</td>
<td>1445+09900002_0</td>
<td>5</td>
<td>0.421 -22.6</td>
<td>0.478 -106.1</td>
<td>0.463 -147.4</td>
<td>0.443 2.0</td>
</tr>
<tr>
<td>10:05:27.9</td>
<td>1445+09900002_0</td>
<td>6</td>
<td>0.436 -22.7</td>
<td>0.478 -106.7</td>
<td>0.459 -146.6</td>
<td>0.443 2.4</td>
</tr>
<tr>
<td>09:21:46.0</td>
<td>1331+30500002_0</td>
<td>4</td>
<td>0.261 -26.0</td>
<td>0.285 -107.1</td>
<td>0.279 -149.5</td>
<td>0.263 2.9</td>
</tr>
<tr>
<td>09:21:46.0</td>
<td>1331+30500002_0</td>
<td>5</td>
<td>0.269 -26.1</td>
<td>0.295 -107.2</td>
<td>0.288 -148.8</td>
<td>0.274 2.3</td>
</tr>
<tr>
<td>10:05:27.9</td>
<td>1445+09900002_0</td>
<td>4</td>
<td>0.416 -24.0</td>
<td>0.450 -106.4</td>
<td>0.437 -147.3</td>
<td>0.414 3.2</td>
</tr>
<tr>
<td>10:05:27.9</td>
<td>1445+09900002_0</td>
<td>5</td>
<td>0.421 -22.6</td>
<td>0.478 -106.1</td>
<td>0.463 -147.4</td>
<td>0.443 2.0</td>
</tr>
<tr>
<td>10:05:27.9</td>
<td>1445+09900002_0</td>
<td>6</td>
<td>0.436 -22.7</td>
<td>0.478 -106.7</td>
<td>0.459 -146.6</td>
<td>0.443 2.4</td>
</tr>
<tr>
<td>09:21:46.0</td>
<td>1331+30500002_0</td>
<td>4</td>
<td>0.261 -26.0</td>
<td>0.285 -107.1</td>
<td>0.279 -149.5</td>
<td>0.263 2.9</td>
</tr>
<tr>
<td>09:21:46.0</td>
<td>1331+30500002_0</td>
<td>5</td>
<td>0.269 -26.1</td>
<td>0.295 -107.2</td>
<td>0.288 -148.8</td>
<td>0.274 2.3</td>
</tr>
<tr>
<td>10:05:27.9</td>
<td>1445+09900002_0</td>
<td>4</td>
<td>0.416 -24.0</td>
<td>0.450 -106.4</td>
<td>0.437 -147.3</td>
<td>0.414 3.2</td>
</tr>
<tr>
<td>10:05:27.9</td>
<td>1445+09900002_0</td>
<td>5</td>
<td>0.421 -22.6</td>
<td>0.478 -106.1</td>
<td>0.463 -147.4</td>
<td>0.443 2.0</td>
</tr>
<tr>
<td>10:05:27.9</td>
<td>1445+09900002_0</td>
<td>6</td>
<td>0.436 -22.7</td>
<td>0.478 -106.7</td>
<td>0.459 -146.6</td>
<td>0.443 2.4</td>
</tr>
</tbody>
</table>
Listed 108 antenna solutions.
calibrater.posangcal.html

**calibrater.posangcal - Function**

2.1.1 Apply position angle calibration to an existing cal table

**Description**

This function is used to apply position angle calibration for observations made using circularly polarized feeds. According to the Measurement Equation formalism, this correction should be applied to a \( D \) (instrumental polarization) calibration table.

If no \( D \) calibration is performed (and thus no such table is available), the correction can be applied to a \( G \) table, but it should NEVER be applied to both, and always applied to a \( D \) table if one is available. An input table must be specified. If no output table is specified, then the input table will be modified in place.

Specify, as a vector of values, a position angle adjustment (in degrees) for each spectral window. If only one value is specified, it will be duplicated to all spectral windows; otherwise, the number of values specified must match the number of spectral windows. The sign convention for the position angle adjustment is such that the specified value is the that which, when added to the position angle implied by the data, will yield the correct position angle. For example, if \( G^- \), \( D^- \), and \( P^- \)-calibrated data for 3c286 suggests a position angle of 45 degrees, the posangcor value should be -12 degrees as this will yield the correct position angle of 33 degrees when added. In general, posangcor equals correct position angle minus observed position angle.

A future version of this function will have an option to recognize standard position angle calibrators and determine the correction automatically.

(NB: It may be desirable to use solutions for 'X' to handle position angle calibration, rather than this method.)

**Arguments**
Inputs

<table>
<thead>
<tr>
<th>posangcor</th>
<th>Position angle corrections (degrees)</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>doubleArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tablein</th>
<th>Input calibration table name</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tableout</th>
<th>Output calibration table name. Default is input table name.</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool

Example

cb.open('polcal.ms');
cb.posangcal(tablein='3C286.dcal', tableout='3C286.dpacal',
             posangcor=[-12.0, 54.0]);
cb.close();

This example takes an existing calibration table containing \{\tt D\} Jones matrices, and applies a position angle calibration of 45 and 54 degrees to spectral windows 1 & 2, respectively, writing the result to a new table. The observed position angles for 3C286 must have been 45 and -21 degrees; the corrections specified yield the correct value of 33 degrees when added to the observed values.
calibrater.linpolcor.html

**calibrater.linpolcor - Function**

2.1.1 Correct the gain table for linear polarization of the calibrator

**Description**

THIS METHOD IS CURRENTLY DISABLED.
This function can be used to correct the gains derived from secondary calibrators with unknown or variable polarization. It should only be used for arrays with linear (X/Y) feeds and an Alt-Az mount for which the observed polarization varies with feed position angle on the sky. The function fits the gains with a sine and cosine term in feed position angle and extracts the Q and U components of the secondary calibrator. This is only possible if there is sufficient range in the position angle (i.e., minimum of about 6 scans spanning at least 90 degrees in position angle). Check the error of the fit to judge if the fit was successful, it should generally be smaller than 0.5%. Use the `fields` argument to select calibrator fields to be fitted. The function takes a calibration table as input, and can write the adjusted gain solutions to the same table on output, or create a new table containing these results. The function also prints the derived polarization for each field for each spectral window.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablein</td>
<td>Input calibration table name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>tableout</td>
<td>Output calibration table name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>fields</td>
<td>Calibrator field names</td>
<td>stringArray</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool
Example

cb.open('atca.ms');
cb.linpolcor(tablein='atca.gcal', tableout='atca.gcal2',
    fields='2254-367');
cb.close();

This example takes an existing calibration table containing `$G$` Jones matrices, and writes a corrected output table, correcting only gains derived from 2254-367 for linear polarization.
calibrater.plotcal.html

**calibrater.plotcal - Function**

2.1.1 Plot a calibration table

**Description**

This function plots a calibration table either to a plotter or to a file. The argument `plottype` can take the following values for all types of solutions:

- **AMP**  Gain Amplitude vs. Time
- **1/AMP** Inverse Gain Amplitude vs. Time (useful for comparing with classic AIPS)
- **PHASE** Gain Phase vs. Time
- **RI**  Gain Real vs. Imaginary
- **RLPHASE** Right/Left Gain phase difference (if polarizations are R,L)
- **XYPHASE** X/Y Gain phase difference (if polarizations are X,Y)

The argument `plottype` can take the following values for D tables:

- **DAMP**  Cross-polarized Gain Amplitude vs. Time
- **DPHASE** Cross-polarized Gain Phase vs. Time
- **DRI**  Cross-polarized Gain Real vs. Imaginary

The quality of the solutions can be examined with the following `plottype` choices:

- **FIT**  Fit per spectral window
- **FITWGT** Fit weight per spectral window
- **TOTALFIT** Total fit

By default, all antennas (as specified in the `antennas` argument) will appear on the same plot. Separate plots (all with the same scale) for each antenna can be activated by setting `multiplot=T`. The `multiplot` argument only separates plots by antenna (not, e.g., by the `field_id(s)` specified in the `fields` argument). If `multiplot=T`, the `nx` and `ny` arguments can be used to specify the number of plots per page.
At the moment, only one polarization can be plotted per execution. This restriction will be relaxed in the near future. For B solutions, the plotting will loop over timestamps (if more than one). A hardcopy plot can be created by specifying the \texttt{psfile} argument (which is especially useful for batch processing when a display screen is not available). This will cause the plot to be written to a PostScript file which can be subsequently sent to a printer.

\textbf{Arguments}
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>antennas</strong></td>
<td>Antennas to plot. Default is none.</td>
</tr>
<tr>
<td>allowed</td>
<td>intArray</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td><strong>fields</strong></td>
<td>Fields to plot. Default is none.</td>
</tr>
<tr>
<td>allowed</td>
<td>intArray</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td><strong>spwids</strong></td>
<td>Spectral windows id.’s to plot. Default is none.</td>
</tr>
<tr>
<td>allowed</td>
<td>intArray</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td><strong>plottype</strong></td>
<td>Plot type</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>1/AMP, PHASE, RLPHASE, XYPHASE, RI, DAMP, DPHASE, DRI, FIT, FITWGT, TOTALFIT, AMP</td>
</tr>
<tr>
<td><strong>tablename</strong></td>
<td>Calibration table name</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td><strong>polarization</strong></td>
<td>Polarization to plot</td>
</tr>
<tr>
<td>allowed</td>
<td>int</td>
</tr>
<tr>
<td>Default</td>
<td>1</td>
</tr>
<tr>
<td><strong>multiplot</strong></td>
<td>Turn on separate antenna plots</td>
</tr>
<tr>
<td>allowed</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td><strong>nx</strong></td>
<td>If multiplot=T, number of plots on horizontal axis</td>
</tr>
<tr>
<td>allowed</td>
<td>int</td>
</tr>
<tr>
<td>Default</td>
<td>1</td>
</tr>
<tr>
<td><strong>ny</strong></td>
<td>If multiplot=T, number of plots on vertical axis</td>
</tr>
<tr>
<td>allowed</td>
<td>int</td>
</tr>
<tr>
<td>Default</td>
<td>1</td>
</tr>
<tr>
<td><strong>psfile</strong></td>
<td>Name of output PostScript file to write plot to. Default is to send plot to the screen.</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

1219
bool

Example

cb.open('ngc5921.ms');
cb.plotcal(plottype='PHASE', tablename="gcal", antennas=[1,3], polarization=2);
cb.close();
Description

This method fits single-component models (points, elliptical Gaussians or elliptical Disks, to the CORRECTED_DATA of the selected field. A first guess for the component parameters may be specified in the `par` parameter.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>vary</td>
<td>If specified where T, let this parameter (in par) vary in fit</td>
<td>boolArray</td>
<td></td>
</tr>
<tr>
<td>niter</td>
<td>Number of non-linear fitting iterations</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>compshape</td>
<td>Component shape, P=point G=gaussian</td>
<td>string</td>
<td>P</td>
</tr>
<tr>
<td>par</td>
<td>Initial guess for fit parameters (default is for ”P”)I flux, rel RA, rel Dec, 1.0, 0.0, 0.0 are defaults</td>
<td>doubleArray</td>
<td>1.0 0.0 0.0</td>
</tr>
<tr>
<td>file</td>
<td>If specified, output componentslist file name, if empty don’t write componentslist file</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

Returns
doubleArray

Example

1221
This example fits a point source mode using 15.0 Jy at the origin (phase center) as a first guess.

This example fits a Guassian model with a starting guess of 15 Jy at the phase center (0,0), with 2.0 arcsec major axis, 1.0 axial ratio, at position angle 0.0 deg.
**calibrater.createcaltable** - Function

2.1.1 Create an empty calibration table

**Description**

Creates an empty calibration table that can subsequently be filled with by using the table tool.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>caltable</td>
<td>Calibration table name</td>
<td>allowed: string</td>
<td>Default:</td>
</tr>
<tr>
<td>partype</td>
<td>Parameter type name</td>
<td>allowed: string</td>
<td>Default:</td>
</tr>
<tr>
<td>caltype</td>
<td>Calibration type name</td>
<td>allowed: string</td>
<td>Default:</td>
</tr>
<tr>
<td>singlechan</td>
<td>Create a calibration table with frequency-independent parameters?</td>
<td>allowed: bool</td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
cb.open('ngc5921.ms',F,F,F)  # don’t add MODEL_DATA, etc.
cb.createcaltable('ngc5921.gcal', 'Complex', 'G Jones', True)
cb.close()
```
calibrater.updatecaltable - Function

2.1.1 Caltable modernizer.

Description

This method can be used to update a caltable (from v3.4 or later) to the current version of CASA.
The following updates are currently supported.
- At CASA v4.1.0, the OBSERVATION subtable and OBSERVATION_ID column were added to caltables. This method adds trivial versions of these elements to pre-v4.1 caltables.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>caltable</td>
<td>string</td>
<td>Name of the caltable.</td>
</tr>
</tbody>
</table>

Returns

bool

Example

cb.updatecaltable("mycaltable");
calibrater.close.html

**calibrater.close - Function**

**2.1.1** Close the calibrater tool

**Description**

Close the `calibrater` tool, which is hardly ever necessary.

**Arguments**

**Returns**

`bool`

**Example**

```cpp
cb.open('ngc5921.ms');
cb.close();
```
calibrater.done.html

**calibrater.done - Function**

2.1.1 Destroy the calibrater tool

**Description**

This function is redundant with the `close` method.

**Arguments**

**Returns**

`bool`

**Example**

```plaintext
cb.open('ngc5921.ms');
cb.done();
```
2.1.2 calanalysis - Tool

Get and fit data from a calibration table (CASA 3.4 and later).

Requires:

Synopsis

Description

Overall Description

The calibration analysis (ca) tool is a standardized interface to the new format (CASA 3.4 and later) calibration tables. It is designed to handle all types of tables, e.g., gain, bandpass, Tsys, etc. The ca tool takes advantages of newly implemented features in the CASA C++ code tree, e.g., iteration and parameter selection, which means that calibration tables can be accessed in a manner very similar to measurement sets.

The ca tool was originally designed to facilitate getting and/or processing data from an entire calibration table in an organized fashion so that the information could be written to other files. Additional features, e.g. introspective member functions, were added so that scripters and general users can easily employ the ca tool without using iteration (get one piece of data at a time).

Like the imaging tool (im), image analysis tool (ia), calibration tool (cb), etc., a native calibration analysis tool (ca) is created when CASA starts up. Other instances of the calibration analysis tool can be created, if required, using the command:

```
from casac import casac
caloc = casac.calanalysis()
```

Open/Close Member Functions

The purpose of the open() and close() member functions is obvious, i.e., they open and close the new format calibration table. Each ca tool instance can have only one open table. If no table has been opened, the other member functions don’t return anything.

The member function definitions are:

- `ca.open( ‘caltable name’ )` - This member function opens the calibration table. If successful True is returned, otherwise False is returned.
- `ca.close()` - This member function closes the calibration table. If a table was open True is returned, otherwise False is returned.
Introspective Member Functions

The introspective member functions provide information about the shape and contents of the file. For example, the numchannel() member function returns the number of channels corresponding to each spectral window. Also, the field() member function returns the field names or numbers. With this information, users can easily select and keep track of limited regions of the calibration table, even from the command line, by minimizing the number of iterations.

The member function definitions are:

- ca.antenna(name=True) - This member function returns the antenna numbers or names as a python list of strings.
- ca.calname() - This member function returns the new format calibration table name as a python string.
- ca.feed() - This member function returns the feed names as a python list of strings (‘X’, ‘Y’ for linear; ‘R’, ‘L’ for circular; ‘S’ for ”scalar”, when the calibration solutions are performed simultaneously for both polarizations). If the basis is unknown, then the basis functions are ‘1’ and ‘2’. This kludge was added to handle incomplete calibration tables.
- ca.field(name=True) - This member function returns the field numbers or names as a python list of strings.
- ca.freq() - This member function returns the frequencies in the table as a python dictionary (the keys are the spectral window numbers and the elements are numpy float arrays containing the frequencies).
- ca.msname() - This member function returns the parent measurement set name as a python string.
- ca.numantenna() - This member function returns the number of antennas as a python integer.
- ca.numchannel() - This member function returns the number of channels for each spectral window as a list of python integers.
- ca.numfeed() - This member function returns the number of feeds as a python integer.
- ca.numfield() - This member function returns the number of fields as a python integer.
- ca.numschw() - This member function returns the number of spectral windows as a python integer.
- ca.numtime() - This member function returns the number of times as a python integer.
- ca.partype() - This member function returns the parameter column type (‘Float’ or ‘Complex’).
- ca.polbasis() - This member function returns the polarization basis (‘L’ for linear or ‘C’ for circular). If the basis is unknown, ‘U’ is returned. This kludge was added to handle incomplete calibration tables.
- ca.spw(name=True) - This member function returns the spectral window numbers or names as a python list of strings.
ca.time() - This member function returns the times as a python list of floats (in units of MJD seconds). In the future, date strings will be available.

ca.viscal() - This member function returns the type of new formation calibration table as a python string. For example 'B' is a bandpass table, 'G' is a gain table, etc.

Process Member Functions

The process member functions process data. As of CASA 3.4, there are two: get() and fit(). The get() member function iterates through the calibration table and returns the selected data. The fit() member function does the same as the get() member function and returns the fits as well. Tables with complex parameters are converted to either amplitudes or phases.

The get() and fit() member functions employ two levels of iteration. The first level involves field, antenna 1, and antenna 2 (slowest to fastest). The data for each first-level iteration are placed in a cube whose dimensions are feed x frequency x time. Two of these dimensions represent the second level of iteration. The feed axis is always an iteration axis, and the user can select either frequency or time as the other one.

In addition to providing a logical way of getting data, this two-level iteration scheme also allows users to fit along the non-iteration axis. For example, a fit can be performed along the frequency axis for each iteration of the selected field, antenna 1, antenna 2, feed, and time.

Inputs

As mentioned above, the selection syntax originally designed for measurement sets has been implemented in the ca tool. It is available for feed, antenna 1 and 2, and spectral window with channel. For more information, consult the selection documentation. Their argument lists are:

ca.get( field=", antenna=", timerange=[], spw=", feed=", axis='TIME', ap='AMPLITUDE', norm=True, unwrap=True, jumpmax=0.0 )
ca.fit( field=", antenna=", timerange=[], spw=", feed=", axis='TIME', ap='AMPLITUDE', norm=True, unwrap=True, jumpmax=0.0, order='AVERAGE', type='LSQ', weight=False )

'*' is equivalent to ". Both numbers and names can be used for field, antenna, and spw. Names have not been implemented in present EVLA and ALMA datasets for some quantities. Check if they are available using the introspective methods.

The least-squares fit is quite standard. The robust fit, which minimizes the effects of outliers, is experimental. Robust fits are simple to compute, but they don’t provide parameter variances and covariances. To minimize outliers and obtain (co)variances, the following algorithm is used:
- Calculate the least-squares fit.
- Using the fit parameters from the least squares fit as starting values, perform
the robust fit (which is essentially a zero-finding algorithm).
- Flag all outliers with residuals greater than 5 times the mean deviation.
  These flags are actually returned, so they can be applied elsewhere.
- Recalculate the least-squares fit without the outliers.

Arguments for get() and fit():

field = A comma-delimited string or a python list of strings containing the fields. E.g., field = '0,1'. The default is " (all fields).

antenna = A comma- and semi-colon- delimited string containing the antenna 1s and antenna 2s. E.g, antenna = '3,4,5'. The default is " (all antenna 1s and antenna 2s).

timerange = A python list of floats of length two containing the start and stop times in MJD seconds. Date strings will be implemented in a future release when they are implemented in the selection C++ code. E.g., timerange = [456123.0,456456.0]. The default is [min MJD, max MJD]. For convenience, the MJD times can be obtained from the time() introspective method.

spw = A comma- and semi-column- delimited string containing the spectral window and channel selection. E.g., spw = '0:4~20:25~59,2:10~30,6'. The default is " (all spectral windows and channels).

feed = A comma-delimited string or python list of strings containing the feed names (X’, Y’, R’, L’, or ’S’ [scalar]). E.g., feed='X,Y'. The default is " (all feeds).

axis = A python string containing the user-defined iteration axis ('TIME' or 'FREQ'). E.g., axis='FREQ'. The default is 'TIME' (the frequency axis is a non-iteration axis).

ap = A python string containing the amplitude/phase selection ('AMPLITUDE' or 'PHASE'). E.g., ap = 'PHASE'. The default is 'AMPLITUDE'. It is ignored if the parameters in the calibration table are real.

norm = A python boolean which determines whether amplitudes are normalized for each iteration. E.g., norm = False. The default is True. It is ignored if the parameters in the calibration table are real or ap = 'PHASE'.

unwrap = A python boolean which determines whether phases are unwrapped for each iteration. E.g., unwrap = False. The default is True. It is ignored if the parameters in the calibration table are real or ap = 'PHASE'. It is ignored if the parameters in the calibration table are real or ap = 'PHASE'.

jumpmax = A python float which determines the maximum phase jump near +/- PI before unwrapping is performed. E.g., jumpmax = 0.1. The default is 0.0. It is ignored if the parameters in the calibration table are real or ap = 'PHASE'.

Arguments for fit() only:

order = A python string containing the fit order ('AVERAGE', 'LINEAR', or 'QUADRATIC'). E.g., order = 'LINEAR'. The default is 'AVERAGE'.

'QUADRATIC' is not available when the fit type is 'ROBUST'.

1230
type = A python string containing the fit type ('LSQ' or 'ROBUST'). E.g., type = 'ROBUST'. The default is 'LSQ'. The robust fit, which minimizes the effects of outliers, is experimental. Robust fits are simple to compute, but they don’t provide parameter variances and covariances. To minimize outliers and obtain (co)variances, the following algorithm is used:
- Calculate the least-squares fit.
- Using the fit parameters from the least squares fit as starting values, perform the robust fit (which is essentially a zero-finding algorithm).
- Flag all outliers with residuals greater than 5 times the mean deviation. These flags are actually returned, so they can be applied elsewhere.
- Recalculate the least-squares fit without the outliers.

weight = A python boolean which determines whether weights are applied. E.g., weight = True. The default is False.

Outputs

The get() and fit() member function return dictionaries of dictionaries. They both return this information (the '#' represents the iteration number):
- ['#'][field] = The python string containing the field number.
- ['#'][antenna1] = The python string containing the antenna 1 number.
- ['#'][antenna2] = The python string containing the antenna 2 number.
- ['#'][feed] = A python string containing the feed.
- ['#'][value] = The numpy float array containing the parameters (either along the time or frequency axis) from the new format calibration table (if the table contains complex numbers, these numbers are either amplitudes or phases).
- ['#'][valueErr] = The numpy float array containing the parameter errors (either along the time or frequency axis) from the new format calibration table (if the table contains complex parameters, these numbers are either amplitude or phase errors).
- ['#'][flag] = The numpy boolean array containing the parameter flags.
- ['#'][abscissa] = The python string containing the name of the non-iteration axis ('frequency' or 'time').
- ['#'][frequency] = The numpy float array containing the frequencies. If the frequency axis is not an iteration axis, the frequencies correspond to the values, value errors, and flags. If the frequency axis is an iteration axis, this array has only one value.
- ['#'][time] = The numpy float array containing the times. If the time axis is not an iteration axis, the times correspond to the values, value errors, and flags. If the time axis is an iteration axis, this array has only one value.
- ['#'][rap] = The python string containing 'REAL', 'AMPLITUDE', or 'PHASE', describing the values and their errors.
- ['#'][norm] = The python boolean determining whether the amplitudes are normalized per iteration or not. It is not present for 'REAL' or 'PHASE' data.
- ['#'][unwrap] = The python boolean determining whether the phases are unwrapped per iteration or not. It is not present for 'REAL' or 'AMPLITUDE' data.
[
'jumpmax'] = The python float containing the maximum phase jump near +/- PI before unwrapping is performed. It is not present for 'REAL' or 'AMPLITUDE' data. If the non-iteration axis is 'frequency':
- if jumpmax == 0.0, fringe fitting was used (only available when the non-iteration axis is time).
- if jumpmax != 0.0, simple unwrapping was unused (same algorithm as used when the non-iteration axis is time or frequency).

In addition to these entries, the fit() member function returns these:
[
'order'] = The python string describing the fit order ('AVERAGE', 'LINEAR', or 'QUADRATIC'). 'QUADRATIC' is not available for 'ROBUST' fitting.
[type] = The python string containing the fit type ('LSQ' or 'ROBUST').
[weight] = The python boolean determining whether the fit was weighted or not.
[validFit] = The python boolean telling whether the fit was valid or not.
[pars] = The numpy float array containing the fit parameters.
[vars] = The numpy float array containing the fit parameter variances.
[covars] = The numpy float array containing the fit parameter covariances (par0-par1, par0-par2, ..., par1-par2).
[redChi2] = The python float containing the reduced chi2 (set to 1.0 for unweighted fits).
[model] = The numpy float array containing the model versus the abscissa.
[res] = The numpy float array containing the fit residuals versus the abscissa.
[resMean] = The python float containing the mean of the residuals.
[resVar] = The python float containing the variance of the residuals.

Methods

- calanalysis: Construct a calibration analysis tool.
- open: Open a calibration table.
- close: Close a calibration table.
- calname: Return the calibration table name.
- muname: Return the name of the MS that created this calibration table.
- viscal: Return the type of calibration table ('B', 'G', 'T', etc.).
- partype: Return the parameter column type in the calibration table ('Complex' or 'Float').
- polbasis: Return the polarization basis in the calibration table ('L' for linear or 'C' for circular).
- numfield: Return the number of fields in the calibration table.
- field: Return the fields in the calibration table.
- numantenna: Return the number of antennas in the calibration table.
- numantenna1: Return the number of antenna 1s in the calibration table.
- numantenna2: Return the number of antenna 2s in the calibration table.
- antenna: Return the antennas in the calibration table.
- antenna1: Return the antenna 1s in the calibration table.

1232
antenna2  Return the antenna 2s in the calibration table.
numfeed  Return the number of feeds in the calibration table.
feed  Return the feeds in the calibration table.
numtime  Return the number of times in the calibration table.
time  Return the times (in MJD seconds) in the calibration table.
numspw  Return the number of spectral windows in the calibration table.
spw  Return the spectral windows in the calibration table.
numchannel  Return the number of channels per spectral window in the calibration table.
freq  Return the frequencies per spectral window in the calibration table.
get  Return the calibration data.
fit  Return the calibration data and fits along the non-iteration axis.
Construct a calibration analysis tool.

**Description**

Construct a calibration analysis tool.

**Arguments**

**Returns**

calanalysisobject
calanalysis.open.html

**calanalysis.open - Function**

2.1.2 Open a calibration table.

**Description**

This member function opens a calibration table.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Python string containing the calibration table name.</th>
</tr>
</thead>
<tbody>
<tr>
<td>caltable</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
ca.open( '<caltable name>' )
```
calanalysis.close.html

**calanalysis.close** - Function

**2.1.2** Close a calibration table.

**Description**

This member function closes a calibration table.

**Arguments**

**Returns**

`bool`

**Example**

```cpp
ca.close()
```
Return the calibration table name.

**Description**

This member function returns calibration table name.

**Arguments**

**Returns**

string

**Example**

```python
caltable = ca.calname()
```
Return the name of the MS that created this calibration table.

**Description**

This member function returns the name of the MS that created this calibration table.

**Arguments**

**Returns**

string

**Example**

```plaintext
msname = ca.msname()
```
2.1.2 Return the type of calibration table ('B', 'G', 'T', etc.).

Description

This member function returns the type of calibration table ('B', 'G', 'T', etc.).

Arguments

Returns

string

Example

viscal = ca.viscal()
Return the parameter column type in the calibration table ("Complex" or "Float").

Description

This member function returns the parameter column type in the calibration table ("Complex" or "Float").

Arguments

Returns

string

Example

partype = ca.partype()
2.1.2 Return the polarization basis in the calibration table ('L' for linear or 'C' for circular).

Description

This member function returns the polarization basis in the calibration table ('L' for linear or 'C' for circular).

Arguments

Returns

string

Example

polbasis = ca.polbasis()
Return the number of fields in the calibration table.

This member function returns the number of fields in the calibration table.

Returns

int

Example

numfield = ca.numfield()
**calanalysis.field**

**calanalysis.field - Function**

2.1.2 Return the fields in the calibration table.

**Description**

This member function returns the fields in the calibration table.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The python boolean which determines whether field names (True) or field numbers (False) are returned.</td>
</tr>
<tr>
<td>allowed</td>
<td>boolean</td>
</tr>
<tr>
<td>Default</td>
<td>true</td>
</tr>
</tbody>
</table>

**Returns**

stringArray

**Example**

```python
field = ca.field()
```

---

1243
Return the number of antennas in the calibration table.

Description

This member function returns the number of antennas in the calibration table.

Arguments

Returns

int

Example

numantenna = ca.numantenna()
Return the number of antenna 1s in the calibration table.

**Description**

This member function returns the number of antenna 1s in the calibration table.

**Arguments**

**Returns**

`int`

**Example**

```plaintext
numantenna1 = ca.numantenna1()
```
2.1.2 Return the number of antenna 2s in the calibration table.

Description

This member function returns the number of antenna 2s in the calibration table.

Arguments

Returns

int

Example

```python
numantenna2 = ca.numantenna2()
```
calanalysis.antenna.html

calanalysis.antenna - Function

2.1.2 Return the antennas in the calibration table.

Description

This member function returns the antennas in the calibration table.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The python boolean which determines whether antenna names (True) or antenna numbers (False) are returned.</td>
</tr>
<tr>
<td>allowed:</td>
<td>boolean</td>
</tr>
<tr>
<td>Default:</td>
<td>true</td>
</tr>
</tbody>
</table>

Returns

stringArray

Example

```python
antenna = ca.antenna()
```
2.1.2 Return the antenna 1s in the calibration table.

Description

This member function returns the antenna 1s in the calibration table.

Arguments

<table>
<thead>
<tr>
<th>Inputs name</th>
<th>The python boolean which determines whether antenna 1 names (True) or antenna 1 numbers (False) are returned.</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>boolean</td>
</tr>
<tr>
<td>Default:</td>
<td>true</td>
</tr>
</tbody>
</table>

Returns

stringArray

Example

antenna1 = ca.antenna1()
calanalysis.antenna2.html

**calanalysis.antenna2 - Function**

2.1.2 Return the antenna 2s in the calibration table.

**Description**

This member function returns the antenna 2s in the calibration table.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The python boolean which determines whether antenna 2 names (True) or antenna 2 numbers (False) are returned.</td>
</tr>
<tr>
<td>allowed:</td>
<td>boolean</td>
</tr>
<tr>
<td>Default:</td>
<td>true</td>
</tr>
</tbody>
</table>

**Returns**

stringArray

**Example**

```python
antenna2 = ca.antenna2()
```
calanalysis.numfeed - Function

2.1.2 Return the number of feeds in the calibration table.

Description

This member function returns the number of feeds in the calibration table.

Arguments

Returns

int

Example

numfeed = ca.numfeed()
2.1.2 Return the feeds in the calibration table.

**Description**

This member function returns the feeds in the calibration table.

**Arguments**

**Returns**

stringArray

**Example**

```python
feed = ca.feed()
```
2.1.2 Return the number of times in the calibration table.

Description

This member function returns the number of times in the calibration table.

Arguments

Returns

int

Example

numtime = ca.numtime()
calanalysis.time.html

**calanalysis.time** - Function

2.1.2 Return the times (in MJD seconds) in the calibration table.

**Description**

This member function returns the times (in MJD seconds) in the calibration table.

**Arguments**

**Returns**

doubleArray

**Example**

time = ca.time()
calanalysis.numspw.html

**calanalysis.numspw - Function**

2.1.2 Return the number of spectral windows in the calibration table.

**Description**

This member function returns the number of spectral windows in the calibration table.

**Arguments**

**Returns**

int

**Example**

```
numspw = ca.numspw()
```
Return the spectral windows in the calibration table.

Description

This member function returns the spectral windows in the calibration table.

Arguments

**Inputs**

<table>
<thead>
<tr>
<th>name</th>
<th>The python boolean which determines whether spectral window names (True) or spectral window numbers (False) are returned.</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>boolean</td>
</tr>
<tr>
<td>Default:</td>
<td>true</td>
</tr>
</tbody>
</table>

**Returns**

stringArray

Example

```python
spw = ca.spw()
```

---

1255
2.1.2 Return the number of channels per spectral window in the calibration table.

**Description**

This member function returns the number of channels per spectral window in the calibration table.

**Arguments**

**Returns**

intColor

**Example**

```plaintext
numChannel = ca.numchannel()
```

```plaintext
1256
```
calanalysis.freq.html

**calanalysis.freq - Function**

2.1.2 Return the frequencies per spectral window in the calibration table.

**Description**

This member function returns the frequencies per spectral window in the calibration table.

**Arguments**

**Returns**

record

**Example**

```python
freq = ca.freq()
```
Return the calibration data.

Description

This member function returns the calibration data.

Arguments
### Inputs

**field**  
The python comma-delimited string or list of strings containing the field names or numbers. The default is "" (all fields).
- *allowed:* variant  
- *Default:* 

**antenna**  
The python comma-delimited string or list of strings containing the antenna 1s and antenna 2s. The default is "" (all antenna 1s and antenna 2s).
- *allowed:* variant  
- *Default:* 

**timerange**  
The python list of floats of length two containing the start and stop times (in MJD seconds). The default is [] (the minimum start time and the maximum stop time).
- *allowed:* variant  
- *Default:* 

**spw**  
The python comma-delimited string containing the spectral window names and numbers along with their channel numbers. The default is "" (all spectral windows and channels).
- *allowed:* variant  
- *Default:* 

**feed**  
The python comma-delimited string or list of strings containing the feeds. The default is "" (all feeds).
- *allowed:* variant  
- *Default:* 

**axis**  
The python string containing the user-specified iteration axis. The allowed values are ""TIME"" and ""FREQ"". The default is ""FREQ"".
- *allowed:* string  
- *Default:* TIME

**ap**  
The python string which determines whether complex gains are converted to amplitudes or phases. The allowed values are ""AMPLITUDE"" and ""PHASE"". The default is ""AMPLITUDE"". This parameter is ignored when the "gain" values in the calibration table are real.
- *allowed:* string  
- *Default:* AMPLITUDE

**norm**  
The python boolean which determines whether the amplitudes are normalized along each non-iteration axis. The default is False. This parameter is ignored when the "gain" values in the calibration table are real or ap=""PHASE"".
- *allowed:* boolean  
- *Default:* false

**unwrap**  
The python boolean which determines whether the phases are unwrapped along each non-iteration axis. The default is False. This parameter is ignored when the "gain" values in the calibration table are real or ap=""AMPLITUDE"".
- *allowed:* boolean  
- *Default:* false

**jumpmax**  
The python float which determines the maximum phase jump near +/- PI before unwrapping is performed. E.g., jumpmax = 0.1. The default is 0.0. It is ignored if the non-iteration axis is frequency:
- 1) if jumpmax == 0.0, use fringe fitting (only available when the non-iteration axis is time);
- 2) if jumpmax != 0.0, use simple unwrapping (same algorithm as used when the non-iteration axis is time or frequency).
- *allowed:* double  
- *Default:* 0.0
Returns
record

Example

# All data limited only by the spectral window and channel input
data = ca.get( spw="0:4~15,1,2:10~20" )
calanalysis.fit.html

**calanalysis.fit - Function**

2.1.2 Return the calibration data and fits along the non-iteration axis.

**Description**

This member function returns the calibration data and fits along the non-iteration axis.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>The python comma-delimited string or list of strings containing the field names or numbers. The default is &quot;&quot; (all fields). allowed: variant Default:</td>
</tr>
<tr>
<td>antenna</td>
<td>The python comma-delimited string or list of strings containing the antenna 1s and antenna 2s. The default is &quot;&quot; (all antenna 1s and antenna 2s). allowed: variant Default:</td>
</tr>
<tr>
<td>timerange</td>
<td>The python list of floats of length two containing the start and stop times (in MJD seconds). The default is [] (the minimum start time and the maximum stop time). allowed: variant Default:</td>
</tr>
<tr>
<td>spw</td>
<td>The python comma-delimited string containing the spectral window names and numbers along with their channel numbers. The default is &quot;&quot; (all spectral windows and channels). allowed: variant Default:</td>
</tr>
<tr>
<td>feed</td>
<td>The python comma-delimited string or list of strings containing the feeds. The default is &quot;&quot; (all feeds). allowed: variant Default:</td>
</tr>
<tr>
<td>axis</td>
<td>The python string containing the user-specified iteration axis. The allowed values are &quot;TIME&quot; and &quot;FREQ&quot;. The default is &quot;&quot; (&quot;FREQ&quot;). allowed: string Default: TIME</td>
</tr>
<tr>
<td>ap</td>
<td>The python string which determines whether complex gains are converted to amplitudes or phases. The allowed values are &quot;AMPLITUDE&quot; and &quot;PHASE&quot;. The default is &quot;&quot; (&quot;AMPLITUDE&quot;). This parameter is ignored when the &quot;gain&quot; values in the calibration table are real. allowed: string Default: AMPLITUDE</td>
</tr>
<tr>
<td>norm</td>
<td>The python boolean which determines whether the amplitudes are normalized along each non-iteration axis. The default is False. This parameter is ignored when the &quot;gain&quot; values in the calibration table are real or ap=&quot;PHASE&quot;. allowed: boolean Default: false</td>
</tr>
<tr>
<td>unwrap</td>
<td>The python boolean which determines whether the phases are unwrapped along each non-iteration axis. The default is False. This parameter is ignored when the &quot;gain&quot; values in the calibration table are real or ap=&quot;AMPLITUDE&quot;. allowed: boolean Default: false</td>
</tr>
<tr>
<td>jumpmax</td>
<td>The python float which determines the maximum phase jump near +/- PI before unwrapping is performed. E.g., jumpmax = 0.1. The default is 0.0. It is ignored if the jump at +/- PI is less than 0.1. allowed: double Default: 0.0</td>
</tr>
</tbody>
</table>
Returns
record

Example

```python
# All data limited only by the spectral window and channel input. The fit order
# is linear.
data_fit = ca.fit( spw="0:4~15,1,2:10~20", order="LINEAR" )
```

2.2 agentflagger - Module

Module for flagging of synthesis data
include agentflagger

Description The agentflagger module provides synthesis flagging capabilities within CASA. The primary purpose of this module is to flag data inside a MeasurementSet.
**2.2.1 agentflagger - Tool**

Tool for manual and automated flagging

Requires:

**Synopsis**

**Description**

The *agentflagger* tool performs manual as well as automatic synthesis flagging operations within casapy. The *agentflagger* tool can operate on one measurement set at a time.

**Open the Measurement Set or Calibration Table and Attach it to the Tool**

The first thing to do is to open the MS or calibration table and attach it to the agentflagger tool. Use the `af.open` method, which requires the MS name and optionally the time interval, over which to buffer data before running the algorithm. The time interval is set by default to 0.0, which means a ‘scan’ length. The ‘ntime’ parameter is important for the modes tfcrop, rflag and extend.

```python
af.open('uid_X002.ms')
```

**Select the Data**

Once the MS is open, the next step is to select the data. This step will use the MS selection tool to select the portion of the MS given by the parameters. There are two ways of selecting the data:

1) Create a Python dictionary which internally will be transformed into a record containing the selection parameters.

```python
# Select the whole MS.
af.selectdata()

Select a portion of the MS using a dictionary.
myrecord={}
myrecord['scan']='1~3'
myrecord['spw']='0:1~10'
af.selectdata(myrecord)
```

2) Parse the parameter names directly to the function.

```python
af.selectdata(myrecord)
```
af.selectdata(scan='1~3', spw='0:1~10')

Parse the Parameters for the Flagging Mode(s)

Each flagging mode is called an agent. The available agents are: manual, clip, quack, shadow, elevation, tcfcrop, rflag, extend, unflag and summary. Each one of these agents may or may not take configuration parameters and data selection parameters. Once the desired flagging modes are chosen, it is time to give the configuration parameters to the tool. Omitted parameters will take default values as defined in each agent. There are two ways of parsing the agent’s parameters.

1) Using the general method af.parseagentparameters().

   Construct a dictionary with the parameters for each agent. Each agent’s parameters should go to a different ‘key’ of the dictionary. Example:

   # Create a shadow agent:
   myagents = {}
   myagents['mode'] = 'shadow'
   af.parseagentparameters(myagents)

   # Add a summary agent to the list.
   myagents = {}
   myagents['mode'] = 'summary'
   myagents['spwchan'] = True
   af.parseagentparameters(myagents)

   # Add a manual agent to the same internal list of agents.
   myagents = {}
   myagents['mode'] = 'manual'
   myagents['scan'] = '1~3,18~20'
   af.parseagentparameters(myagents)

   # Add a clip agent to flag the zero-value data.
   myagents = {}
   myagents['mode'] = 'clip'
   myagents['clipzeros'] = True
   af.parseagentparameters(myagents)

   # Add another summary agent to the list.
   myagents = {}
   myagents['mode'] = 'summary'
   myagents['spwchan'] = True
   af.parseagentparameters(myagents)

2) The other way to parse agent’s parameters is to use the convenience functions. The above example would become:
# Create a shadow agent:
af.parseshadowparameters()

# Add a summary agent to the list.
af.parsesummaryparameters(spwchan=True)

# Add a manual agent to the same internal list of agents.
af.parsemanualparameters(scan='1~3,18~20')

# Add a clip agent to flag the zero-value data.
af.parseclippparameters(clipzeros=True)

# Add another summary agent to the list.
af.parsesummaryparameters(spwchan=True)

**Initialize the Agents**
The above step create a list of the agents that the tool will use to process the data. This step will check several parameters and apply constraints. It will set the iteration approach to COMBINE_SCAN, MAP, ANTENNA_PAIRS, ONLY if the agent is either tfcrop or extend and combinescans is set to True. Otherwise it will set it to COMPLETE_SCAN, MAP, ANTENNA_PAIRS, ONLY.

If the list contains agents that set ntime more than once, this method will get the maximum value of ntime and use it for all agents.

If a tfcrop agent is present, this method will create one agent per each polarization available, if correlation is set to ALL.

In the same way, if an agent tfcrop, rflag or clip is present, the asyncio mechanism will be switched on.

af.init()

**Run the tool**
Run the tool to apply or unapply the flags. The run method takes two parameters, writeflags and sequential. The parameter writeflags controls whether to write the flags or not to the MS. By default it is set to True. The sequential parameter tells to apply/unapply the flags in parallel or not. By default it is set to True, which means that the agents will run in sequential.

The run method gathers several reports, depending on which agents are run. The display and summary agents produce reports that can be retrieved from calling the run method. The reports are returned via a Python dictionary.

myreports = af.run(writeflags=True)

The dictionary returned in 'myreports' will contain four reports from the two summary agents that were added previously. The first report is the normal summary for each selection parameter. The second report gives the antenna positions for plotting.
Destroy the tool
Do not forget to destroy and close the tool at the end.

    af.done()

Methods

- `agentflagger` Construct a flag tool
- `done` Destroy the flag tool
- `open` Open the MS or a calibration table and attach it to the tool.
- `selectdata` Select the data based on the given parameters. For unspecified parameters, the full data range is assumed.
- `parseagentparameters` Parse the parameters for the agent (flagging mode).
- `init` Initialize the agents
- `run` Execute a list of flagging agents
- `getflagversionlist` Print out a list of saved flag versions.
- `printflagselection` Print out a list of current flag selections.
- `saveflagversion` Save current flags with a version name.
- `restoreflagversion` Restore flags from a saved flag version. versionname : name of flag version to restore to main table
- `merge = and` Logical AND with main table flags
- `merge = or` Logical OR with main table flags
- Default : replace.
- `deleteflagversion` Delete a saved flag version.
- `parsemanualparameters` Parse data selection parameters and specific parameters for the manual mode.
- `parseclipparameters` Parse data selection parameters and specific parameters for the clip mode.
- `parsequackparameters` Parse data selection parameters and specific parameters for the quack mode.
- `parseelevationparameters` Parse data selection parameters and specific parameters for the elevation mode.
- `parsetfcropparameters` Parse data selection parameters and specific parameters for the time and frequency mode.
- `parseextendparameters` Parse data selection parameters and specific parameters for the extend mode.
- `parsesummaryparameters` Parse data selection parameters and specific parameters for the summary mode.
agentflagger.agentflagger.html

agentflagger.agentflagger - Function

2.2.1 Construct a flag tool

Description

Create a agentflagger tool, and initialize some variables.

Arguments

| Inputs |

| Returns |
unknown

Example

af.agentflagger()
agentflagger.done - Function

2.2.1 Destroy the flag tool

Arguments

Returns
bool

Example

af.done()
agentflagger.open - Function

2.2.1 Open the MS or a calibration table and attach it to the tool.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>msname</td>
<td>Measurement set or calibration table to be processed.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
<td></td>
</tr>
<tr>
<td>ntime</td>
<td>Time interval. If not given, the default will be used.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: double</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: 0.0</td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool

Example

af.open(msname, ntime)
agentflagger.selectdata.html

**agentflagger.selectdata - Function**

2.2.1 Select the data based on the given parameters. For unspecified parameters, the full data range is assumed. All data selection parameters follow the MS Selection syntax.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>config</td>
<td>The record (dictionary) config may be given or not. If it is not given, and no specific parameter is given either, the whole MS will be selected. The record may contain any data selection parameters supported by MS Selection such as:</td>
</tr>
<tr>
<td>allowed</td>
<td>record</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>field</td>
<td>Field indices or source names : example : '2,3C48'</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>spw</td>
<td>Spectral Window Indices or names : example : '1,2'</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>array</td>
<td>Array Indices or names : example : 'VLAA'</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>feed</td>
<td>Feed index or name : example : '1,2' (not supported yet)</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>scan</td>
<td>Scan number : example : '1,2,3'</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>antenna</td>
<td>Baseline number : example : '2,3,4,5'</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>uvrange</td>
<td>UV-distance range, with a unit : example : '2.0-3000.0 m'</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>timerange</td>
<td>Time range, as MJDs or date strings : example : 'xx.x.x.x~yy.y.y.y.y'</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>correlation</td>
<td>Correlations/polarizations : example : 'RR,LL,RL,LR,XX,YY,XY,YX,Sol1'</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>intent</td>
<td>Scan intent : example : '<em>CAL</em>, <em>BAND</em>'</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>observation</td>
<td>Observation Id : example : '2~4'</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
</tbody>
</table>
Returns

bool

Example

Select the whole MS
af.selectdata()

Select a portion of the MS
myrecord=
myrecord['scan']='1~3'
myrecord['spw']='0:1~10'
af.selectdata(myrecord)

Another way to select a portion of the MS
af.selectdata(scan='3~5', spw='0')
agentflagger.parseagentparameters.html

agentflagger.parseagentparameters - Function

2.2.1 Parse the parameters for the agent (flagging mode).

Description

The specific data selection parameters for the agent (flagging mode) are parsed. These parameters are the data selection and mode-specific parameters. See the example below:

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aparams</td>
<td>It takes a record (dictionary) with the specific parameters for the flagging mode. The record may contain any data selection parameters supported by MS Selection, as well as mode-specific parameters. (1) array, feed, scan, field, spw, intent, correlation, antenna, uvranges, observation (2) mode (which can be: manual, clip, quack, shadow, elevation, tfcrop, extendflags, unflag or summary) For flagging mode=clip, the parameters are: expression, datacolumn, clipminmax, etc. See the documentation of the task flagdata for all the available parameters for each mode. (3) apply: default is true (true for flagging and false for unflagging). Example: myrecord=myrecord['mode']='clip' myrecord['scan']='1\sim3' myrecord['clipminmax'] allowed: record Default:</td>
</tr>
</tbody>
</table>

Returns

bool

Example

myrecord={}
myrecord['mode']='clip'
myrecord['scan']='1~3'
myrecord['clipminmax']=[0.02,0.3]
myrecord['apply']=True
af.parseagentparameters(myrecord)
**agentflagger.init.html**

**agentflagger.init - Function**

2.2.1 Initialize the agents

**Description**

This method will initialize the agents and create a list of agents with their specific parameters. It takes no parameters.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
</table>

**Returns**

bool

**Example**

af.init()
agentflagger.run - Function

2.2.1 Execute a list of flagging agents

Description

Execute a list of flagging agents and write or not to the MS/cal table. The parameter writeflags controls whether or not to write to the MS.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>writeflags</td>
<td>Write flags to MS</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>sequential</td>
<td>Run the agents in the order they are inserted in the list or not. Default is True to run in the original order.</td>
<td>bool</td>
<td>true</td>
</tr>
</tbody>
</table>

Returns

record

Example

af.run()
agentflagger.getflagversionlist.html

**agentflagger.getflagversionlist - Function**

2.2.1 Print out a list of saved flag versions.

**Description**

Print out the list of flag versions in the MS, unless the parameter `printflags=False`. The list of names is returned.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>printflags</td>
<td>Print flagversions in logger?</td>
</tr>
<tr>
<td>allowed</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>true</td>
</tr>
</tbody>
</table>

**Returns**

`stringArray`
agentflagger.printflagselection - Function

2.2.1 Print out a list of current flag selections.

Description
Print out a list of current flag selections.

Arguments

| Inputs |

Returns
bool
agentflagger.saveflagversion - Function

2.2.1 Save current flags with a version name.

Description

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>versionname</td>
<td>Version name</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td></td>
</tr>
<tr>
<td>comment</td>
<td>Comment for this flag table</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td></td>
</tr>
<tr>
<td>merge</td>
<td>merge type</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool
agentflagger.restoreflagversion.html

**agentflagger.restoreflagversion - Function**

2.2.1 Restore flags from a saved flag version.  versionname : name of flag version to restore to main table  merge : Type of operation to perform during restoration.  merge = replace : replaces the main table flags.  merge = and : logical AND with main table flags.  merge = or : logical OR with main table flags  Default : replace.

**Description**

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>versionname</td>
<td>Version name</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>merge</td>
<td>merge type</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool
agentflagger.deleteflagversion.html

agentflagger.deleteflagversion - Function

2.2.1 Delete a saved flag_version.

Description

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Version name</th>
<th>allowed:</th>
<th>stringArray</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>versionname</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool
agentflagger.parsemanualparameters.html

**agentflagger.parsemanualparameters - Function**

2.2.1 Parse data selection parameters and specific parameters for the manual mode. Data selection follows the MS Selection syntax.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed Types</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>Field indices or source names. Example: '2,3C48'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>spw</td>
<td>Spectral Window Indices or names. Example: '1,2'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>array</td>
<td>Array Indices or names. Example: 'VLAA'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>feed</td>
<td>Feed index or name. Example: '1,2' (not supported yet)</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>scan</td>
<td>Scan number. Example: '1,2,3'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>antenna</td>
<td>Baseline number. Example: '2,3,4,5,132'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>uvrange</td>
<td>UV-distance range, with a unit. Example: '2.0-3000.0 m'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>Time range, as MJDs or date strings. Example: 'xx.x.x.x~yy.y.y.y'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>correlation</td>
<td>Correlations/polarizations. Example: 'RR,LL,RL,LR,XX,YY,XY,YX'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>intent</td>
<td>Scan intent. Example: '<em>CAL</em>, <em>BAND</em>'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>observation</td>
<td>Observation Id. Example: '2~4'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>autocorr</td>
<td>Parameter to flag only auto-correlations.</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>apply</td>
<td>Parameter to flag or unflag the data.</td>
<td>bool</td>
<td>true</td>
</tr>
</tbody>
</table>
Returns
bool

Example

af.parsemanualparameters(autocorr=True)
2.2.1 Parse data selection parameters and specific parameters for the clip mode. Data selection follows the MS Selection syntax.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>Field indices or source names. Example: '2,3C48'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>spw</td>
<td>Spectral Window Indices or names. Example: '1,2'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>array</td>
<td>Array Indices or names. Example: 'VLAA'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>feed</td>
<td>Feed index or name. Example: '1,2' (not supported yet)</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>scan</td>
<td>Scan number. Example: '1,2,3'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>antenna</td>
<td>Baseline number. Example: '2,3,4,5'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>uvrage</td>
<td>UV-distance range, with a unit. Example: '2.0-3000.0 m'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>Time range, as MJDs or date strings. Example: 'xx.x.x.x~yy.y.y.y'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>correlation</td>
<td>Correlations/polarizations. Example: 'RR,LL,RL,LR,XX,YY,XY,YX'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>intent</td>
<td>Scan intent. Example: '<em>CAL</em>, <em>BAND</em>'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>observation</td>
<td>Observation Id. Example: '2~4'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>datacolumn</td>
<td>Data column to use for clipping. Supported columns for cal tables are FPARAM,CPARAM,SNR. Example: 'DATA'.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>clipminmax</td>
<td>Range to use for clipping. Example: [100.0,200.0]</td>
<td>doubleArray</td>
<td>DATA</td>
</tr>
<tr>
<td>clipoutside</td>
<td>Clip points outside this range? [True/False].</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>channelavg</td>
<td>Average data over channels before clipping? [True/False].</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>chanbin</td>
<td>Width (bin) of input channels to average to form an output channel.</td>
<td>variant</td>
<td></td>
</tr>
</tbody>
</table>
Returns
bool

Example

The correlation parameter can be used with an operator for the clip mode. The operator should be written only once and it will affect all the polarizations given in the string. See the example below:

```python
af.parseclipparameters(clipzeros=True, clipminmax=[0.,4.], correlation='ABS_XX,XY')
```

or for a calibration table:

```python
af.parseclipparameters(clipzeros=True, clipminmax=[0.,4.], correlation='Sol1')
```
agentflagger.parsequackparameters - Function

2.2.1 Parse data selection parameters and specific parameters for the quack mode. Data selection follows the MS Selection syntax.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Example</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>Field indices or source names. Example: '2,3C48'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>spw</td>
<td>Spectral Window Indices or names. Example: '1,2'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>array</td>
<td>Array Indices or names. Example: 'VLAA'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>feed</td>
<td>Feed index or name. Example: '1,2' (not supported yet)</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>scan</td>
<td>Scan number. Example: '1,2,3'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>antenna</td>
<td>Baseline number. Example: '2,3,4,5'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>uvrange</td>
<td>UV-distance range, with a unit. Example: '2.0-3000.0 m'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>Time range, as MJDs or date strings. Example: 'xx.x.x.x~yy.y.y.y'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>correlation</td>
<td>Correlations/polarizations. Example: 'RR,LL,RL,LR,XX,YY,XY,YX'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>intent</td>
<td>Scan intent. Example: '<em>CAL</em>, <em>BAND</em>'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>observation</td>
<td>Observation Id. Example: '2~4'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>quackmode</td>
<td>Quack mode. Default:</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>quackinterval</td>
<td>Quack length in seconds. Default:</td>
<td></td>
<td>double</td>
<td>0.0</td>
</tr>
<tr>
<td>quackincrement</td>
<td>Flag incrementally in time. Default:</td>
<td></td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>apply</td>
<td>Parameter to flag or unflag the data. Default:</td>
<td></td>
<td>bool</td>
<td>true</td>
</tr>
</tbody>
</table>
Returns
bool

Example

af.parsequackparameters(scan='1~3', quackmode='beg', quackinterval=1)
agentflagger.parseelevationparameters.html

**agentflagger.parseelevationparameters** - Function

2.2.1 Parse data selection parameters and specific parameters for the elevation mode. Data selection follows the MS Selection syntax.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Example</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>Field indices or source names. Example: '2,3C48'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>spw</td>
<td>Spectral Window Indices or names. Example: '1,2'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>array</td>
<td>Array Indices or names. Example: 'VLAA'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>feed</td>
<td>Feed index or name. Example: '1,2'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>scan</td>
<td>Scan number. Example: '1,2,3'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>antenna</td>
<td>Baseline number. Example: '2,3,4,5'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>uvrange</td>
<td>UV-distance range, with a unit. Example: '2.0-3000.0 m'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>Time range, as MJDs or date strings. Example: 'xx.x.x.x~yy.y.y.y'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>correlation</td>
<td>Correlations/polarizations. Example: 'RR,LL,RL,LR,XX,YY,XY,YX'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>intent</td>
<td>Scan intent. Example: '<em>CAL</em>, <em>BAND</em>'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>observation</td>
<td>Observation Id. Example: '2~4'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>lowerlimit</td>
<td>The limiting elevation in degrees. Data obtained at lower antenna elevations will get flagged. Default:</td>
<td></td>
<td>double</td>
<td>0.0</td>
</tr>
<tr>
<td>upperlimit</td>
<td>The limiting elevation in degrees. Data obtained at higher antenna elevations will get flagged. Default:</td>
<td></td>
<td>double</td>
<td>90.0</td>
</tr>
<tr>
<td>apply</td>
<td>Parameter to flag or unflag the data. Default:</td>
<td></td>
<td>bool</td>
<td>true</td>
</tr>
</tbody>
</table>
Returns

bool

Example

To unflag, set the apply parameter.
af.parseelevationparameters(upperlimit=50.,lowerlimit=10.0, apply=False)
2.2.1 Parse data selection parameters and specific parameters for the time and frequency mode. Data selection follows the MS Selection syntax.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Example</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>Field indices or source names. Example: ’2,3C48’</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>spw</td>
<td>Spectral Window Indices or names. Example: ’1,2'</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>array</td>
<td>Array Indices or names. Example: ’VLAA’</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>feed</td>
<td>Feed index or name. Example: ’1,2’ (not supported yet)</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>scan</td>
<td>Scan number. Example: ’1,2,3’</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>antenna</td>
<td>Baseline number. Example: ’2,3,4,5’</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>uvrange</td>
<td>UV-distance range, with a unit. Example: ’2.0-3000.0 m’</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>Time range, as MJDs or date strings. Example: ’xx.x.x.x.x<del>yy.y.y.y</del>’</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>correlation</td>
<td>Correlations/polarizations. Example: ’RR,LL,RL,LR,XX,YY,XY,YX’</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>intent</td>
<td>Scan intent. Example: ’<em>CAL</em>, <em>BAND</em>’</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>observation</td>
<td>Observation Id. Example: ’2~4’</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>ntime</td>
<td>Time-range to use for each chunk (in seconds or minutes). Default:</td>
<td></td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>combinescans</td>
<td>Accumulate data across scans depending on the value of ntime. Default:</td>
<td></td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>datacolumn</td>
<td>Data column to use for clipping. Example: ’DATA’. Default:</td>
<td></td>
<td>string</td>
<td>DATA</td>
</tr>
<tr>
<td>timecutoff</td>
<td>Flagging thresholds in units of deviation from the fit. Default:</td>
<td></td>
<td>double</td>
<td>4.0</td>
</tr>
<tr>
<td>freqcutoff</td>
<td>Flagging thresholds in units of deviation from the fit. Default:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Returns
bool

Example

The correlation parameter can be used with an operator for the tfcrop mode. The operator should be written only once and it will affect all the polarizations given in the string. Note that if ntime='scan' and combinescans=True, all the scans will be loaded at once, thus requesting a lot of memory depending on the available spws. The parameter combinescans should be set to True only when ntime is specified as a time-interval (not 'scan'). By default, the flags will be extended in time, if more than 50\% of the timeranges are flagged, 80\% of the channels are flagged and it will extend to other polarizations too. This is similar to running the extend mode after running tfcrop on the MS.

af.parsetfcropparameters(spw='9', ntime=10.0, combinescans=True, correlation='ABS_XX,XY', extendflags=True)
agentflagger.parseextendparameters.html

agentflagger.parseextendparameters - Function

2.2.1 Parse data selection parameters and specific parameters for the extend mode. Data selection follows the MS Selection syntax.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Example</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>Field indices or source names.</td>
<td>'2,3C48'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>spw</td>
<td>Spectral Window Indices or names.</td>
<td>'1,2'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>array</td>
<td>Array Indices or names.</td>
<td>'VLAA'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>feed</td>
<td>Feed index or name.</td>
<td>'1,2'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>scan</td>
<td>Scan number.</td>
<td>'1,2,3'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>antenna</td>
<td>Baseline number.</td>
<td>'2,3,4,5'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>uvrang e</td>
<td>UV-distance range, with a unit.</td>
<td>'2.0-3000.0 m'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>Time range, as MJDs or date strings.</td>
<td>'xx.x.x.x.x~yy.y.y.y'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>correl ation</td>
<td>Correlations/polarizations.</td>
<td>'RR,LL,RL,LR,XX,YY,XY,YX'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>intent</td>
<td>Scan intent.</td>
<td>'<em>CAL</em>, <em>BAND</em>’</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>observation</td>
<td>Observation Id.</td>
<td>'2~4'</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>ntime</td>
<td>Time-range to use for each chunk (in seconds or minutes).</td>
<td></td>
<td>double</td>
<td>0.0</td>
</tr>
<tr>
<td>combinescans</td>
<td>Accumulate data across scans.</td>
<td></td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>extendspols</td>
<td>If any correlation is flagged, flag all correlations.</td>
<td></td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>growtime</td>
<td>Flag all 'ntime' integrations if more than X% of the timerange is flagged (0-100).</td>
<td></td>
<td>double</td>
<td>50.0</td>
</tr>
<tr>
<td>growfreq</td>
<td>Flag all selected channels if more than X% of the frequency range is flagged(0-100).</td>
<td></td>
<td>double</td>
<td></td>
</tr>
</tbody>
</table>
Returns
bool

Example

af.parseextendparameters(extendpols=True)
agentflagger.parsesummaryparameters - Function

2.2.1 Parse data selection parameters and specific parameters for the summary mode. Data selection follows the MS Selection syntax.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>field</strong></td>
<td>Field indices or source names. Example: '2,3C48'</td>
<td>allowed: string</td>
</tr>
<tr>
<td><strong>spw</strong></td>
<td>Spectral Window Indices or names. Example: '1,2'</td>
<td>allowed: string</td>
</tr>
<tr>
<td><strong>array</strong></td>
<td>Array Indices or names. Example: 'VLAA'</td>
<td>allowed: string</td>
</tr>
<tr>
<td><strong>feed</strong></td>
<td>Feed index or name. Example: '1,2' (not supported yet)</td>
<td>allowed: string</td>
</tr>
<tr>
<td><strong>scan</strong></td>
<td>Scan number. Example: '1,2,3'</td>
<td>allowed: string</td>
</tr>
<tr>
<td><strong>antenna</strong></td>
<td>Baseline number. Example: '2,3,4,5'</td>
<td>allowed: string</td>
</tr>
<tr>
<td><strong>uvrange</strong></td>
<td>UV-distance range, with a unit. Example: '2.0-3000.0 m'</td>
<td>allowed: string</td>
</tr>
<tr>
<td><strong>time</strong></td>
<td>Time range, as MJDs or date strings. Example: 'xx.x.x.x~yy.y.y.y.y'</td>
<td>allowed: string</td>
</tr>
<tr>
<td><strong>correlation</strong></td>
<td>Correlations/polarizations. Example: 'RR,LL,RL,LR,XX,YY,XY,YX'</td>
<td>allowed: string</td>
</tr>
<tr>
<td><strong>intent</strong></td>
<td>Scan intent. Example: '<em>CAL</em>, <em>BAND</em>';</td>
<td>allowed: string</td>
</tr>
<tr>
<td><strong>observation</strong></td>
<td>Observation Id. Example: '2~4'</td>
<td>allowed: string</td>
</tr>
<tr>
<td><strong>spwchan</strong></td>
<td>List the number of flags per spw and per channel. Default:</td>
<td>allowed: bool</td>
</tr>
<tr>
<td><strong>spwcorr</strong></td>
<td>List the number of flags per spw and per correlation. Default:</td>
<td>allowed: bool</td>
</tr>
<tr>
<td><strong>basecnt</strong></td>
<td>List the number of flags per baseline. Default:</td>
<td>allowed: bool</td>
</tr>
<tr>
<td><strong>fieldcnt</strong></td>
<td>List the number of flags per field. Default:</td>
<td>allowed: bool</td>
</tr>
<tr>
<td><strong>name</strong></td>
<td>Name of this summary report. Default: summary</td>
<td>allowed: string</td>
</tr>
</tbody>
</table>

Default:
Returns
bool

Example

af.parsesummaryparameters(spwchan=True, basecnt=True)

2.3 imager - Module

Module for synthesis and single dish imaging

include imager.g

imager provides a unified interface for synthesis and single dish imaging including deconvolution starting from a MeasurementSet.

What imager does:

Standard synthesis and single dish imaging imager does nearly all types of synthesis and single dish imaging, including dirty images, point spread functions, deconvolution, combination of single dish and synthesis, spectral imaging, polarimetry, wide-field imaging, mosaicing, holography, near-field imaging, tracking moving objects, etc. As a result of this extensive range of capabilities, it can be complicated to use, especially for the more esoteric forms of imaging.

Fine scaled tools Rather than present one operation to process data from visibilities to a restored, deconvolved image, imager contains a number of distinct, separate tool functions that allow careful tuning of the processing. For example, the weights used in imaging (the IMAGING_WEIGHT column in the MeasurementSet), can be altered via a number of tool functions [weight]filter and inspected via a plotting tool function [plotweights]. Similarly, the deconvolution and restoration steps are separate, allowing user control of each step. It is our intention that other imaging tools may be built on top of imager: see, for example, imagerwizard which also has the side-benefit that it displays the imager (and other tools) commands as they are executed.
Spectral imaging imager can perform either spectral imaging or frequency synthesis (producing either an image with each channel imaged independently or with some or all channels summed together). Channels may be selected in a number of ways, either as channels or as velocities. Also a continuum model image may be subtracted prior to making a cube.

Many different deconvolution algorithms imager is rich in deconvolution algorithms, including a number of clean variants, maximum entropy, non-negative least squares, and the pixon algorithm.

Mixing of deconvolution functions Since the deconvolved images are calculated and kept purely as images (rather than lists of clean components), deconvolution functions may be mixed as desired. Thus, one may use NNLS to deconvolve part of the Stokes I of an image, and then use CLEAN to deconvolve another part of all polarizations in the image. Note that a list of clean components is not available.

Ability to fix model images In a multifield deconvolution, it is possible to specify that some fields are not to be deconvolved, using the fixed argument of clean.

Single dish imaging imager can process single dish observations much as it does synthesis images. To make images with no deconvolution, use the makeimage function. This allows construction of traditional single dish images and holography images. To deconvolve images, just use the “multifield” deconvolution algorithms in clean and mem. You will want to set the gridmachine in setoptions to ‘sd’.

Combination of single dish and synthesis data If the single dish and interferometer data are in the same MeasurementSet, then imager can perform a joint deconvolution using “multifield” deconvolution algorithms in clean and mem. You will want to set the gridmachine in setoptions to ‘both’. You can change the relative weighting of synthesis and single dish data by using setsdoptions. If the single dish and synthesis data cannot be combined into one MeasurementSet then you can still use the feather function to combine already deconvolved images.

Multi-field processing imager can be run on any number of images, each of which can have any direction for the phase center. All coordinate transformations are done correctly. Using the measures system, these fields may be given moving positions (such as the Sun using dm.direction(‘sun’) to specify the phase center) or positions in strange coordinates (such as Supergalactic using e.g. dm.direction(‘supergal’, ‘0d’, ‘0d’) as well as the more conventional representations (e.g. dm.direction(‘b1950’, ‘12h26m33.248000’, ‘02d19m43.290000’) specifies the coordinates of the core of 3C273). Note that for some coordinate systems a location must be supplied via the setoptions tool function. For example, one can put an image at a specific azimuth and elevation (e.g. dm.direction(‘azel’,

1304
Wide-field imaging imager can perform wide-field imaging as needed to overcome the non-coplanar baselines effect for the VLA and other non-coplanar arrays.

Mosaicing imager can perform clean-based or mem-based mosaicing of many pointings into one image, using variants of the multi-field algorithms.

Processing of component lists Discrete components (not the same as clean components!) can be represented by componentmodels. A componentlist can hold any number of components. The components are subtracted from the visibility data before construction of an image. For high precision imaging, it is recommended that components be used for bright sources since the Fourier Transform of components avoids the limitations of the gridded transforms.

Joint deconvolution of Stokes IQUV imager can produce images of either I alone, or I, V or I, Q, U, V, deconvolving jointly as appropriate. The point spread function is constrained to be the same for all processed polarizations so asymmetric u,v coverage is not allowed.

Production of complex images imager can produce dirty or residual images or point spread functions in the original data representation (e.g. RR, RL, RL, LL or XX, XY, YX, YY.

Fine control and evaluation of visibility weighting Various tool functions for controlling the visibility weights are available (weight, filter) as well as tool functions for evaluating the effects of the weighting (plotweights, sensitivity, fits). The Briggs algorithm for weighting of visibility data can be used (see weight and Dan Briggs’ thesis).

Flexible windowing in the deconvolution Rather than use boxes to limit the region CLEANed, a mask image is used to constrain the region in which flux is allowed. There are various tool functions for making a mask image, including from regions and blc/trc specifications, and via thresholding the Stokes I image. In the Clark Clean, the mask is soft: it can vary between 0 and 1. Intermediate values of the mask bias against but do not rule out subtraction of clean components.

Non-Negative Least Squares Deconvolution This algorithm is very effective at producing high dynamic range images of moderately resolved sources (see Dan Briggs’ thesis). It works on Stokes I alone so the recommended procedure is to CLEAN I, Q, U, V using clean and then use NNLS to refine the I part of the image using nnls.

Specification of arguments as A measure is a measured quantity with optional units, coordinates and reference frames. These are allowed in a
number of circumstances. The advantage is that the user can specify arguments in very convenient form, and let the measures system do whatever conversion is required. For example:

**Cell sizes**  These can be specified as a quantity (see the `measures` module).

```python
imgr.setimage(cellx='7arcsec', celly='7arcsec')
```

**Image center direction**  This must be specified as a direction (see the `measures` module).

```python
imgr.setimage(phasecenter=dm.direction('j2000', '05h30m', '-30.2deg'))
imgr.setimage(phasecenter=dm.direction('gal', '0deg', '0deg'))
imgr.setimage(phasecenter=dm.direction('mars'))
imgr.setimage(phasecenter=image('myother.image').coordmeasures().direction);
```

**Velocities**  These can be specified as radial velocities.

```python
imgr.setimage(start=dm.radialvelocity('25km/s'),
               step=dm.radialvelocity('-500m/s'))
```

**Position**  For construction of images in some coordinate frames (e.g. azimuth-elevation) the position to be used in processing must be set:

```python
imgr.setoptions(location=dm.observatory('ATCA'))
```

**More choice in image size**  Any even image size will work, though to speed the FFT, it is advisable to use a highly composite number (one that has many factors). The `advise` function will calculate an acceptable number.

**Integrated plotting**  Plots of visibility amplitude, weights (both point-by-point and gridded), uv coverage, and field and spectral window ids are available (plotvis, plotweights, plotuv, plotsummary).

**Synchronous or Asynchronous processing**  Operations that take a substantial amount of time to run can be run in the background either by setting the global variable `dowait := F` or by setting an argument `e.g. imgr.clean(async=T)`.

To retrieve a result, use the result tool function of defaultservers with the job number as the argument. For example:

```python
- imgr:=imager('ss433.MS')
  T
- imgr.setimage(cellx='0.05arcsec', celly='50marcsec', nx=256, ny=256,
                spwid=1:2, fieldid=1, stokes='IV')
  T
- imgr.fitpsf()
  T
# Wait for it to finish and then ask for the result:
- defaultservers.result(1)
  [psf=, bpa=[value=42.7269936, unit=deg], bmin=[value=0.13008301,
         unit=arcsec], bmaj=[value=0.159367442, unit=arcsec]]
A novel sort-less gridding algorithm The visibility data are not sorted before the gridding step. Instead, a cache of tiles is allocated to hold each baseline as it moves around in the Fourier plane. When a baseline moves off an existing tile, the results are written to disk and the necessary new tile is read in. Since the rotation of baselines in the uv plane is usually quite slow, the hit rate of such a cache is high. The size of the cache is by default set to half the physical memory of the machine, as specified by the aipsrc variable system.resources.memory. This can be overridden by the user, via the setoptions tool function. The cache can be made smaller at the expense of more paging of tiles in and out. The tile size can also be changed but this is seldom needed. This approach is optimal for arrays with small numbers of antennas but can be slow for e.g. the VLA. We intend to rectify this in the near future.

Plug-in commands imager can be customized by attaching commands using the CASA plug-in system. See the file code/trial/apps/imager/imager_standard.gp for an example of how to attach commands.

Suite of tests imager has a suite of tests. A standard test data set and component list can also be created.

imagerwizard The simplemage function is a wizard that performs interactively guided imaging of synthesis data.

dragon The dragon tool performs wide-field imaging using imager.

vpmanager The vpmanager tool manages specification of primary beams for imager.

Near-field imaging experimental Images of objects in the near-field of an array can be made. If the distance to the object is specified in setimage then the extra delay due to the wavefront curvature is corrected in the transforms. Note that some telescopes (e.g. VLA) make this correction in the real-time system. This effect is important if the distance to the object is comparable to or less than:

\[ \frac{B^2}{\lambda} \]  

where B is the baseline. Note that the sign of the correction could be in error in this experimental version: try using a negative distance as well as a positive distance.

What imager needs:

imager operates on a specified MeasurementSet to produce any of a range of different types of image: dirty, point spread function, clean, residual, etc. A MeasurementSet is the holder for measurements from a telescope. It is simply
an CASA Table obeying certain conventions as to required and optional contents. The intention is that it should contain all the information needed to reduce synthesis and single dish observations (see [CASA Note 191]). A UVFITS file can be converted to a MeasurementSet using the fitstoms tool function (a constructor of the ms tool).

`imager` adds some extra columns to the MeasurementSet to store results of processing. The following columns in the MS are particularly important:

**DATA** The original observed visibilities are in a column called DATA. These are not altered by any processing in CASA.

**CORRECTED_DATA** During a calibration process, as carried out by *e.g.* `calibrator`, the visibilities may be corrected for calibration effects. This corrected visibilities are stored in a column CORRECTED_DATA which is created on demand by `calibrator` and `imager`. In creating the CORRECTED_DATA column, `imager` will only correct for parallactic angle rotation. This can be controlled using the `correct` tool function. All imaging performed by `imager` is from the CORRECTED_DATA column (apart from the tool function `makeimage` which can also make dirty images from the other visibility columns).

**MODEL_DATA** During various phases of processing, the visibilities as predicted from some model are required. These model visibilities are stored in a column MODEL_DATA. These are used by the `calibrator` tool for calibration.

**IMAGING_WEIGHT** Weighting of data (including natural, uniform and Briggs weighting, and tapering) is accomplished by setting the column IMAGING_WEIGHT appropriately.

Standard tools such as the `table` module and the `ms` can be used to access and possibly change these (and all other) columns.

`imager` can handle an initial model in a number of forms: as an image, as a list of images, as a `componentmodels:componentlist` or as some combination. Fitting of `componentmodels` is planned but is not currently supported.

`imager` uses a number of scratch files. Following CASA practice, these are placed in the directories specified in the aipsrc variable user.directories.work. Those disks that possess sufficient free disk space are chosen in sequence. So to spread your scratch files over two disks each of which has a directory tcornwel/tmp do *e.g.*

```bash
user.directories.work: /bigdisk1/tcornwel/tmp /bigdisk2/tcornwel/tmp
```

1308
How to control imager:

To use imager, one has to construct a imager tool using a MeasurementSet as an argument, for example:

\[
\text{myimager}:=\text{imager('3C273XC1.ms')}
\]

The Glish variable myimager then contains the tool functions that may be used to do various operations on the MeasurementSet 3C273XC1.ms. These tool functions can be broken down into those that set imager up in some state, and those that actually do some processing. The setup tool functions are:

- **setimage** is a required tool function that defines the parameters (size, sampling, phase center, etc.) of any image that is to be constructed. If you omit to call setimage prior to any operation that needs these parameters, an error message will result. setimage is passive: nothing happens immediately but subsequent processing is altered.

- **setdata** is an optional tool function that selects which data are to be operated on during the processing. This selection can consist of choosing the spectral windows or fields that are to be operated on, or setting channels that are to be operated on in subsequent processing. setdata is active: the selection occurs immediately and is effective for all subsequent operations (until setdata is called again).

- **setoptions** is an optional tool function that sets parameters of lesser importance such as gridding parameters, cache sizes. While these affect the processing, usually the default values will suffice. setoptions is passive: nothing happens immediately but subsequent processing is altered.

- **setbeam** is an optional tool function that sets the parameters of the synthesized beam to be used in restoring deconvolved images. setbeam is passive: nothing happens immediately but subsequent processing is altered.

- **setvp** is an optional tool function that sets the parameters of the voltage pattern model used in mosaicing. setvp is passive: nothing happens immediately but subsequent processing is altered.

- **setsdoptions** is an optional tool function that sets the relative scaling and weighting of single dish data versus interferometer data and also other single dish specific parameters like the convolution support when doing single dish imaging.

Thus to understand what imager is doing, one has to remember that at any time, it has a state that has been set by using these tool functions. The state may be viewed in one of two ways: either summary can be used to output the current state to the logger, or, in the GUI, the current state of these parameters is displayed and updated following any relevant changes.

1309
All the other tool functions of **imager** are active: something happens immediately. Hence, for example, the **weight** tool function acts immediately to change the weighting of the selected data. In particular, unlike other packages, it does *not* set the weighting parameters for latter operations. The **clean** tool function performs a clean deconvolution of an image, reading and writing a model image. Note that operations that require or produce an image usually take an appropriate image name in the argument list. Often if such an image is not given then it is constructed using the image parameters set via **setimage** and using an appropriate name (e.g. a restored image is named from the model image by appending .restored so that 3C273XC1.clean becomes 3C273XC1.clean.restored).

The concept of the *state* of **imager** bears a little more explanation. The MeasurementSet can potentially contain data for many different fields and spectral windows. One therefore has to have some way of distinguishing which data are to be included in processing. Rather than have each possible tool function (e.g. weight, image, clean) take a long list of parameters to determine which data are to be included, **imager** has a **setdata** tool function that sets **imager** up in a way that in subsequent processing only the selected data are processed. For example, to select only field id 1 and spectral windows 1 and 2, one would do:

```python
myimager.setdata(fieldid=1, spwid=1:2)
```

The state of **imager** also consists of information about the default image settings (set via **setimage**) and various less important options (set via **setoptions**).

**What **imager** produces:**

**imager** reads and writes CASA MeasurementSets and Images. The format of images is 4 dimensional, with the first two being right ascension and declination, the third being polarization and the fourth being frequency. By suitable choice of the input parameters, one can make images of $I$ alone, $I, V$ or $I, Q, U, V$ for one or all channels. The **makeimage** tool function can also make a complex image of the original polarizations e.g. RR, RL, LR, LL. This latter type of image is useful for diagnostic purposes.

Images generated by **imager** may be viewed using **viewer** tool or retrieved using the **images** tool, the MeasurementSets may be accessed using the **ms** tool. More on this in the example below.

**What **imager** does not do:**

**imager** does not handle calibration of visibility data beyond correction for parallactic angle variations. Instead, you should use the **calibrater** tool for this purpose. However, **imager** and calibrater can cooperate on the self-calibration of data.
What improvement to imager are in the works:

We are currently working on a number of improvements:

- Improved griddser to handle many telescopes and many channels more efficiently.
- Parallelized CLEAN and gridding

Advanced use of imager:

As with all CASA applications, imager is designed to be open: all the results are written to and read from standard CASA table files. This open design of imager also allows the user to try out new methods of processing data. Models may be read into Glish, edited or manipulated via standard Glish facilities, and then written out and used subsequently in imager. Suppose that we want to halve the Stokes I of all pixels with negative Stokes I. The following Glish fragment does the trick:

```glish
m:=image('myimage')
shape:=im.shape()
blc:=[1,1,1,1]
trc:=[shape[1],shape[2],1,shape[4]]
a:=m.getchunk(blc,trc)
a[a<0.]*:=0.5
m.putchunk(a,blc)
m.flush()
m.close()
```

Overview of imager tool functions:

- Data access: open, close, done
- Data selection: setdata
- Data editing: clipvis
- Data calibration: correct
- Data examination: plotvis, plotuv, plotweights, plotsummary
- Weighting: weight, filter, uvrange, sensitivity, fitpsf, plotweights
- Image definition: advise, setimage, make
Example The following example shows the quickest way to make a CLEAN image and display it. Note that this can be more easily done from the toolmanager.

include 'imager.g'
#
# First make the MS from a FITS file:
#
m:=fitstoms(msfile='3C273XC1.MS', fitsfile='3C273XC1.FITS'); m.close();
#
# Now make an imager tool for the MS
#
imgr:=imager('3C273XC1.MS')
#
# Set the imager to produce images of cellsize 0.7 and
# 256 by 256 pixels
#
imgr.setimage(nx=256,ny=256, cellx='0.7arcsec',celly='0.7arcsec');
#
# Wait for results before proceeding to the next step
#
dowait:=T
#
# Make and display a clean image
#
imgr.clean(niter=1000, threshold='30mJy',
model='3C273XC1.clean.model', image='3C273XC1.clean.image')

dd.image('3C273XC1.clean.image')
#
# Fourier transform the model
#
imgr.ft(model='3C273XC1.clean.model')
#
# Plot the visibilities
#
imgr.plotvis()
#
# Write out the final MS and close the imager tool
#
imgr.close()
2.3.1 imager - Tool

tool for synthesis imaging

Requires:

Synopsis

Description

imager is an tool that accomplishes synthesis processing. A imager must be constructed for each MeasurementSet for which one wishes to do processing. Multiple copies of imager may be made at any time (provide they are given different names).

Methods

- **imager**: Construct an imager tool
- **advise**: Advise (and optionally use) parameter values
- **advisechansel**: Advise on spw and chan selection optimal for the image frequency range wanted
- **approximatepsf**: Calculate approximate point spread functions
- **boxmask**: Construct a mask image from blc, trc
- **calcuvw**: Calculates (u, v, w) coordinates for the ms.
- **clean**: Calculate a deconvolved image with selected clean algorithm
- **clipimage**: Zero all pixels where Stokes I is below a threshold
- **clipvis**: Flag visibilities where residual exceeds a threshold
- **close**: Close the imager tool, with data written on disk, keeping imager process running for further use
- **defineimage**: Set the image parameters for subsequent processing
- **done**: Terminate the imager process
- **drawmask**: Allows you do draw mask using the viewer
- **exprmask**: Construct a mask image from a LEL expression
- **feather**: Feather together an interferometer and a single dish image in the Fourier plane
- **filter**: Apply additional weighting by filtering (u-v taper)
- **fitpsf**: Fit the point spread function, making psf image first if needed
- **fixvis**: Performs visibility adjustments.
- **ft**: Fourier transform the specified model and componentlist
- **getweightgrid**: get the requested weight grids
- **linearmosaic**: Make a linear mosaic of several images
- **make**: Make an empty (i.e. blank) image
- **predictcomp**: Make a component list for a known object
- **makeimage**: Calculate images by gridding, etc.
makemodelfromsd - Make an initial model image from a Single Dish image
mask - Construct a mask image by thresholding an image
mem - Calculate a deconvolved image with selected mem (maximum entropy) algorithm
nnls - Calculate a deconvolved image using the NNLS algorithm
open - Open a new MeasurementSet, for processing, closing current MeasurementSet
pb - Applies or corrects for a primary beam
plotsummary - Plot a summary of field and spectral window ids
plotuv - Plot the uv coverage
plotvis - Plot the visibility amplitudes as a function of u-v radius (also, see visplot tool
plotweights - Plot the visibility weights as a function of u-v radius
regionmask - Construct a mask image from a region
regiontoimagemask - Union a mask image with various regions
residual - Calculate the residual image with respect to current model and component list
restore - Calculate the restored image with restored model, component list, and residuals
updateresidual - Calculate the residual and restored images with new modified model, component list, and residuals
sensitivity - Calculate rms sensitivity
apparentsens - Calculate rms sensitivity directly from weights
setbeam - Set the beam parameters for clean restoration
selectvis - Select visibilities for subsequent processing
setjy - Compute the model visibility for a specified source flux density
ssoflux - Use setjy instead.
setmfcontrol - Set various cycle control parameters for multi-field and wide-field imaging.
ssetoptions - Set some general options for subsequent processing
setscales - Set the scale sizes for MultiScale Clean
setscalebias - Set bias toward smaller scales for MultiScale Clean
settaylorterms - Set the number of Taylor series terms for Multi-Frequency Clean
setsdoptions - Set some options for single dish processing
setvp - Set the voltage pattern model for subsequent processing
setweightgrid - Set the requested weight grids
smooth - Calculate an image smoothed with a Gaussian beam
stop - Stop the currently executing function asap
summary - Summarize the current state of the imager tool
uvrange - Select data within the limit of a given range
weight - Apply additional weighting to the visibility weights
mapextent - Compute map extent from given set of MSs
**imager.imager - Function**

2.3.1 Construct an imager tool

**Description**

This is used to construct *imager* tools associated with a MeasurementSet. The *imager* tool may then be used to generate various types of images. Note that a new executable is started every time the constructor is called. This returns a Glish variable containing the tool functions of imager in an alternate universe that you have to tunnel to with a wormhole.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>MeasurementSet to be imaged</td>
</tr>
<tr>
<td>compress (allowed: bool, default: true)</td>
<td>Compress calibration columns?</td>
</tr>
<tr>
<td>host (allowed: string, default: F)</td>
<td>Host on which to run imager</td>
</tr>
<tr>
<td>forcenewserver (allowed: bool, default: true)</td>
<td>Flag to force a new imager client</td>
</tr>
</tbody>
</table>

**Returns**

`imager`

**Example**

```plaintext
im.open('3C273XC1.MS')
```
```python
im.defineimage(nx=256, ny=256, cellx='0.7arcsec', celly='0.7arcsec')
im.image(type='corrected', image='3C273XC1.dirty')
im.close()
```
imager.advise.html

imager.advise - Function

2.3.1 Advise (and optionally use) parameter values

Description

Advise on recommended values of certain parameters. Return these values and optionally use them in Imager.
The calculations are performed as following:

cell The maximum uv distance in wavelength is found and then half of the inverse is taken as the maximum cell size allowed.

pixels The field of view is converted to a number of pixels using the calculated cell size.

facets The number of facets on an axis is calculated in two different ways. The first method simply requires that the peeling of facets away from the celestial sphere should not cause an amplitude drop of more than the argument amplitudeloss. The positions may be incorrect, but all the sources will be removed correctly. The second method requires that the source positions be accurate to the same fraction of the beam specified by amplitudeloss. The second calculates the second moment in w and in uv distance and chooses the number of facets correspondingly. The first method does the same but after fitting a plane to the sampling: \( w = au + bv \). For an approximately coplanar array, the positions may be wrong but the removal of sidelobes will be accurate. The number of facets returned is the second, usually smaller, number. The formula used is:

\[
N_{\text{facets}} = N_{\text{pixels}} \sqrt{\frac{\Delta \theta}{\delta A}} \frac{w_{\text{rms}}}{uv_{\text{rms}}} \quad (2.2)
\]

where \( \Delta \theta \) is the cellsize in radians, and \( \delta A \) is the amplitude loss. This formula can be derived from (a) the peeling of facets from the celestial sphere, and (b) a quadratic approximation for the beam size both in the plane of the sky and along the \( w \) axis.

Arguments
### Outputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixels</td>
<td>Number of pixels on a side</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>cell</td>
<td>Recommended maximum cellsize</td>
<td>record</td>
<td></td>
</tr>
<tr>
<td>facets</td>
<td>Recommended number of facets on one axis</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>phasemeter</td>
<td>Direction of phase center as a measure</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

### Inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>takeadvice</td>
<td>Use the advised values?</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>amplitudeloss</td>
<td>Maximum fractional amplitude loss due to faceting</td>
<td>double</td>
<td>0.05</td>
</tr>
<tr>
<td>fieldofview</td>
<td>Desired field of view</td>
<td>any</td>
<td>variant 1.0deg</td>
</tr>
</tbody>
</table>

### Returns

bool
imager.advisechansel.html

imager.advisechansel - Function

2.3.1 Advise on spw and chan selection optimal for the image frequency range wanted

Description

Basically tells you what channels of which spectral window need to be selected for your image spectral parameters. The freqstep is used to calculate the extra padding needed for data selection at the beginning and end of the range. The freqframe parameter is the frame in which the frequency range is being given. It will be converted to the frame of the data with time to locate which channel match. A record will be returned with an element for each ms used in selectvis. Each element of the record will have the spwids and channel start and nchan for each spwid. if the parameter msname is used then the MSs associated associated with this tool (that have been either 'open'ed or 'selectvis'ed) are ignored In this mode it is a helper function to the general world ...no need to open or selectvis. You need to specify the field_id for which this calculation is being done for in the helper mode. If you have already set MS's and selected data and msname="" then the calculation is done for the field(s) selected in selectvis.

If the parameter getfreqrange=True then the reverse is requested. You set spwselection to be the range of data selection you want to use and you'll get the range of frequency covered in the frame you set.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>freqstart</td>
<td>Beginning of frequency range in Hz</td>
<td>double</td>
<td>1.0e6</td>
</tr>
<tr>
<td>freqend</td>
<td>End of frequency range in Hz</td>
<td>double</td>
<td>1.1e6</td>
</tr>
<tr>
<td>freqstep</td>
<td>Spectral channel resolution of intended image in Hz</td>
<td>double</td>
<td>100.0</td>
</tr>
<tr>
<td>freqframe</td>
<td>Frame in which frequency is being expressed in other parameters</td>
<td>string</td>
<td>LSRK</td>
</tr>
<tr>
<td>msname</td>
<td>Name of an ms, if empty string it will use the ms’s used in selectvis</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>fieldid</td>
<td>Fieldid to use when msname is not empty otherwise ignored and field selected in selectvis is used</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>getfreqrange</td>
<td>If set then freqrange is returned in the frame requested for the data selected</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>spwselection</td>
<td>If getfreqrange=True then this is needed to find the range of frequency in the frame requested</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

| Returns           | record                                                                     |

| Example           | In this example, we are interested in an image cube which span 20.0682GHz to 20.1982 in LSRK |

```python
im.selectvis(vis='test1.ms', field='4', spw='*')
```
im.selectvis(vis='test2.ms', field='4', spw='*')

selinfo=im.advisechansel(freqstart=2.00682e10, freqend=2.01982e10, freqstep=3.9e3, freqframe="LSRK")

###The output "selinfo" will be a record which will look like thus

{'ms_0': {'nchan': array([109, 23], dtype=int32),
'spw': array([4, 5], dtype=int32),
'start': array([19, 0], dtype=int32)
},

'ms_1': {'nchan': array([109, 23], dtype=int32),
'spw': array([4, 5], dtype=int32),
'start': array([19, 0], dtype=int32)}}

###Thus from the first ms a spw selection like '4:19~127, 5:0~22' is all that is needed. Similarly from the second ms.

###If you need this info without needing to change the state of the imager tool then you can it as follows

im.advisechansel(freqstart=2.00682e10, freqend=2.01982e10, freqstep=3.9e3, freqframe="LSRK")

###Now if you want to see what frequency range is covered, in the frame defined by freqframe, in spwselection you want to use

im.selectvis(vis='test3.ms', spw='0:20~210')
im.advisechansel(getfreqrange=True, freqframe="LSRK")

###the output will be something

{'freqend': 346020345384.64178, 'freqstart': 345683852920.1723}

###and if you just want to use it as a helper function without touching the state of imager

im.advisechansel(msname='test3.ms', getfreqrange=True, spwselection='0:20~210')
**imager.approximatepsf - Function**

**2.3.1 Calculate approximate point spread functions**

**Description**

Calculate the approximate point spread function. *Note that the model visibilities are updated.*

Some types of imaging do not yield a well-defined point spread function. For example, mosaicing or single dish imaging both yield point spread functions that are position dependent. Nevertheless, one can still usefully define an approximate PSF that is of some utility. This is calculated by doing the following calculation: a point source is located at the center of the specified coordinate system and the model data predicted. The approximate PSF is then formed from those model data using the full sky equation. For regular sampling in the image plane, this approximate PSF is actually quite good. It can be used in a deconvolution. For a mosaic with similar uv sampling per pointing, the approximate PSF is roughly the PSF per pointing multiplied by the primary beam. For a single dish image, it is roughly the telescope primary beam convolved with itself (if the gridfunction='pb' was selected).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psf</td>
<td>Name of output point spread function</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

1323
Example of how to make the approximate psf for a mosaic:

```python
im.open('orion.ms')
im.selectvis(spwid=[0, 1], field=range(2, 11));
im.defineimage(nx=300, ny=300, cellx='2.0arcsec', celly='2.0arcsec', stokes="I", phasecenter=6, spwid=[0, 1])
im.weight('natural')
im.setvp(dovp=T, usedefaultvp=True)
im.setoptions(ftmachine='mosaic', padding=1.0)
im.approximatepsf(psf='LePSF.image')
```
**imager.boxmask - Function**

2.3.1 Construct a mask image from blc, trc

**Description**

A mask image is an image with the same shape as the other images but with values between 0.0 and 1.0 as a pixel value. Mask images are used in imager to control the region selected in a deconvolution. In the Clark CLEAN, the mask image can usefully have any value between 0.0 and 1.0. Intermediate value discourage but do not rule out selection of clean components in that region. This is accomplished by multiplying the residual image by the mask prior to entering the minor cycle. Note that if you do use a mask for the Clark or Hogbom Clean, it must cover only a quarter of the image. boxmask does not enforce this requirement.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>name of mask image</td>
<td>allowed: string</td>
</tr>
<tr>
<td>blc</td>
<td>Bottom left corner</td>
<td>allowed: intArray</td>
</tr>
<tr>
<td>trc</td>
<td>Top right corner, should be image shape</td>
<td>allowed: intArray</td>
</tr>
<tr>
<td>value</td>
<td>Value to fill in</td>
<td>allowed: double</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

1325
im.boxmask(mask='bigmask', blc=[56,45,1,1], trc=[87,93,4,1])
im.clean(mask='bigmask', model='3C273XC1.clean.masked', niter=1000)

Makes the image bigmask, and then sets it to unity for all points in the region bounded by the blc and trc. Then cleans using it as the mask.
2.3.1 Calculates (u, v, w) coordinates for the ms.

Description

This calculates (u, v, w) positions for the visibilities using the antenna and feed positions and offsets, the time, and the phase tracking center(s).

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fields</td>
<td>Field IDs (numbered relative to 0) to operate on. Blank = all.</td>
</tr>
<tr>
<td>allowed:</td>
<td>intArray</td>
</tr>
<tr>
<td>Default:</td>
<td>-1</td>
</tr>
<tr>
<td>refcode</td>
<td>Reference frame to use for the generated (u, v, w)s.</td>
</tr>
<tr>
<td>WARNING: clean</td>
<td>and the im tool ignore the reference frame claimed by the UVW column (it</td>
</tr>
<tr>
<td>the im tool</td>
<td>is often mislabelled as ITRF when it is really J2000) and instead assume</td>
</tr>
<tr>
<td>ignore</td>
<td>the (u, v, w)s are in the same frame as the phase tracking center.</td>
</tr>
<tr>
<td>the reference</td>
<td>calcuvw does not yet force the UVW column and field centers to use the</td>
</tr>
<tr>
<td>frame</td>
<td>same reference frame!</td>
</tr>
<tr>
<td>Blank:</td>
<td>use the phase tracking frame of vis.</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>string</td>
</tr>
<tr>
<td>reuse</td>
<td>Start from the UVWs in vis (True) or calculate them from the antenna</td>
</tr>
<tr>
<td>allowed:</td>
<td>positions?</td>
</tr>
<tr>
<td>Default:</td>
<td>true</td>
</tr>
</tbody>
</table>

Returns

bool

Example

1327
im.open("3C273XC1.MS")
im.calcuvw()
im.done()
imager.clean.html

**imager.clean - Function**

2.3.1 Calculate a deconvolved image with selected clean algorithm

**Description**

Makes a clean image using either the Hogbom, Clark, multi-scale or multi-field algorithms. The Clark algorithm is the default. The clean is performed on the residual image calculated from the visibility data currently selected. Hence the first step performed in clean is to transform the current model or models (optionally including a componentlist) to fill in the `MODEL_DATA` column, and then inverse transform the residual visibilities to get a residual image. This residual image is then cleaned using the corresponding point spread function. This means that the initial model is used as the starting point for the deconvolution. Thus if you want to restart a clean, simply set the model to the model that was previously produced by clean.

Rather than explicit CLEAN boxes, mask images are used to constrain the region that is to be deconvolved. To make mask images, use either boxmask (to define a mask via the corner locations blc and trc) or mask (to define a mask via thresholding an existing image) or regionmask (to make masks via regions using the regionmanager or interactively through the viewer). The default mask is the inner quarter of the image.

The CLEAN deconvolution is joint in whatever Stokes parameters are present. Thus it searches for peaks in either $I$ or $I + |V|$ or $I + \sqrt{Q^2 + U^2 + V^2}$, the rationale for the latter two forms being to be biased towards finding strongly polarized pixels first (these forms are also the maximum eigenvalue of the coherency matrix). The PSF is constrained to be the same in all polarizations (a feature of this implementation, not of the Hamaker-Bregman-Sault formalism). But the option of searching peaks in the stokes planes independently is available via the `clarkstokes` parameter.

The clean algorithms possible are:

- **Hogbom** The classic algorithm: points are found iteratively by searching for the peak. Each point is subtracted from the full residual image using the shifted and scaled point spread function.

- **Multiscale** An experimental multi-scale clean algorithm is invoked. The algorithm is fully described in deconvolver.

- **Clark** The faster algorithm: the cleaning is split into minor and major cycles. In the minor cycles only the brightest points are cleaned, using a subset of the point spread function. In the major cycle, the points thus found are subtracted correctly by using an FFT-based convolution.
Multi-field Cleaning is split into minor and major cycles. For each field, a Clark-style minor cycle is performed. In the major cycle, the points thus found are subtracted either from the original visibilities (for multiple fields) or using a convolution (for only one field). The latter is much faster. Multi-field imaging has been implemented for Clark, Hogbom, and Multi-scale deconvolution algorithms.

Cotton-Schwab Cleaning is split into minor and major cycles. For each field, a Clark-style minor cycle is performed. In the major cycle, the points thus found are subtracted from the original visibilities. A fast variant does a convolution using a FFT. This will be faster for large numbers of visibilities. Double the image size from that used for Cotton-Schwab and set a mask to clean only the inner quarter.

Wide-field The user will need to use a wide-field algorithm to deconvolve if the array is not coplanar over the field of view being imaged. The technique used is to break the field being imaged into smaller pieces (facets), over each of which the array appear planar. We implement a rectangular facetting scheme. If the number of facets specified in defineimage is greater than one, Either wfhogbom or wfclark algorithm has to be selected here to perform a wide-field decovolution. The function advise can be used to calculate or check if you need to use a wide-field deconvolution. Note that aliasing can be reduced by using the padding argument in setoptions. In practice the previous sentence means that if you notice the clean to diverge at the edges of the facets then you need to use a larger amount of padding for the FT; the default being 1.2. Wide-field imaging has been implemented for Clark and Hogbom algorithms.

The multi-field clean should be used if either of two conditions hold:

1. Multiple fields are to be cleaned simultaneously OR
2. Primary beam correction is enabled. In this case, a mosaiced clean is performed.

Note that for the single pointing algorithms, only a quarter of the image may be cleaned. If no mask is set, then the cleaned region defaults to the inner quarter. If a mask larger than a quarter of the image is set, then only the inner quarter part of that mask is used. However, for the wide-field and multi-field imaging (including the Cotton-Schwab algorithm), the entire field may be imaged because the major cycles either do an exact subtraction from the visibilities or because PSF extent is more than twice the extent of the primary beam support.

Before clean can be run, you must run selectvis and defineimage. Before clean can be run with a multi-field algorithm (especially for mosaic), you should run setvp. You may want to run setmfcontrol before running clean with a multi-field or wide-field algorithm, though the default control values
may be acceptable. Before clean can be run with a multi-scale algorithm, setscales must be run.

Interactive cleaning/masking: If the user wants to see what the clean image looks like after npercycle iteration and mask or modify the mask each time, he/she should set interactive=True and give npercycle to a fraction of niter. A viewer with the last residual image along with an overlayed mask appear after every npercycle iteration. The user can add or delete regions (by clicking on the appropriate button) to the mask using the region button and drawing regions and double clicking inside the region. When satisfied and ready to continue cleaning press 'DONE with masking' (if the user want to terminate the cleaning process use the 'STOP' button). The button 'No more mask changes' should be used if the user want clean to proceed without any further interruption. Even if interactive=False, and if the parameter 'mask' is non-empty, it is still used in limiting the search area for clean components. If the parameter 'masktemplate' is not empty this means that the user want to use an apriori image to make the mask the first time (e.g a previously cleaned image).

This function returns a record containing convergence, iterations used and threshold reached.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Algorithm to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm</td>
<td>allowed: string</td>
</tr>
<tr>
<td>Default:</td>
<td>clark</td>
</tr>
<tr>
<td></td>
<td>clarkstokes</td>
</tr>
<tr>
<td></td>
<td>hogbom</td>
</tr>
<tr>
<td></td>
<td>multiscale</td>
</tr>
<tr>
<td></td>
<td>mfclark</td>
</tr>
<tr>
<td></td>
<td>mfclarkstokes</td>
</tr>
<tr>
<td></td>
<td>csclen</td>
</tr>
<tr>
<td></td>
<td>csfast</td>
</tr>
<tr>
<td></td>
<td>mfhgobom</td>
</tr>
<tr>
<td></td>
<td>mfmultiscale</td>
</tr>
<tr>
<td></td>
<td>wfclean</td>
</tr>
<tr>
<td></td>
<td>wfhgobom</td>
</tr>
<tr>
<td></td>
<td>clark</td>
</tr>
<tr>
<td>niter</td>
<td>Number of Iterations, set to zero for no CLEANing</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>1000</td>
</tr>
<tr>
<td>gain</td>
<td>Loop Gain for CLEANing</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>0.1</td>
</tr>
<tr>
<td>threshold</td>
<td>Flux level at which to stop CLEANing</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant 0.0Jy</td>
</tr>
<tr>
<td>displayprogress</td>
<td>Display the progress of the cleaning?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>model</td>
<td>Names of clean model images</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>keepfixed</td>
<td>Keep one or more models fixed</td>
</tr>
<tr>
<td>allowed:</td>
<td>boolArray</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>complist</td>
<td>Name of component list</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>mask</td>
<td>Names of mask images used for CLEANing</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>image</td>
<td>Names of restored images</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>residual</td>
<td>Names of residual images</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>psfimage</td>
<td>Names of psfs if they are needed</td>
</tr>
<tr>
<td>allowed:</td>
<td>1332 stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>interactive</td>
<td>whether to stop clean and interactively mask</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>npercycle</td>
<td>If interactive is 'T', then no of iter of clean before stopping, usually a fraction of niter</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>100</td>
</tr>
</tbody>
</table>
Returns
record

Example

\begin{verbatim}
im.clean(model='3C273XC1.clean.model', mask='3C283XC1.mask', niter=1000, gain=0.25, threshold='0.03Jy')

A few points should be noted in this example:
\begin{itemize}
  \item When the mask parameter is specified, the number of mask images listed should be equal to
        the number of model images. They should also have the same coordinate system as their
        corresponding model images.
  \item If one or more model images are listed in the model parameter but the image and residual
        parameters are empty, the restored and residual images are automatically named as the model
        names appended with '.restored' and '.residual', respectively.
  \item No restored or residual image is made if the respective image string is explicitly unset.
\end{itemize}
\end{verbatim}

include 'imager.g';
msfile = 'vlac125K.ms';
im.open(msfile);
npix = 500; cell='5arcsec';
#
# CS on 500 by 500
#
im.defineimage(nx=npix, ny=npix, cellx=cell, celly=cell, stokes='I', spw=[0,1]);
im.setoptions(padding=1.0);
im.selectvis(spwid=[0,1]);
im.clean('cs', model='vlac125K.cs', image='vlac125K.cs.restored',
        niter=1000, gain=0.1);
#
# CSF on 1000 by 1000, cleaning a given box
#
im.defineimage(nx=2*npix, ny=2*npix, cellx=cell, celly=cell, stokes='I',
spwid=[1,2]);
reg=rg.box(blc=[400,500], trc=[450,550])
im.regionmask('vlac125K.mask', region=reg);
im.clean('csf', model='vlac125K.csf', image='vlac125K.csf.restored',
mask='vlac125K.mask', niter=1000, gain=0.1);
#
# CS on 1000 by 1000, cleaning entire image
#
im.defineimage(nx=2*npix, ny=2*npix, cellx=cell, celly=cell, stokes='I',
spwid=[1,2]);
im.clean('cs', model='vlac125K.cs', image='vlac125K.cs.restored',
mask='vlac125K.mask', niter=1000, gain=0.1);
im.done();
imager.clipimage.html

**imager.clipimage - Function**

2.3.1 Zero all pixels where Stokes I is below a threshold

**Description**

All pixels in the image with Stokes I less than some threshold are set to zero. This is useful prior to self-calibration where one oftens wishes to remove negative pixels from the model. Note that if the image has polarization information, then the polarized part of a pixel is also set to zero if Stokes I is less than the threshold.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>name of image</td>
</tr>
<tr>
<td>threshold</td>
<td>Threshold</td>
</tr>
</tbody>
</table>

- **image**
  - allowed: string
  - Default:
- **threshold**
  - allowed: any
  - Default: variant 0.0Jy

**Returns**

bool

**Example**

im.clipimage(image='clean', threshold='50mJy')
imager.clipvis.html

**imager.clipvis - Function**

2.3.1 Flag visibilities where residual exceeds a threshold

**Description**

All visibilities where the residual exceeds some threshold are flagged. This provides a simple way of flagging bad data.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>threshold</td>
<td>Threshold</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
im.plotvis('residual')
# determine threshold then apply it
im.clipvis(threshold='50mJy')
```
imager.close.html

**imager.close - Function**

2.3.1 Close the imager tool, with data written on disk, keeping imager process running for future use

**Description**

This is used to close imager tools. Note that the data is written to disk. The imager process keeps running until a done tool function call is performed.

**Arguments**

**Returns**

bool

**Example**

```python
im.open('3C273XC1.MS')
im.makeimage(image='3C273XC1.dirty', type='corrected')
im.close()
```
**imager.defineimage - Function**

Set the image parameters for subsequent processing

**Description**

Define the default image parameters. If an image is to be made, then these parameters are used in the construction of the image. Thus, for example, the tool function make makes an (empty) image using these parameters. Note that some parameters can be specified either in canonical units or via measures. To establish default values, the ids for the default spectral window and default field id must be given.

The parameter **mode** can be one of the following:

- mfs
- channel
- velocity or opticalvelocity
- frequency

**imager** can perform multi-frequency synthesis over several spectral windows (mode='mfs'). To achieve this, you should set spwid to an array of the required spectral windows (e.g. `spwid=[0,1]`).

WARNING: For multifrequency synthesis, 'mfs', it is important that the spwid's selected in selectvis be the SAME as the one selected in **defineimage**. Otherwise the frequency at which the image is made is not going to be the same as to the one as the one used in gridding the visibility and can lead to image artifacts. For **mode='velocity'** and **mode='frequency'** the **step** parameter has to be a measure/quantity of velocity or frequency, otherwise for **mode='channel'** step is the number of data channels to be averaged to make one image channel (see examples below).

The phase center of the image defaults to that of the specified phascenter (the first fieldid in the ms is taken if none is specified), this parameter can be a fieldid or a measure string or the record output from the direction function of the measures tool( `direction` ). This is important if you have multiple pointings in the data. The user would have used selectvis to select which pointings would be used in imaging. If the conversion from the observed direction requires frame information then this is taken as follows:

- Direction information, including the coordinate system, is taken from the relevant entry in the Field table of the MeasurementSet.
• The epoch is taken from the time of observation of each visibility.
• A position is specified via the imager tool function setoptions

If the specified number of facets is greater than unity then the image is split into facets (this number along the x and y axes) and processed. This is necessary when using wide-field algorithm for deconvolving the image, in cases of non-coplanar arrays (e.g. the VLA at low frequencies but can be safely left at 1 for the ATCA or WSRT). This is now recommended only when memory or image size is of a problem, otherwise for widefield issues, wpagination (ftmachine parameter in setoptions) is recommended with a single facet. For spectral imaging defineimage and selectvis defines the spectral channels that are imaged. Examples are given in the selectvis section. The parameter restfreq can be used to define what rest frequency to use in the resulting images. If none is specified imager will try to use the one that is defined in the ms. It will use the first one defined in the first spectral window selected.

For wide-field or 3D imaging see setoptions section for some examples.

If the telescope is observing moving source (e.g. planet or moon) over a period of time. One may wish to image in a frame where the source is fixed. The parameter movingsource is for that. Setting it to a source that measures is aware of will force the imaging to realign (shift in SD imaging or phase rotation in interferometry imaging) the data so that the source appears fixed in the image. Obviously in doing so the background sources will be blurred. The coordinate system used to fix the source on is the one where the source is at the first time observed in the selected data.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>nx</td>
<td>Total number of spatial pixels in x</td>
<td>allowed: int</td>
</tr>
<tr>
<td>ny</td>
<td>Total number of spatial pixels in y</td>
<td>allowed: int</td>
</tr>
<tr>
<td>cellx</td>
<td>Cellsize in x (e.g. '1arcsec')</td>
<td>allowed: any</td>
</tr>
<tr>
<td>celly</td>
<td>Cellsize in y (e.g. '1arcsec')</td>
<td>allowed: any</td>
</tr>
<tr>
<td>stokes</td>
<td>Stokes parameters to image (e.g. 'IQUV')</td>
<td>allowed: string</td>
</tr>
<tr>
<td>phasecenter</td>
<td>Direction of phase center as a direction measure or a field id</td>
<td>allowed: any</td>
</tr>
<tr>
<td>mode</td>
<td>Type of processing (velocity = radiovelocity)</td>
<td>allowed: string</td>
</tr>
<tr>
<td>nchan</td>
<td>Number of channels; a -1 (default) means all the channels as selected in selectvis and combined into one continuum channel</td>
<td>allowed: int</td>
</tr>
<tr>
<td>start</td>
<td>Start channel; A 0-relative channel number of the spwid or a frequency quantity or a velocity quantity or radial velocity measure</td>
<td>allowed: any</td>
</tr>
<tr>
<td>step</td>
<td>Step in channel; integer for number of channels or frequency quantity or velocity quantity or radial velocity measure</td>
<td>allowed: any</td>
</tr>
<tr>
<td>spw</td>
<td>Spectral Window Id (0 relative) that defines center of image</td>
<td>allowed: intArray</td>
</tr>
<tr>
<td>restfreq</td>
<td>rest frequency to use; default = use the one available in ms</td>
<td>allowed: any</td>
</tr>
<tr>
<td>outframe</td>
<td>frequency frame of output image (default LSRK, &quot;&quot; =&gt; same as input ms or LSRK in case of multiple ms’s), options are LSRK, LSRD, BARY, GALACTO, LGROUP, CMB</td>
<td>allowed: string</td>
</tr>
</tbody>
</table>

### Inputs

**nx**
- Total number of spatial pixels in x
- allowed: int
- Default: 128

**ny**
- Total number of spatial pixels in y
- allowed: int
- Default: -1

**cellx**
- Cellsize in x (e.g. '1arcsec')
- allowed: any
- Default: variant 1.0

**celly**
- Cellsize in y (e.g. '1arcsec')
- allowed: any
- Default: variant

**stokes**
- Stokes parameters to image (e.g. 'IQUV')
- allowed: string
- Default: IV
- IQU
- IQUV
- I

**phasecenter**
- Direction of phase center as a direction measure or a field id
- allowed: any
- Default: variant 0

**mode**
- Type of processing (velocity = radiovelocity)
- allowed: string
- Default: frequency
- radiovelocity
- opticalvelocity
- truevelocity
- mfs

**nchan**
- Number of channels; a -1 (default) means all the channels as selected in selectvis and combined into one continuum channel
- allowed: int
- Default: -1

**start**
- Start channel; A 0-relative channel number of the spwid or a frequency quantity or a velocity quantity or radial velocity measure
- allowed: any
- Default: variant 0

**step**
- Step in channel; integer for number of channels or frequency quantity or velocity quantity or radial velocity measure
- allowed: any
- Default: variant 1

**spw**
- Spectral Window Id (0 relative) that defines center of image
- allowed: intArray
- Default: 0

**restfreq**
- rest frequency to use; default = use the one available in ms
- allowed: any
- Default: variant

**outframe**
- frequency frame of output image (default LSRK, "" => same as input ms or LSRK in case of multiple ms’s), options are LSRK, LSRD, BARY, GALACTO, LGROUP, CMB
- allowed: string
- Default: LSRK

### Options

- relativistic
- radio

---

**Note:**
- The table above outlines the inputs for a specific data processing task, detailing each parameter's name, description, default value, and allowed types.
- Each parameter's specification is crucial for configuring the processing accurately, ensuring that the output aligns with the desired outcome.
Returns
bool

Example

## Example 1
im.defineimage(nx=1024, ny=1024, cellx='30marcsec', celly='30marcsec',
nchan=1, stokes='IV', phasecenter=me.direction('mars'));
## Example 2
im.defineimage(nx=1024, ny=1024, cellx='30marcsec', celly='30marcsec',
nchan=1, stokes='IV', phasecenter=['J2000', '19:00:30.5', '-45d00m25.6']);
## Example 3
im.selectvis(nchan=10, start=3, spw=[0, 1], field=[3, 4, 5, 6, 7, 9, 10])
im.defineimage(nx=500, ny=500, mode='mfs', spwid=[0, 1], fieldid=7)
im.clean(algorithm='mfclark', niter=1000, model='mosaic.model', image='mosaic.image')
## Example 4

dir1=me.direction('J2000', '20h00m00', '21d00m00')
dir2=me.direction('J2000', '20h10m00', '21d00m00')
dir3=me.direction('J2000', '20h00m00', '21d03m00')
im.defineimage(nx=100, cellx='0.1arcsec', phasecenter=dir1)
im.make('box1')
im.defineimage(nx=100, cellx='0.1arcsec', phasecenter=dir2)
im.make('box2')
im.defineimage(nx=100, cellx='0.1arcsec', celly='0.1arsec', phasecenter=dir3)
im.make('box3')
im.clean(algorithm='mfclark', model=['box1', 'box2', 'box3'],
    image=['box1.restored', 'box2.restored', 'box3.restored'],
    residual=['box1.residual', 'box2.residual', 'box3.residual'])

In the first example, the image parameters are set for 1024 by 1024 pixels of 30milli arcsec, 1 channel will be made, Stokes I and V will be imaged, and the phase center will be the direction of Mars as given by the JPL DE-200 ephemeris. In the second, the phase center is taken to be an absolute coordinate value.

The third example shows the use of selectvis and defineimage to setup a mosaic. In the set data we have chosen 10 channels (for each spectral window) of data starting form
channel 3. We also have selected spectral windows 0 and 1. We have selected data from fields 3 to 10. In the defineimage we decide to use the data to make a multifrequency synthesis image. We center the image on the field 7 pointing.

The fourth example is use to clean regions where the user knows the sources are and ignore all the other regions. This is very efficient in large fields with few sources. Smaller outlier images are made and deconvolved around known sources rather than making a big image englobing all three fields.

Now here are some examples about defining cubes using different {\tt mode} parameters.

defining channels cubes use the channel as defined in the data

\begin{verbatim}
im.defineimage(cellx=1000, mode='channel', nchan=100, start=10, step=1, spwid=range(0,10))
\end{verbatim}

now using frequency and overriding the rest frequency defined in the ms or if its not defined in the ms

\begin{verbatim}
im.defineimage(cellx=1000, mode='frequency', nchan=100, start='1GHz', step='10kHz', restfrequency='1.421GHz')
\end{verbatim}

in case you have a frame with the frequency

\begin{verbatim}
im.defineimage(cellx=1000, mode='frequency', nchan=100, start=[LSRK', '1GHz'], step='10kHz', restfrequency='1.421GHz')
\end{verbatim}

OR using measures

\begin{verbatim}
freqstart=me.frequency(LSRK', '1GHz')
im.defineimage(cellx=1000, mode='frequency', nchan=100, start=freqstart, step='10kHz', restfrequency='1.421GHz')
\end{verbatim}

similarly if you want to use velocity to define your cube

\begin{verbatim}
im.defineimage(cellx=1000, mode='velocity', nchan=100, start=[LSRK', '10km/s'], step='1m/s', restfrequency='1.421GHz')
\end{verbatim}

OR using measures
velstart=me.radialvelocity('LSRK', '10km/s')
im.defineimage(cellx=1000, mode='velocity', nchan=100, start=velstart, step='1m/s')

Change mode to 'opticalvelocity' if your velocity values are using optical definition
**imager.done** - Function

2.3.1 Terminate the imager process

**Description**

This is used to totally stop the **imager** process. It is a good idea to conserve memory use on your machine by stopping the process once you no longer need it.

**Arguments**

**Returns**

bool

**Example**

```plaintext
im.open('3C273XC1.MS')
im.makeimage(image='3C273XC1.dirty', type='corrected')
im.done()
```
imager.drawmask.html

**imager.drawmask - Function**

2.3.1 Allows you to draw a mask using the viewer.

**Description**

A mask image is an image with the same shape as the other images but with values between 0.0 and 1.0 as a pixel value. Mask images are used in imager to control the region selected in a deconvolution. drawmask is used to interactively draw regions over a template image which you want to allow deconvolution to occur.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>image</strong></td>
<td>name of template image</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><strong>mask</strong></td>
<td>name of image to save mask in</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><strong>niter</strong></td>
<td>Total number of iteration to display in box; just for display or python packing</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td><strong>npercycle</strong></td>
<td>npercycle value to display in box; just for display or python packing</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td><strong>threshold</strong></td>
<td>threshold to display in box; just for display or python packaging</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>0 mJy</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

1345
im.drawmask(image='mytemplate.image', mask='myregions.mask')
im.clean(mask='myregions.mask', model='3C273XC1.clean.masked', niter=1000)

Make mask image by drawing interactively over a given image 'mytemplate.image', then image '3C273XC1.clean.masked' using the 'myregions.mask' as the "clean regions".
imager.exprmask.html

**imager.exprmask - Function**

[2.3.1] Construct a mask image from a LEL expression

**Description**

A mask image is an image with the same shape as the other images but with values between 0.0 and 1.0 as a pixel value. Mask images are used in imager to control the region selected in a deconvolution.

In the Clark CLEAN, the mask image can usefully have any value between 0.0 and 1.0. Intermediate value discourage but do not rule out selection of clean components in that region. This is accomplished by multiplying the residual image by the mask prior to entering the minor cycle. Note that if you do use a mask for the Clark or Hogbom Clean, it must cover only a quarter of the image. boxmask does not enforce this requirement.

This function allows Lattice Express Language (LEL) expressions to be used in defining a mask. See the documentation on imagecalc for more details.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>name of mask image</td>
<td>string</td>
</tr>
<tr>
<td>allowed:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>expr</td>
<td>Value to set the mask to. Any scalar or LEL expression</td>
<td>double</td>
</tr>
<tr>
<td>allowed:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
<td>1.0</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
im.exprmask(mask='bigmask', expr='3C273XC1.clean>0.5')
im.clean(mask='bigmask', model='3C273XC1.clean.masked', niter=1000)
```
Makes the image bigmask, and then sets it to unity for all points in the region where 3C273XC1.clean is greater than 0.5Jy. Then cleans using it as the mask.
imager.feather.html

imager.feather - Function

2.3.1 Feather together an interferometer and a single dish image in the Fourier plane

Description

Basically the "imerg" algorithm of AIPS and SDE, or the "feather" algorithm of MIRIAD, we regrid the total power (or low resolution) image onto the interferometer (or high resolution) image, Fourier transform both the interferometer and single dish images, down weight the Fourier transform of the interferometer image by 1.0 - FT(low res psf), add the weighted interferometer Fourier plane to the single dish Fourier plane, and transform back into the image plane.

The tapering is by the transform of a point spread function. If lowpsf is specified, that image is used, otherwise the appropriate telescope beam is used.

The point spread function for a single dish image may be calculated using makeimage.

Advice: Note that if you are feathering large images, you’d be advised to have the number of pixels along the X and Y axes to be composite numbers and definitely not prime numbers. In general FFTs work much faster on even and composite numbers. You may use subimage function of image tool to trim the number of pixels to something desirable.

Arguments
Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>Name of output feathered image</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>highres</td>
<td>Name of high resolution (interferometer) image</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>lowres</td>
<td>Name of low resolution (single dish) image</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>lowpsf</td>
<td>Name of optional low resolution point spread function</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>effdishdiam</td>
<td>Optional new SD dish diameter in m to use in feathering; can be smaller than true dish size</td>
<td>double</td>
<td>-1.0</td>
</tr>
<tr>
<td>lowpassfiltersd</td>
<td>Reject the high spatial frequency of the SD image</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

Returns

bool

Example

```python
im.setvp(dovp=True, usedefaultvp=True)
im.feather(image='feathered.image', highres='casa.vlaonly', lowres='casa.sd');
```

In the above example its using the default beams and the observatory information is in the image header.

But if you have a single dish image with a beam which is not defined in the casa database then the example below is a guide of how to do that, say you know the beam of the single dish as a gaussian.
# create a beam pattern table using vpmanager
include 'vpmanager.g'
vpman=vpmanager();
vpman.setpbgauss(telescope='OTHER', othertelescope='BONN',
halfwidth='1arcmin', maxrad='20arcmin', reffreq='1.4GHz');
vpmn.saveastable('bonn.pb')
vpmn.done()

## would have done your usual imager setup (defineimage etc) then before feathering
im.setvp(dovp=True, usedefaultvp=false, vptable='bonn.pb')
im.feather(image='feathered.image', highres='casa.vlaonly',
lowres='casa.sd');

###
imager.filter.html

**imager.filter - Function**

2.3.1 Apply additional weighting by filtering (u-v taper)

**Description**

Apply visibility tapering to emphasize certain scale structures. The imaging tapers are applied to a Table column called IMAGING_WEIGHT, which may be plotted using tb and pl.plotweights. In addition, this column may be accessed directly using either the table or ms modules. Note that the taper is multiplicative and so the weights must be calculated first using weight. The points are not flagged!

Note that the scale size to be emphasized is given in the image plane as the parameters of the corresponding Gaussian. Note also use of this function provides an optimum detection for the given scale size, which is not the same as requiring that the resulting dirty beam have the specified Gaussian fit. The resultant fitted beam size will *very roughly* be the quadratic sum of the original beam and the specified beam. If you wish to obtain a specified beam, then the best approach is to perform this calculation and check the value obtained using imager.fitpsf.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>type</strong></td>
<td>Type of filtering or u-v tapering</td>
<td>string</td>
</tr>
<tr>
<td><strong>bmaj</strong></td>
<td>Major axis of filter</td>
<td>any</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>variant 1arcsec</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td>gaussian</td>
</tr>
<tr>
<td><strong>bmin</strong></td>
<td>Minor axis of filter</td>
<td>any</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>variant 1arcsec</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td>0deg</td>
</tr>
<tr>
<td><strong>bpa</strong></td>
<td>Position angle of filter</td>
<td>any</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>variant 0deg</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td>boolean</td>
</tr>
<tr>
<td><strong>async</strong></td>
<td>Run asynchronously in the background</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>false</td>
</tr>
</tbody>
</table>

1352
Returns
bool

Example

im.weight('uniform')
im.filter(type='gaussian', bmaj='2.3arcsec', bmin='1.67arcsec', bpa='34.5deg')
imager.fitpsf.html

**imager.fitpsf - Function**

2.3.1 Fit the point spread function, making psf image first if needed

**Description**

This fits an elliptical Gaussian to the point spread function and returns the fitted beam parameters. If psf image is not specified then a psf is made and used. The values for the beam fit are saved internally and used whenever needed (for example in the functions restore or smooth) until invalidated. The values are invalidated by selectvis, defineimage or any tool function that changes the weights. Use the function summary to check if there is a valid fitted psf stored internally.

**Arguments**

<table>
<thead>
<tr>
<th>Outputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bmaj</td>
<td>Major axis of beam</td>
<td>allowed: record</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bmin</td>
<td>Minor axis of beam</td>
<td>allowed: record</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bpa</td>
<td>Position angle of beam</td>
<td>allowed: record</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>psf</td>
<td>Name of input psf</td>
<td>allowed: string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background</td>
<td>allowed: bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

1354
im.makeimage(type='psf', image='3C273XC1.psf')
params=im.fitpsf('3C273XC1.psf')
#This returns a python dict params here
print params['bmaj'].value, params['bmin'].value, params['bpa']
im.restore(model='bla', complist='', image='bla.restored', residual='bla2.residual')

Or if one wants to generate a psf from the uv coverage and use that subsequently as in the

- im.fitpsf(psf='')
- im.restore(model='bla', complist='', image='bla.restored', residual='bla2.residual')
imager.fixvis.html

**imager.fixvis - Function**

2.3.1 Performs visibility adjustments.

**Description**

Corrects UVW coordinates and optionally the visibilities for various effects that can be calculated without fitting a model to the data. The effects include:

- changing the phase tracking center(s),
- correcting for differential aberration, (Not yet implemented)
- changing the equinox (i.e. B1950_VLA to J2000 or APP, etc.) of the UVW coordinates,
- changing the projection, as in (-)NCP to SIN. (Not yet implemented),
- refocusing.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fields</td>
<td>Field IDs (numbered relative to 0) to operate on. Blank = all.</td>
</tr>
<tr>
<td>allowed:</td>
<td>intArray</td>
</tr>
<tr>
<td>Default:</td>
<td>-1</td>
</tr>
<tr>
<td>phasedirs</td>
<td>Phase tracking centers for each field in fields, in the same order.</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>refcode</td>
<td>Reference frame to use for the generated UVWs.</td>
</tr>
<tr>
<td>WARNING: clean</td>
<td>clean and the im tool ignore the reference frame claimed by the UVW column</td>
</tr>
<tr>
<td>(it is often</td>
<td>(it is often mislabelled as ITRF when it is really J2000) and instead</td>
</tr>
<tr>
<td>mislabelled</td>
<td>assume the (u, v, w) are in the same frame as the phase tracking center.</td>
</tr>
<tr>
<td>the UVW column</td>
<td>calcuvw does not yet force the UVW column and field centers to use the same</td>
</tr>
<tr>
<td>and field</td>
<td>reference frame!</td>
</tr>
<tr>
<td>center</td>
<td>Blank = use the phase tracking frame of vis.</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>distances</td>
<td>A list of distances (in m) for the fields listed in fields. 0 = infinity.</td>
</tr>
<tr>
<td>allowed:</td>
<td>doubleArray</td>
</tr>
<tr>
<td>Default:</td>
<td>0.0</td>
</tr>
<tr>
<td>datacolumn</td>
<td>Which of DATA, MODEL_DATA, and/or CORRECTED_DATA to operate on. Default:</td>
</tr>
<tr>
<td>allowed:</td>
<td>&quot;all&quot;.</td>
</tr>
<tr>
<td>Default:</td>
<td>all</td>
</tr>
</tbody>
</table>

| Returns        | bool                                                                        |

| Example        |                                                                             |
|----------------|                                                                             |
|                | im.open("3C273XC1.MS")                                                     |
|                | im.fixvis()                                                                |
|                | im.done()                                                                   |
imager.ft.html

**imager.ft - Function**

2.3.1 Fourier transform the specified model and componentlist

**Description**

Fourier transform the specified model (and optionally componentlist) and insert into the MODEL_DATA column. The current contents of the MODEL_DATA column are replaced unless incremental is set to T (in which case the results are added to the column).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>Name of image</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>complist</td>
<td>Name of component list</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>incremental</td>
<td>Add to the existing MODEL_DATA column?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
im.ft(model='3C273XC1.nnls.model')
im.ft(model='3C273XC1.another.model', incremental=True)
```
Fourier transforms the model in the image 3C273XC1.nnls.model and then adds the visibility due to 3C273XC1.another.model
**imager.getweightgrid.html**

**imager.getweightgrid - Function**

2.3.1 get the requested weight grids

**Description**

This is a utility function when running multi imager processes in parallel on subsection of an ms/data independently. One would wish to weight the dirty image before averaging or set the imaging weight density (when using uniform or Brigg's style weighting) to account for all the data being used. This is NOT for the general user but for people who are parallelizing at the scripting level. 

**imaging:** will return a the weight griddensity 

**ftweight:** will put the FT-machine weight images in the names given in wgtimage parameters..these may be needed to average residual images from different processes running seperately on different section of the data.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Type of weight requested (imaging, ftweight)</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: imaging</td>
</tr>
<tr>
<td>wgtimages</td>
<td>names of weightimages to save</td>
</tr>
<tr>
<td></td>
<td>allowed: stringArray</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

anyvariant

**Example**

```wght=im.getweightgrid('imaging')
wght2=im2.getweightgrid('imaging')
wght=wght+wght2```

1360
im.setweightgrid(weight=wght, type='imaging')
imager.linearmosaic.html

**imager.linearmosaic - Function**

2.3.1 Make a linear mosaic of several images

**Description**

Make a linear mosaic of several images. Currently, the pointing center is not specified in the image, so we specify the pointing center in terms of the row numbers of the FIELD subtable.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>images</td>
<td>Input images to be mosaiced</td>
<td>stringArray</td>
<td></td>
</tr>
<tr>
<td>mosaic</td>
<td>Output mosaic image</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>fluxscale</td>
<td>Fluxscale image</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>sensitivity</td>
<td>Sensitivity image</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>fieldids</td>
<td>List of field ids that correspond each of the images, used to center the PB of each image. (0-based list)</td>
<td>intArray</td>
<td></td>
</tr>
<tr>
<td>usedefaultvp</td>
<td>Use the default vp type?</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>vptable</td>
<td>Voltage pattern table from the vpmanager for detailed specification</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>
Returns

bool

Example

```python
im.linearmosaic(images=['orion.1.cln', 'orion.2.cln', 'orion.4.cln'], mosaic='orion.linmos', fluxscale='orion.linmos.fluxscale', fieldid=[1,2,4]);
```
imager.make.html

**imager.make - Function**

2.3.1 Make an empty (i.e. blank) image

**Description**

Make an empty image using the current image parameters. Often this is unnecessary, but you will typically need to use this if you wish to deconvolve a set of images. The steps are to make the empty images that you require to be deconvolved, and then pass them into clean as a vector of strings.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>name of output image</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```plaintext
im.defineimage(nx=1024, ny=1024, cellx='30marcsec', celly='30marcsec',
nchan=1, stokes='IV', phasecenter=me.direction('mars'));
im.make('mars.moving');
im.defineimage(nx=1024, ny=1024, cellx='30marcsec', celly='30marcsec',
nchan=1, stokes='IV', phasecenter=me.direction('J2000', '12:23:48.7', '-15:56:32.9'))
im.make('mars.fixed');
im.clean(algorithm='mf', model=['mars.moving', 'mars.fixed'],
image=['mars.moving.restored', 'mars.fixed.restored'])
```

1364
This makes two empty images, one moving with Mars and one fixed in J2000, and then deconvolves the two jointly using clean. Finally the images are restored.
**imager.predictcomp - Function**

2.3.1 Make a component list for a known object

**Description**

Make a component list for an object recognized by standard, one of setjy’s flux density standards.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>objname</td>
<td>Name of the object</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>standard</td>
<td>Name of the flux standard</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>epoch</td>
<td>Time to use, as an epoch measure, e.g. me.epoch('UTC', '55555d'), for Solar System objects</td>
<td>any</td>
<td>variant 55555.0d</td>
</tr>
<tr>
<td>freqs</td>
<td>The frequencies to use, in Hz</td>
<td>doubleArray</td>
<td>1.0e11</td>
</tr>
<tr>
<td>pfx</td>
<td>Prefix for the name of the component list</td>
<td>string</td>
<td>predictcomp</td>
</tr>
</tbody>
</table>

**Returns**

string

**Example**

1366
cline = im.predictcomp('Ceres', 'Butler-JPL-Horizons 2010',
    '2012-02-14/13:33:00', [3.45e11, 6.90e11], 'vd_')

This writes a component "list" named vd_spw0_Ceres_345GHz55971.6d.cl to disk 
containing a uniform disk component for Ceres as it is expected to appear at 
345 and 690 GHz at 2012-02-14/13:33:00 UTC, and returns the name of the 
component list. Returns '' on error.
imager.makeimage.html

**imager.makeimage - Function**

2.3.1 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:

- **observed** Make the dirty image from the DATA column *(default)*
- **model** Make the dirty image from the MODEL_DATA column
- **corrected** Make the dirty image from the CORRECTED_DATA column
- **residual** Make the dirty image from the difference of the CORRECTED_DATA and MODEL_DATA columns
- **psf** Make the point spread function
- **singledish** Make a single dish image
- **coverage** Make a single dish coverage image
- **holography** Make a complex holography image
- **pb** Make the primary beam as defined by setvp

Note the full `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 `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 uvw coordinates are reprojected prior to gridding, and a phase rotation is applied. If the image is a PSF then no phase shift is applied but the uvw are recomputed. To see the effects of the uvw reprojected, you can use the plotuv function.

If desired, the full complex image (before conversion to stokes I,Q,U,V) may be retained. Note that the image tool cannot load a complex image directly. Instead, use the imagecalc constructor to take *e.g.* the real and imaginary parts of the image.

For making single dish and holography images, the data are convolved onto the grid using a one of a number of options:
**gridfunction='SF'** Circularly symmetric prolate spheroidal wavefunction. This is always the same function in pixels. To get this to match to the antenna primary beam, the optimum cellsize to use in constructing the image is the antenna primary beam half-width-half-maximum times 1.20192.

**gridfunction='BOX'** Nearest neighbor gridding.

**gridfunction='PB'** The telescope primary beam is used as the convolution function. This function is the same in arcseconds, independent of the cellsize. This choice is optimum in the least squares sense. To override the default choice of telescope primary beam for a given telescope, use the function setvp. Usually the default will be acceptable.

To make a reasonable approximation to the sky, one should divide the type='singledish' image by the type='coverage' image, thresholding at some level. For example:

```python
ia.open('scanweight');
ia.statistics(s);
threshold = s.max / 10.0;
#
ia.imagecalc('sdimage',
    pixels=spaste('scanimage[scanweight>', threshold, ']/scanweight[scanweight>', threshold, ']'))
###ia.view(raster=True, axislabels=True);
```

### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Type of output image</td>
</tr>
<tr>
<td>image</td>
<td>Name of output image</td>
</tr>
<tr>
<td>compleximage</td>
<td>Name of output complex image</td>
</tr>
<tr>
<td>verbose</td>
<td>Report things like the center frequency to the logger</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background</td>
</tr>
</tbody>
</table>
Returns

bool

Example

im.ft(model='3C273XC1.model', complist='3C273XC1.complist');
im.makeimage(type='residual', image='3C273XC1.residual')
im.makeimage(type='psf', image='3C273XC1.psf')

Fill in the MODEL\_DATA column from Fourier transforming the model and the componentlist. Make the residual image and write it to 3C273XC1.residual.

Example

im.setvp(dovp=T, usedefaultvp=T, telescope='GBT');
im.makeimage(type='pb', image='gbt.pb')

In the above we may want to see what the primary beam we are using look like. May also be useful to deconvolve single dish images in the deconvolver tool.
2.3.1 Make an initial model image from a Single Dish image

Description

This function uses an image from a single dish and makes a model (clean component) image out of it. This allows one to use this as the starting model in a deconvolution function, e.g., clean or mem. This provides an alternative to feather. The difference between the two is that in feather, the interferometer image is deconvolved first and the single dish image is put in at the end. Whereas if one starts with a model from the single dish image, it will give a different starting point for the deconvolving algorithm to interpolate the missing short baseline.

The function `setsoptions` may be used to set a factor by which to scale the SD image, if necessary.

The `sdpsf` parameter (optional) should be used if an external PSF image of the single dish is needed to calculate the beam parameters of the primary beam of the dish. This is usually needed if the dish image is from a non-standard telescope or the beam is not in the CASA system.

The `mask` is a mask image that may be needed to be used for clean. This is usually the case when the dish image does not fully cover the field defined by `defineimage`.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sdimage</code></td>
<td>Single Dish image</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><code>modelimage</code></td>
<td>Name of output image to be used as model</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><code>sdpsf</code></td>
<td>PSF of Single Dish if needed</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><code>maskimage</code></td>
<td>mask image</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>
Returns
bool

Example

```python
im.open('orion_only.ms')
im.selectvis(field=range(10), spw=range(2))
im.defineimage(nx=1000, cellx='1arcsec', phasecenter=4, spwid=[0,1])
im.setvp(dovp=T)
im.setoptions(ftmachine='mosaic')
im.setscales(nscales=3)
im.setsdoptions(scale=0.9);
im.makemodelfromsd(sdimage='orion_gbt.im', modelimage='orion_model', maskimage='orion.mask')
im.clean(algorithm='mfmultiscale', model='orion_model', residual='orion.residual', image='orion.restored', gain=0.2, niter=500, mask='orion.mask')
```

In the above example we are making a mosaic with the fields 0 to 9. A single dish image `orion_gbt.im` is used scaled down by a factor 0.9 to make the initial model that is passed to multi-scale clean.
imager.mask.html

**imager.mask - Function**

2.3.1 Construct a mask image by thresholding an image

**Description**

A mask image is an image with the same shape as the other images but with values between 0.0 and 1.0 as a pixel value. Mask images are used in *imager* to control the region selected in a deconvolution. One makes a mask image by clipping the I part of the restored image (this function) or via the boxmask, regionmask, and exprmask functions. In this function, all points greater than the threshold are set to unity. The mask is the same in I,Q,U, and V. Note that exprmask is the most powerful method for making mask images.

In the Clark CLEAN, the mask image can usefully have any value between 0.0 and 1.0. Intermediate value discourage but do not rule out selection of clean components in that region. This is accomplished by multiplying the residual image by the mask prior to entering the minor cycle.

Note that if you do use a mask for the Clark or Hogbom Clean, it must cover only a quarter of the image. It is particularly important to check this when creating an image using a threshold. If it extends further, the easiest fix is to use getchunk and getchunk to set parts of it to zero.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>name of template image</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>mask</td>
<td>name of mask image</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>threshold</td>
<td>threshold for mask</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant 0.0Jy</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**
Example

```python
im.mask( image='bigimage', mask='bigmask', threshold='0.07Jy')
im.clean(mask='bigmask', model='3C273XC1.clean.masked', niter=1000)
```

Makes the image bigmask, and then sets it to unity for all points where the Stokes I in bigimage is greater than 0.07. Then clean using it as the mask.
Calculate a deconvolved image with selected mem (maximum entropy) algorithm

Description

Makes a mem image using either the Cornwell-Evans maximum entropy or maximum emptiness algorithms, using the single field or multi-field contexts. The maximum entropy algorithm is the default. The mem is performed on the residual image calculated from the visibility data currently selected. Hence the first step performed in mem is to transform the current model or models (optionally including a componentlist) to fill in the MODEL_DATA column, and then inverse transform the residual visibilities to get a residual image. This residual image is then deconvolved using the corresponding point spread function. This means that the initial model is used as the starting point for the deconvolution. Thus if you want to restart a mem, simply set the model to the model that was previously produced by clean. Mask images are used to constrain the region that is to be deconvolved. To make mask images, use either boxmask (to define a mask via the corner locations blc and trc) or mask (to define a mask via thresholding an existing image). The default mask is the inner quarter of the image.

The MEM deconvolution only operates on one Stokes parameter at a time. Joint MEM deconvolution for multiple Stokes parameters will be implemented in the future.


The mem algorithms possible are:

Cornwell-Evans Maximum Entropy (entropy) The classic "vm" or "vtess" deconvolution algorithm.

Cornwell-Evans Maximum Emptiness (emptiness) The historic, but largely undocumented, modification to the Cornwell-Evans algorithm which seeks a model image which is consistent with the data and simultaneously minimizes the number of pixels with no emission (meaning "with pixel values below the noise level").

Multi-field Maximum Entropy (mfentropy) Deconvolution is split into minor and major cycles. For each field, the MEM analog of a Clark Clean...
minor cycle is performed. In the major cycle, the emission thus modelled is subtracted either from the original visibilities (for multiple fields) or using a convolution (for only one field). The latter is much faster.

**Multi-field Maximum Emptiness (mfemptiness)** Just like mfentropy, but with emptiness.

The multi-field mem (mfentropy or mfemptiness) should be used if either of two conditions hold:

1. Multiple fields are to be deconvolved simultaneously OR
2. Primary beam correction is enabled. In this case, a mosaiced mem is performed.

Note that for the single pointing algorithms, only a quarter of the image may be deconvolved. If no mask is set, then the deconvolved region defaults to the inner quarter. If a mask larger than a quarter of the image is set, then only the quarter starting at the bottom left corner is used. However, for the multi-field imaging, the entire field may be imaged because the major cycles either do an exact subtraction from the visibilities or because PSF extent is more than twice the extent of the primary beam support.

Before mem can be run, you must run selectvis and defineimage. Before mem can be run with a multi-field algorithm, you should run setvp. You may want to run setmfcontrol before running mem with a multi-field algorithm, though the default control values may be acceptable.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Algorithm to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm</td>
<td>Algorithm to use</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>entropy</td>
</tr>
<tr>
<td></td>
<td>emptiness</td>
</tr>
<tr>
<td></td>
<td>mfentropy</td>
</tr>
<tr>
<td></td>
<td>mfemptiness</td>
</tr>
<tr>
<td></td>
<td>entropy</td>
</tr>
<tr>
<td>niter</td>
<td>Number of Iterations</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>20</td>
</tr>
<tr>
<td>sigma</td>
<td>Image sigma to try to achieve</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant 0.001Jy</td>
</tr>
<tr>
<td>targetflux</td>
<td>Target flux for final image</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant 1.0Jy</td>
</tr>
<tr>
<td>constrainflux</td>
<td>Constrain image to match target flux? else targetflux</td>
</tr>
<tr>
<td>used only to initialize model</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>displayprogress</td>
<td>Display the progress of the cleaning?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>model</td>
<td>Names of model images</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>keepfixed</td>
<td>Keep model fixed</td>
</tr>
<tr>
<td>allowed:</td>
<td>boolArray</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>complist</td>
<td>Name of component list</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>prior</td>
<td>Names of mem prior images</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>mask</td>
<td>Names of mask images (0=&gt;no emission, 1=&gt;emission permitted</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>image</td>
<td>Names of restored images</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>residual</td>
<td>Names of residual images</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>
Returns
bool

Example

```
im.mem(model='3C273XC1.mem.model',
mask='3C283XC1.mask', niter=40, sigma='0.001Jy')
```
imager.nnls.html

**imager.nnls - Function**

2.3.1 Calculate a deconvolved image using the NNLS algorithm

**Description**

Solve for the model brightness using the Briggs’ Non-Negative Least Squares algorithm. Since NNLS works only on the $I$ image, the $I$ pixels in the current image is set to zero where the fluxmask is $> 0.0$, then NNLS is used to estimate the $I$-pixels for that region. The deconvolution is performed on the residual image calculated from the visibility data currently selected. Hence the first step performed in clean is to transform the current model to fill in the MODEL_DATA column, and then inverse transform the residual visibilities to get a residual image. This residual image is then deconvolved using the corresponding point spread function. Some other points to remember are that rather than explicit boxes, mask images are used to constrain the region that is to be deconvolved. For NNLS, there are two masks, the fluxmask specifying the region within which flux is allowed, and the datamask specifying the region of the dirty image to be used as constraints. Typically the datamask will be somewhat larger than the fluxmask. On a large machine, a practical limit to both will be about 5000-6000 pixels. Hence NNLS is only useful for compact tools. (For more details, see the Briggs thesis). To make mask images, use either boxmask (to define a mask via the corner locations blc and trc) or mask (to define a mask via thresholding an existing image).

On the canonical CASA machine with 64MBytes of physical memory, you should try to keep the product of the pixels in the fluxmask and the datamask below about 5-10 million. Otherwise the solution phase will swap badly.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>Name of image</td>
</tr>
<tr>
<td></td>
<td>allowed: stringArray</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>keepfixed</td>
<td>Keep model fixed</td>
</tr>
<tr>
<td></td>
<td>allowed: boolArray</td>
</tr>
<tr>
<td></td>
<td>Default: false</td>
</tr>
<tr>
<td>complist</td>
<td>Name of component list</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>niter</td>
<td>Number of Iterations, set to zero for no NNLS</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
<tr>
<td>tolerance</td>
<td>Tolerance for solution</td>
</tr>
<tr>
<td></td>
<td>allowed: double</td>
</tr>
<tr>
<td></td>
<td>Default: 1e-06</td>
</tr>
<tr>
<td>fluxmask</td>
<td>Name of mask for allowed flux</td>
</tr>
<tr>
<td></td>
<td>allowed: stringArray</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>datamask</td>
<td>Name of mask for constraint pixels in dirty image</td>
</tr>
<tr>
<td></td>
<td>allowed: stringArray</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>image</td>
<td>Names of restored images</td>
</tr>
<tr>
<td></td>
<td>allowed: stringArray</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>residual</td>
<td>Names of restored images</td>
</tr>
<tr>
<td></td>
<td>allowed: stringArray</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: false</td>
</tr>
</tbody>
</table>

| Returns | bool |

| Example  |

```python
im.nnls(image='3C273XC1.nnls.image', model='3C273XC1.nnls.model', fluxmask='3C283XC1.fluxmask', datamask='3C273XC1.datamask', niter=1000, tolerance=0.00001)
```
**imager.open** - Function

2.3.1 Open a new MeasurementSet, for processing, closing current MeasurementSet

**Description**

Close the current MeasurementSet and open a new MeasurementSet instead. The current state of imager is retained, except for the data selection.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>thems</td>
<td>New MeasurementSet to be processed</td>
</tr>
<tr>
<td>compress</td>
<td>Compress calibration columns?</td>
</tr>
<tr>
<td>usescratch</td>
<td>If true: Imager will use corrected data column and make scratch columns of they donot exist</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td></td>
</tr>
<tr>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

bool
imager.pb.html

**imager.pb - Function**

2.3.1 Applies or corrects for a primary beam

**Description**

Multiply (`operation='apply'`) or divide (`operation='correct'`) by the primary beam function. The primary beam can be applied to images and/or Componentlists.

If `pointingcenter==false` then you must specify `inimage` and the pointing center is taken from its reference direction. Otherwise, `pointingcenter` must be a Direction measure. It cannot take on the value `True`.

The applied primary beam function is determined as follows. If you used function `Imager.setvp` to set an external voltage pattern table, then this is where the applied primary beam will come from (regardless of whether you set `inimage` or not). If you did not run this function, then you must supply argument `inimage`. The telescope name embedded in its Coordinate System will be used to determine the primary beam function.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>inimage</td>
<td>Input image to apply beam to</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>outimage</td>
<td>Output image after beam is applied</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>incomps</td>
<td>Input Componentlist table name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>outcomps</td>
<td>Output Componentlist table name</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>operation</td>
<td>Operation</td>
<td>string</td>
<td>correct</td>
</tr>
<tr>
<td>pointingcenter</td>
<td>Pointing center for primary beam application: default</td>
<td></td>
<td>apply</td>
</tr>
<tr>
<td></td>
<td>N.Pole</td>
<td></td>
<td>variant</td>
</tr>
<tr>
<td>parangle</td>
<td>Parallactic angle for calculation</td>
<td>any</td>
<td>variant 0.0deg</td>
</tr>
<tr>
<td>pborvp</td>
<td>Primary Beam or Voltage Pattern</td>
<td>string</td>
<td>vb</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>pb</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```plaintext
# make a flat image
im.make('flat.image');
```
ia.open('flat.image');
arr=ia.getchunk();
arr[0:len(arr), 0:len(arr[0])] = 1.0;
ia.putchunk(arr);
ia.done()
arr = false;
#
# as we are using "pointingcenter=F", it defaults to the image center
im.pb(inimage='flat.image', outimage='pb.image', pointingcenter=F)
imager.plotsummary.html

**imager.plotsummary - Function**

2.3.1 Plot a summary of field and spectral window ids

**Description**

Performs a simple plot of the field and spectral window IDs versus time (after sorting).

**Arguments**

**Returns**

bool

**Example**

```plaintext
m = fitstoms('3C273XC1.ms', '3C273XC1.fits'); m.close()
im.open('3C273XC1.ms')
im.plotsummary()
```
imager.plotuv.html

**imager.plotuv - Function**

2.3.1 Plot the uv coverage

**Description**

Performs a simple plot of the uv coverage of all selected data. Optionally, plotuv will rotate the uvw coordinates to the specified phase center (set via defineimage).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>rotate</td>
<td>Rotate uvw coordinates to specified phase center?</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```plaintext
im.open('3C273XC1.ms')
im.plotuv(false)
```
imager.plotvis.html

**imager.plotvis - Function**

2.3.1 Plot the visibility amplitudes as a function of u-v radius (also, see visplot tool)

**Description**

Performs a simple plot of the visibility amplitudes of all selected data.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Type of plot: can contain all, observed, corrected, model, residual allowed: string Default: all observed corrected model residual all</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Increment in points to plot allowed: int Default: 1</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
im.open('3C273XC1.ms')
im.plotvis(increment=10)
```
imager.plotweights.html

**imager.plotweights - Function**

2.3.1 Plot the visibility weights as a function of u-v radius

**Description**

Performs a plot of the visibility weights of all selected data (stored in the IMAGING_WEIGHT column of the MeasurementSet). The plot can be of the gridded weights (type='gridded') or ungridded.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>gridded</td>
<td>Do gridded plot?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>increment</td>
<td>Increment in points to plot</td>
<td>int</td>
<td>1</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
im.open('3C273XC1.ms')
im.defineimage(cellx='0.7arcsec', celly='0.7arcsec')
im.weight('briggs')
im.plotweights(gridded=True, increment=10)
```
A mask image is an image with the same shape as the other images but with values between 0.0 and 1.0 as a pixel value. Mask images are used in imager to control the region selected in a deconvolution.

In the Clark CLEAN, the mask image can usefully have any value between 0.0 and 1.0. Intermediate value is discouraged but do not rule out selection of clean components in that region. This is accomplished by multiplying the residual image by the mask prior to entering the minor cycle. Note that if you do use a mask for the Clark or Hogbom Clean, it must cover only a quarter of the image. regionmask does not enforce this requirement.

The function regionmask also allows multiple regions to be used. A record of the regions can be made as in the example below.

Regions can be made in many different ways using the regionmanager functions. An example using wbox function is given below. The default regionmanager tool 'rg' can be used for cases the user want to have flexibility in manipulating regions. The region parameter takes a record that comes from the regionmanager output. The parameter boxes allow the user to sent in a list of 4 elements numbers representing blc’s and trc’s

If both the parameters, regions and boxes are used the a union is done with the two sets of region thus defined.

Arguments
Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>name of mask image</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>region</td>
<td>Region record usually from regionmanager</td>
<td>record</td>
<td>unset</td>
</tr>
<tr>
<td>boxes</td>
<td>list of 4 elements lists e.g. [[xblc1, yblc1, xtrc1, ytrc1], ...]</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>circles</td>
<td>list of 3 elements lists e.g. [[rad0, xcen0, ycen0], [rad1, xcen1, ycen1], ...]</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>value</td>
<td>Value to set the mask to</td>
<td>double</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Returns

bool

Example

Makes a mask then cleans using it.

```python
im.open('test.ms')
im.selectvis(field=0, spw=0)
im.defineimage(nx=400, cellx='0.001arcsec', phasecenter=0)
a=[100.0, 100.0, 200, 200.0]
b=[50, 50, 80, 80]
im.regionmask(mask='bigmask', boxes=[a, b])
im.clean(mask='bigmask', model='3C273XC1.clean.masked', niter=1000)
```

Another example using rg.wbox function:

```python
ia.open('dirty')
cs = ia.coordsys()
```
rg.setcoordinates(cs.record())

r1 = dg.wbox(blc=['173pix', '347pix'], trc=['183pix', '370pix'])
im.regionmask(mask='bigmask',region=r1)

Or using a dict of regions:

r2=rg.wbox(blc=['180pix', '344pix'], trc=['191pix', '369pix'])
r3=rg.wbox(blc=['189pix', '341pix'], trc=['204pix', '364pix'])
regs={"reg1":r1, "reg2":r2, "reg3":r3}
rec=rg.makeunion(regs)
im.regionmask(mask='bigmask',region=rec)

If quantities are to be used to define regions the following is an example

**imager.regiontoimagemask - Function**

[2.3.1] union a mask image with various regions

**Description**

This function is very similar to regionmask function except that the mask image has to be existant already and this is an independent helper function (i.e does not care about the state of the imager tool... e.g does not need imager to have an attached ms).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>name of mask image</td>
</tr>
<tr>
<td>allowed: string</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>region</td>
<td>Region record usually from regionmanager</td>
</tr>
<tr>
<td>allowed: record</td>
<td></td>
</tr>
<tr>
<td>Default: unset</td>
<td></td>
</tr>
<tr>
<td>boxes</td>
<td>list of 4 elements lists e.g ([xblc1, yblc1, xtrc1, ytrc1], [xblc2, yblc2, xtrc2, ytrc2])</td>
</tr>
<tr>
<td>allowed: any</td>
<td></td>
</tr>
<tr>
<td>Default: variant</td>
<td></td>
</tr>
<tr>
<td>circles</td>
<td>list of 3 elements lists e.g ([rad0, xcen0, ycen0], [rad1, xcen1, ycen1], .....)</td>
</tr>
<tr>
<td>allowed: any</td>
<td></td>
</tr>
<tr>
<td>Default: variant</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>Value to set the mask to</td>
</tr>
<tr>
<td>allowed: double</td>
<td></td>
</tr>
<tr>
<td>Default: 1.0</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

1393
Makes a mask then cleans using it.

\[
a = [100.0, 100.0, 200, 200.0]
b = [50, 50, 80, 80]
\]
\[im.regiontoimagemask(mask='bigmask', boxes=[a, b])\]
\[im.clean(mask='bigmask', model='3C273XC1.clean.masked', niter=1000)\]

Another example using rg.wbox function:
\[ia.open('dirty')\]
\[cs = ia.coordsys()\]
\[rg.setcoordinates(cs.record())\]
\[r1 = dg.wbox(blc=['173pix', '347pix'], trc=['183pix', '370pix'])\]
\[im.regionmask(mask='bigmask',region=r1)\]

Or using a dict of regions:
\[r2=rg.wbox(blc=['180pix', '344pix'], trc=['191pix', '369pix'])\]
\[r3=rg.wbox(blc=['189pix', '341pix'], trc=['204pix', '364pix'])\]
\[regs={"reg1":r1, "reg2":r2, "reg3":r3}\]
\[rec=rg.makeunion(regs)\]
\[im.regionmask(mask='bigmask',region=rec)\]

If quantities are to be used to define regions the following is a an example

\[im.regionmask(mask='joetest',boxes=['15:23:32.902','+05.19.32.089','15:22:28.631','+05.28.52.50'])\]
imager.residual.html

**imager.residual - Function**

2.3.1 Calculate the residual image with respect to current model and component list

**Description**

Calculate the residuals corresponding to the model and componentlist. *Note that the model visibilities are updated.*

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>Names of input models</td>
</tr>
<tr>
<td>complist</td>
<td>Name of component list</td>
</tr>
<tr>
<td>image</td>
<td>Names of output residual images</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

- im.residual(model='3C273XC1.clean', complist='3C273XC1.cl', image='3C273XC1.clean.residual')
imager.restore.html

**imager.restore - Function**

2.3.1 Calculate the restored image with restored model, component list, and residuals

**Description**

Restore the residuals to a smoothed version of the model. The model images are convolved with the specified Gaussian beam and then the residual images are added. *Note that the model visibilities are updated and thus reflect the model and componentlist that was used.* Use setbeam to set the beam parameters.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>Names of input model</td>
<td></td>
</tr>
<tr>
<td>complist</td>
<td>Name of component list</td>
<td></td>
</tr>
<tr>
<td>image</td>
<td>Names of output restored images</td>
<td></td>
</tr>
<tr>
<td>residual</td>
<td>Names of residual images</td>
<td></td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background</td>
<td></td>
</tr>
</tbody>
</table>

| allowed: | stringArray               |                                                                             |
| Default: | string                     |                                                                             |
| allowed: | stringArray               |                                                                             |
| Default: | string                     |                                                                             |
| allowed: | stringArray               |                                                                             |
| Default: | string                     |                                                                             |
| allowed: | bool                       |                                                                             |
| Default: | false                      |                                                                             |

**Returns**

bool

**Example**

1396
- `im.setbeam(bmaj='2.0arcsec', bmin='2.0arcsec')`
- `im.restore(model='3C273XC1.clean', image='3C273XC1.clean.restored',`
**imager.updateresidual - Function**

2.3.1 Calculate the residual and restored images with new modified model, component list,

**Description**

This function is for efficiency and speed purpose only. Same as restore It is to be used after you have used clean or mem ...but you wish to tweak the model image, say by clipping unwanted components and it will avoid unnecessary recalculating of psf but will do a proper prediction of the new model visibilities and recalculate residual and restored images.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>Names of input model</td>
<td>stringArray</td>
<td></td>
</tr>
<tr>
<td>complist</td>
<td>Name of component list</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>image</td>
<td>Names of output restored images</td>
<td>stringArray</td>
<td></td>
</tr>
<tr>
<td>residual</td>
<td>Names of residual images</td>
<td>stringArray</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

- im.setbeam(bmaj='2.0arcsec', bmin='2.0arcsec')
- im.restore(model='3C273XC1.clean', image='3C273XC1.clean.restored',


imager.sensitivity - Function

2.3.1 Calculate rms sensitivity

Description

NB: The implementation in this function will be removed for CASA v4.5. We now recommend that the im.apparentsens() function be used instead of this one, especially if their weights are initialized and calibrated.

Calculate the point source sensitivity for the selected data, both absolutely and relatively (to that for natural weighting).

To do the calculation, we use the imaging weights (in the column called IMAGING_WEIGHT) as calculated from the WEIGHT column, and an estimate of the effective net bandwidth and integration time. The calculation therefore includes all the effects of weight and filter.

The output is an array with mixed elements. Counting from zero, the second element (out[1]) is the net sensitivity, third element is the ratio of the reduction in sensitivity due to the chosen weighting scheme. This ratio is 1.0 for Natural weight and greater than one for all other weighting schemes. (NOTE: Further testing is required of this value and hence this is kept separate for now).

The sensitivity calculations require Tsys and collecting area of the antenna. These quantities are not known from the MS. The sensitivity is therefore returned in units of Jy m^2/K. Multiplying the second elements with the ration of the Tsys and effective antenna collecting area will give the sensitivity in Jy/beam units.

The fourth elements of the return value is a record with the following keys: 'nbaselines', 'effectiveintegration', 'effectivebandwidth', 'sumwt' and 'spwid'. These can be used to get the number of baselines used, effective integration time (in sec), the effective bandwidth (in Hz), the sum of weights and the absolute spectral window IDs used.

Arguments
Outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pointsource</td>
<td>Calculated point source sensitivity (Jy m^2 / (K beam))</td>
</tr>
<tr>
<td>relative</td>
<td>Calculated relative sensitivity</td>
</tr>
<tr>
<td>sumweights</td>
<td>Calculated sum of weights</td>
</tr>
<tr>
<td>senrec</td>
<td>Record per SPW per chan sensitivity calculations</td>
</tr>
</tbody>
</table>

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>async</td>
<td>Run asynchronously in the background</td>
</tr>
</tbody>
</table>

Returns

bool

Example

```python
a = im.sensitivity(False);
print 'Sensitivity =', a[1];
print 'Relative to Natural Weighting = ', a[2];
```
Description

This function calculates the point source sensitivity for the data selected by im.selectvis(...), and according to the imaging weighting parameters specified in im.weight(...) and im.defineimage(...). The calculation is performed solely using the weight information stored in the MS WEIGHT column (WEIGHT_SPECTRUM tbd), and as adjusted by the net imaging weighting function (natural, uniform, robust, taper, etc.). Therefore, it is assumed that the MS WEIGHTs have been properly initialized and calibrated along with the visibility data. As long as the WEIGHTs are in the inverse square units of the visibilities (i.e., inverse variance weights), the calculation should yield the real theoretical imaging sensitivity for data at any stage of the calibration (though data at early and intermediate stages of calibration may not be sufficiently coherent for imaging at high–or even modest–fidelity).

Two values are reported in the logger and returned (see example below). First, the apparent sensitivity (in the units implied by the WEIGHTs’ units), for the specified imaging weighting scheme. Second, a unitless factor describing the ratio of the apparent sensitivity to that obtained with pure 'natural' weighting (the nominal peak sensitivity). When 'natural' weighting is selected, this ratio factor will be 1.0; all other weighting choices will yield an apparent sensitivity ratio greater than 1.0.

Currently, this function reports only the continuum sensitivity for the selected data, and in particular, for the aggregate bandwidth indicated by the spectral window selection. The calculation further assumes that the visibility samples are each entirely independent (i.e., no redundant samples such as would occur for overlapping spectral windows).

A future version of this function will support reporting a sensitivity spectrum for the spectral line case (including support for WEIGHT_SPECTRUM). For now, spectral line sensitivity may be reasonably estimated by dividing the reported sensitivity by the square root of the fractional bandwidth of a single image channel, or by selecting a bandwidth matching the width of a single image channel.

Arguments
### Outputs

<table>
<thead>
<tr>
<th>pointsource</th>
<th>Calculated apparent point source sensitivity (in units implied by the MS weights)</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>relative</th>
<th>Ratio of apparent sensitivity relative to natural weighting</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

### Inputs

<table>
<thead>
<tr>
<th>async</th>
<th>Run asynchronously in the background</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>

### Returns

bool

### Example

```python
# open and set up selection and image plane parameters
im.open('mydata.ms')
im.selectvis(field='2',spw='0')
im.defineimage(mode='mfs',spw=0, stokes='I', cellx='15arcsec', celly='15arcsec', nx=256, ny=256)

# report natural weighting sensitivity
im.weight(type='natural')
nat=im.apparentsens();
print 'Natural Sensitivity = ', nat[1];
print 'Relative to Natural Weighting = ', nat[2];

# switch to uniform weighting
im.weight(type='uniform')
uni=im.apparentsens();
print 'Uniform Sensitivity = ', uni[1];
print 'Relative to Natural Weighting = ', uni[2];

# switch to briggs weighting
im.weight(type='briggs',robust=0.0)
rob=im.apparentsens();
print 'Briggs Sensitivity = ', rob[1];
```
print 'Relative to Natural Weighting = ', rob[2];
im.close()
**Imager.setbeam** - Function

2.3.1 Set the beam parameters for clean restoration

**Description**

This sets the clean beam that will be used in all restoration operations.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>bmaj</td>
<td>Major axis of beam</td>
<td>any</td>
<td>variant 1.0arcsec</td>
</tr>
<tr>
<td>bmin</td>
<td>Minor axis of beam</td>
<td>any</td>
<td>variant 1.0arcsec</td>
</tr>
<tr>
<td>bpa</td>
<td>Position angle of beam</td>
<td>any</td>
<td>variant 0deg</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

bool
imager.selectvis.html

**imager.selectvis - Function**

2.3.1 Select visibilities for subsequent processing

**Description**

This setup tool function selects which data are to be used subsequently. After invocation of selectvis, only the selected data are operated on. Thus, for example, in imaging, only the selected data are gridded into an image, and in plotting, only the selected data are plotted.

Data can be selected by field and spectral window ids. Note that all data thus selected are passed to imaging, and may or may not be imaged, depending on how the image was constructed using defineimage. For example, in mosaicing, use fieldid in defineimage to control what pointing is used to define the field center, and use fieldid in selectvis to control what pointings are used in the imaging.

For spectral processing, it is possible to make cubes out multi-spectral window selections but the selection and combination can be a bit confusing (any hint at how to make it clearer is welcome).

If the default values are not used, then data to be used can be selected channel wise. The

- **nchan** is the number of data channels selected. It defaults to -1 (interpreted as all channels).

- **start** is the first channel from input dataset that is to be used. It defaults to 0 (i.e. first channel).

- **step** gives the increment between selected input channels. It defaults to 1 channel. A value of n means that n-1 data channels will not not be used.

By choosing the parameters for selectvis and defineimage correctly, one may obtain various mappings of visibility channels to image channels. For example, to average 512 visibility channels into 64 image channels (producing image channels consisting of 8 visibility channels):

```
im.defineimage(mode='channel', spw=0, nchan=64, start=1, step=8);
im.selectvis(spw=0, nchan=512, start=1, step=1) im.clean(.....);
```

This averages the spectral channels during the gridding process. If one wanted to only include every 8th channel in the deconvolution, one would do:

```
im.selectvis(nchan=64, start=1, step=8) im.defineimage(mode='channel', nchan=64, start=1, step=8); im.clean(....);
```

For velocity and opticalvelocity modes, the mstart and mstep are the start and step velocities as strings.

1406
im.defineimage(mode='velocity', nchan=64, start='20 km/s', step='-100m/s');
im.selectvis(spwid=[-1]); # selecting all data spectral windows im.clean(...);
If the image and data selections differ, then averaging is done during the
gridding and degridding process in the image deconvolution.
im.defineimage(mode='channel', nchan=64, start=1, step=8);
im.selectvis(nchan=512, start=1, step=1) im.clean()
Note: The channels numbers used in defineimage and selectvis refers to
the same channel. So if a channel is not selected in selectvis but is selected
in defineimage, then blank channels image are made. The example below will
result in the having the first 6 (0-5) channels in the image to be blank.
im.selectvis(nchan=50, start=6, step=1) # selected chan 6-55
im.defineimage(mode='channel', nchan=50, start=0, step=1);
# will try to image channel 1-50. But as previously only channel 6-55 # was
selected only channel 6-50 will have data; images of channels # 1-5 are blank
im.clean(...)
For multi-spectral window cube imaging the selection of the data can be done
as follows
im.selectvis(nchan=[50,60], start=[0,0], step=[1,1], spw=[0,1])
im.defineimage(mode='channel', nchan=110, start=0, step=1, spw=[0,1]);
The above means that you would make a data selection of 50 channels
(starting from 0 steping 1) from the first spectral window and 60 channels
(starting from 1 steping 1). The defineimage defines the image to be a cube of
110 channels. The caveat is the step size in the frequency direction is the step
size of the first spectral window. If the step size of channels of the two spectral
windows are different then one is better of defining the image cube in
velocities (e.g. as below).
im.selectvis(nchan=[50,60], start=[0,0], step=[1,1], spw=[1,2])
im.defineimage(mode='velocity', nchan=200, mstart='20km/s',
mstep='-100m/s');

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>vis</td>
<td>Measurementset for which this selection applies; an empty string &quot;&quot; implies that it is to be applied in ms used in open</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>nchan</td>
<td>Number of channels to select</td>
<td>intArray</td>
<td>-1</td>
</tr>
<tr>
<td>start</td>
<td>Start channels (0-relative)</td>
<td>intArray</td>
<td>0</td>
</tr>
<tr>
<td>step</td>
<td>Step in channel number</td>
<td>intArray</td>
<td>1</td>
</tr>
<tr>
<td>spw</td>
<td>Spectral Window Ids (0 relative) to select; -1 interpreted as all</td>
<td>any</td>
<td>variant -1</td>
</tr>
<tr>
<td>field</td>
<td>Field Ids (0 relative) or Field names (msselection syntax and wildcards are used) to select</td>
<td>any</td>
<td>variant -1</td>
</tr>
<tr>
<td>baseline</td>
<td>Antenna Ids (0 relative) or Antenna names (msselection syntax and wildcards are used) to select</td>
<td>any</td>
<td>variant -1</td>
</tr>
<tr>
<td>time</td>
<td>Limit data selected to be within a given time range. The syntax is defined in the msselection link</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>scan</td>
<td>Limit data selected on scan numbers. The syntax is defined in the msselection link</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>intent</td>
<td>Limit data selected on observation intent. The syntax is defined in the msselection link</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>observation</td>
<td>Limit data using observation IDs. The syntax is defined in the msselection link</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>uvrange</td>
<td>Limit data selected on uv distance. The syntax is defined in the msselection link</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>taql</td>
<td>For the TAQL experts, flexible data selection using the TAQL syntax</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>usescratch</td>
<td>If True: imager will use CORRECTED_DATA column</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>
Returns

bool

Example

```python
im.open('3C273XC1.MS');
im.selectvis(nchan=512, start=1, step=1, taql='SCAN_NUMBER > 10 && FIELD_ID==2')
```

Time range selection

```python
im.selectvis(field=range(0,10), time='> 2000/09/21/12:00:00')
```

select some antennas, for all fields that begins with \tt 'ngc'

```python
im.selectvis(field='ngc*', baseline=[0, 10, 20])
```

And for those that the standard parameters are not flexible enough there is the taql parameter. This for people who knows the different columns of the MeasurementSet

```python
im.selectvis(taql="ANTENNA1==0 &\ ANTE\"NA2==3")
```

Imager allows the user to make an image from multiple ms, without the need to concatenate them. To do this then the im.open method should \bf not be used at all but multiple calls of selectvis with the parameter vis pointing to each ms should be used. The other parameters can be used to make selection on each ms

```python
im.selectvis(vis='myms1.ms', field=0, spw=[0,1], nchan=[40, 50], start=[5,10])
im.selectvis(vis='myms2.ms', field=10, spw=[2], nchan=[40], start=[5])
im.selectvis(vis='myms3.ms', field=range(0,10), time='> 2002/10/15/20:30:45')
```
Compute the model visibility for a specified source flux density, and insert into the MODEL_DATA column. The source flux density for a set of standard flux density reference sources may optionally be pre-computed, by setting the input flux density to -1 (the default). At present, these include 3C286, 3C48, 3C147, 3C138, and 1934-638. In this case, if the source is not in this set, an unpolarized flux density of 1 Jy will be assumed. Users may also specify standard='SOURCE' to use the model(s) in the SOURCE MODEL column of the SOURCE subtable. Users may also specify a model image that will be scaled to the specified total flux density (or that computed for reference sources). When a model image is specified, setjy will only permit processing one field, and will currently only process Stokes I.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>Field Id (0-relative) or name</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>spw</td>
<td>Spectral Window Id. (0-relative)</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>modimage</td>
<td>A model image</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>fluxdensity</td>
<td>Specified flux density (I,Q,U,V) in Jy (lookup the value; use 1.0 if not found)</td>
<td>doubleArray</td>
<td>0.0 0.0 0.0 0.0</td>
</tr>
<tr>
<td>standard</td>
<td>Flux density standard</td>
<td>string</td>
<td>Baars</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Perley 90</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Perley-Taylor 95</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Perley-Taylor 99</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Perley-Butler 2010</td>
</tr>
<tr>
<td>scalebychan</td>
<td>Do the flux scaling on a per channel basis else on a spw basis. Effectively True if fluxdensity is specified.</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>spix</td>
<td>Spectral index for fluxdensity. ( S = \text{fluxdensity} \times \left(\frac{\text{freq}}{\text{reffreq}}\right)^{\text{spix}} )</td>
<td>doubleArray</td>
<td></td>
</tr>
<tr>
<td>reffreq</td>
<td>Reference frequency for spix.</td>
<td>any</td>
<td>variant 1GHz</td>
</tr>
<tr>
<td>polindex</td>
<td>Coefficients for Taylor expansion of Polarization index</td>
<td>doubleArray</td>
<td>0.0</td>
</tr>
<tr>
<td>polangle</td>
<td>Coefficients for Taylor expansion of Polarization angle in radians</td>
<td>doubleArray</td>
<td>0.0</td>
</tr>
<tr>
<td>rotmeas</td>
<td>Rotation measure (rad/lambda**2)</td>
<td>double</td>
<td>0.0</td>
</tr>
<tr>
<td>time</td>
<td>Time range to operate on</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>scan</td>
<td>Scan(s) to operate on</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>intent</td>
<td>Observation intent</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>observation</td>
<td>Observation ID(s) to operate on</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>
Returns
record

Example

```
im.setjy(fieldid=2, spwid=-1, fluxdensity=[2.6,0.2,0.3,0.5],standard='Baars')
```

Compute the model visibility for field id. 2 to the specified point-source (I,Q,U,V) for all spectral windows id.’s on the Baars flux density scale.
imager.ssoflux.html

imager.ssoflux - Function

2.3.1 Use setjy instead.

Description

This was an experimental clone of setjy while flux calibration with Solar System objects was being tested. It has been merged back into setjy.*

Arguments

Inputs

Returns

bool

Example
3.1 Set various cycle control parameters for multi-field and wide-field imaging.

**Description**

Control parameters for mosaicing or wide-field imaging which are not required in single field deconvolution are set here to streamline the user interface. As multifield and widefield imaging is accomplished by deconvolution in cycles, many of these parameters control how the deconvolution cycles are ended.

- **cyclefactor**: this parameter helps in lowering or increasing the threshold at which the deconvolution cycle will stop and degrid and subtract from the visibilities. For very bad PSFs you may want to reconcile with the visibilities often, thus a larger number is required here...(4 to 5). For very well behaved data you may want to deconvolve deep before reconciling: a lower number is used (1.5 to 2.0).

- **cyclespeedup**: this is used if the PSF is not well behaved and you want clean to raise by 2 the threshold if it has not reached the threshold in this number of iteration

- **cyclemaxpsffraction**: similar to cyclefactor, but this is an explicit fraction of the PSF peak. The final threshold is computed using min(cyclemaxpsffraction, cyclefactor * maxPSFsidelobe). Valid values are between 0.0 and 1.0.

- **stoplargenegatives**: This parameter is exclusively for when using multiscale clean. This is used to stop the component search when the largest scale has found this number of negative components. -1 here means that continue component search even if the largest component is negative.

- **stoppointmode**: Again exclusively for when using multiscale clean. The clean will stop if the smallest scale receives this number of consecutive components.

- **minpb**: This is to defined up to what level the voltage pattern is going to applied when using setvp. The default is 0.1 of the primary beam or the voltage pattern defined for the antenna.

- **scaletype**: This parameter cab be NONE or SAULT. If NONE the image is not scaled, if SAULT is used the image is weighted so that the noise is
kept uniform across the image. The next two parameters defines how the SAULT weighting is limited. Obviously then the flux scale is not uniform across the image. To get the right flux multiply the image with the fluxscale image.

constpb: this parameter defines up to what amplitude of the Primary beam the noise floor is kept uniform, when using SAULT as scaletype.

fluxscale: use this to give a filename to store the factor image to apply to the image to get the fluxscale right.

flatnoise: (default true) Set to false if you want clean components for mosaic to be searched in the residual image that is effectively multiplied by the beam$^2$. This means when the noise is determined after the antenna, searching in the optimum domain of signal/(sigma$^2$). For meter wavelengths where noise is determined by the sky, it is no longer optimal.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>cyclefactor</td>
<td>Cycle threshold = max_resid * min(cyclemaxpsffraction, this * max_sidelobe)</td>
<td>double</td>
<td>1.5</td>
</tr>
<tr>
<td>cyclespeedup</td>
<td>Cycle threshold doubles in this number of iterations</td>
<td>double</td>
<td>-1</td>
</tr>
<tr>
<td>cyclemaxpsffraction</td>
<td>Cycle threshold = max_resid * min(this, cyclefactor * max_sidelobe)</td>
<td>double</td>
<td>0.8</td>
</tr>
<tr>
<td>stoplargenegatives</td>
<td>Stop the multiscale cycle for the first n cycles when a negative comp is found on the largest scale</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>stoppointmode</td>
<td>Stop multiscale altogether if the smallest scale receives this many consecutive components</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>minpb</td>
<td>Minimum PB level to use</td>
<td>double</td>
<td>0.1</td>
</tr>
<tr>
<td>scaletype</td>
<td>Image plane flux scale type</td>
<td>string</td>
<td>SAULT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NONE</td>
</tr>
<tr>
<td>constpb</td>
<td>In Sault weighting the flux scale is constant above this PB level</td>
<td>double</td>
<td>0.4</td>
</tr>
<tr>
<td>fluxscale</td>
<td>Names of flux scale images for mosaicing</td>
<td>stringArray</td>
<td></td>
</tr>
<tr>
<td>flatnoise</td>
<td>Set to false if clean component search is to be done in an optimal signal/noise residual image if true will clean in a constant noise image</td>
<td>bool</td>
<td>true</td>
</tr>
</tbody>
</table>

**Returns**

bool
Example

```
im.setmfcontrol(cyclefactor=2.0, cyclespeedup=niter/10, cyclemaxpsffraction=0.8, stoplargenegatives=T, stoppointmode=10, fluxscale='image.fluxscale');
```
**imager.setoptions.html**

**imager.setoptions - Function**

2.3.1 Set some general options for subsequent processing

**Description**

This function is for setting different gridding and memory options.

**ftmachine** The options for ftmachine are:

- **ft** Standard interferometric gridding
- **sd** Standard single dish gridding
- **both** ft and sd as appropriate.

- **wproject** option for using the wpproject algorithm for wide-field imaging; when this option is used the parameter **wprojplanes** define the number of convolution functions to be used.

- **mosaic** option to use the grider that uses the primary beam as the convolution function in gridding.

**cache** The size of the cache used (in complex pixels) during the gridding process. The default is to use half the physical memory of the machine as specified by the aipsrc variable system.resources.memory.

**tile** The side of the tile (in complex pixels) during the gridding process.

**gridfunction** The gridding function used. Currently only Box-car ('BOX') and Prolate Spheriodal Wave Function ('SF') are supported. In the case of Single-Dish imaging the Primary Beam ('PB'), Gaussian ('GAUSS'), and Gaussian * Jinc ('GJINC') also can be used.

**location** For some unusual types of image, one needs to know the location to be used in calculating phase rotations. For example, one can specify images to be constructed in azel, in which case, an antenna position must be chosen. One can use functions of measures: either observatory to get the position of a named observatory (e.g. meobservatory('ATCA')) or position to set the position (e.g. meposition('wgs84',30deg,40deg,10m)). Although this information is available from the MeasurementSet, what location is ambiguous in some cases e.g. VLBI.

**padding** When gridding and transforming, the array may be padded by this factor in the image plane. This reduces aliasing, especially in wide-field cleaning.
usemodelcol  if this is false it tells imager to create and use the model visibility on the fly and in memory as far as possible...otherwise if it is True then imager will use the MODEL_DATA column to do this.

wprojplanes  this parameter is used only of ftmachine is set to wproject. This defines how many convolution functions is used in the Wprojection griddler (a -1 implies an automatic determination).

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftmachine</td>
<td>Fourier transform machine</td>
</tr>
<tr>
<td>allowed: string</td>
<td>Default: ft</td>
</tr>
<tr>
<td>cache</td>
<td>Size of gridding cache in complex pixels; default use half</td>
</tr>
<tr>
<td>allowed: int</td>
<td>the memory available on computer</td>
</tr>
<tr>
<td>Default: -1</td>
<td></td>
</tr>
<tr>
<td>tile</td>
<td>Size of a gridding tile in pixels (in 1 dimension)</td>
</tr>
<tr>
<td>allowed: int</td>
<td>Default: 16</td>
</tr>
<tr>
<td>gridfunction</td>
<td>Gridding function</td>
</tr>
<tr>
<td>allowed: string</td>
<td>Default: SF</td>
</tr>
<tr>
<td>location</td>
<td>Location used in phase rotations</td>
</tr>
<tr>
<td>allowed: any</td>
<td>Default: variant</td>
</tr>
<tr>
<td>padding</td>
<td>Padding factor in image plane (&gt;=1.0)</td>
</tr>
<tr>
<td>allowed: double</td>
<td>Default: 1.0</td>
</tr>
<tr>
<td>freqinterp</td>
<td>Interpolation mode in frequency: options: nearest, linear,</td>
</tr>
<tr>
<td>allowed: string</td>
<td>cubic, spline</td>
</tr>
<tr>
<td>Default: nearest</td>
<td></td>
</tr>
<tr>
<td>wprojplanes</td>
<td>No of gridding convolution functions used in wproject-ft</td>
</tr>
<tr>
<td>machine (-1 means let the code decide this number)</td>
<td>allowed: int Default: -1</td>
</tr>
<tr>
<td>epjtablename</td>
<td>E-Jones table name. This is used if applypointingoffsets</td>
</tr>
<tr>
<td>is set to True.</td>
<td>allowed: string Default: nearest</td>
</tr>
<tr>
<td>applypointingoffsets</td>
<td>Apply pointing offset corrections during deconvolution.</td>
</tr>
<tr>
<td>allowed: bool</td>
<td>Default: false</td>
</tr>
<tr>
<td>dopbgriddingcorrections</td>
<td>Correct for PB gridding before prediction of visibilities.</td>
</tr>
<tr>
<td>This should be True when doing deconvolution.</td>
<td>This should be False when predicting visibilities for model</td>
</tr>
<tr>
<td>sky with no primary beam attenuation in the model.</td>
<td>allowed: bool Default: true</td>
</tr>
<tr>
<td>cfcachedirname</td>
<td>Directory where convolution functions are to be (or are</td>
</tr>
<tr>
<td>being ) cached on the disk.</td>
<td>allowed: string Default:</td>
</tr>
<tr>
<td>rotpastep</td>
<td>The PA increment in degree used for on-the-fly (OTF)</td>
</tr>
<tr>
<td>rotation of the A-term in A-Projection.</td>
<td>allowed: double Default: 5.0</td>
</tr>
<tr>
<td>pastep</td>
<td>The PA increment in degree used to compute the PA-</td>
</tr>
<tr>
<td>rotated A-term in A-Projection.</td>
<td>allowed: double Default: 360.0</td>
</tr>
<tr>
<td>pblimit</td>
<td>Primary beam limit when using PBWProjection</td>
</tr>
<tr>
<td>allowed: double</td>
<td>Default:</td>
</tr>
</tbody>
</table>
Returns
bool

Example

- im.setoptions(cache=10000000, tile=32, gridfunction='BOX', location=me.location('vla'))

The above example is to tell imager to use memory to fit 10000000 complex numbers and tile the image with tiles of 32 pixels on a side. Also it tells imager to use a box function as gridding function. The location parameter will make imager override the position of the telescope to use (the default is the one it gets from the ms).

im.open('n1333.ms')
im.selectvis(fieldid=[2:6, 8:12], spwid=[1:2])
im.defineimage(nx=800, ny=800, cellx='0.5arcsec', celly='0.5arcsec', mode='velocity', nchan=30, mstart='-10km/s', mstep='1.8km/s', spwid=[1,2], fieldid=3)
im.setoptions(ftmachine='mosaic')
im.setvp(dovp=T)
im.setoptions(ftmachine='mosaic')
im.clean(algorithm='mfclark', model='try1', niter=200)

In the above example we are making a mosaic using the fields 2,3,4,5,6,8,9,10,11,12 and we use the mosaic ftmachine. This uses the primary beam of the telescope as the gridding function.

im.open('coma.ms')
im.selectvis(spwid=1, fieldid=1);
mydir=me.direction('J2000', '12h30m48', '12d24m0')
im.defineimage(nx=200, cellx='30arcsec', phasecenter=mydir);
im.make('outlier1');
im.defineimage(nx=1800, cellx='30arcsec');
im.setoptions(ftmachine='wproject', wprojplanes=512, padding=1.0)
im.make('main')
im.clean(algorithm='mfclark', model=['main', 'outlier1'], niter=10000, image=['coma.image', 'outlier1.image'])
im.done()
In the above example we are using the Wprojection algorithm for 3-D imaging. We are using 512 gridding functions. Sometimes if there is a memory issue (very large images and many gridding functions) we suggest the use of facetting of the image with wprojection. So the example above would be something like below. Note that when using facets only the \texttt{wfclark} and \texttt{wfhogbom} can be used for now. Note on how an outlier field (or flanking) field is set on an interfering source outside of the field of interest.

```python
im.open('coma.ms')

im.selectvis(spwid=1, fieldid=1);
mydir = me.direction('J2000', '12h30m48', '12d24m0')
im.defineimage(nx=200, ny=200, cellx='30arcsec', celly='30arcsec', phasecenter=mydir);
im.make('outlier1');
im.defineimage(nx=3000, ny=3000, cellx='30arcsec', celly='30arcsec', facets=3);
im.setoptions(ftmachine='wproject', wprojplanes=200, padding=1.2)
im.make('main')
im.clean(algorithm='wfclark', model=['main', 'outlier1'], niter=10000)
im.done()
```
imager.setscales.html

**imager.setscales - Function**

2.3.1 Set the scale sizes for MultiScale Clean

**Description**

The multiscale clean algorithm cleans an image on a number of different scales, decomposing the image into Gaussians of these scale sizes. This function allows the user to set the number of scales used (using the nscales method), or to directly control the sizes of the scales in pixels (using the uservector method). When using the nscales method, the scales are calculated using the following formula:

$$\theta_{\text{minor}} 10.0^{(i - N_{\text{scales}}/2)/2.0}$$

(2.3)

where $\theta_{\text{min}}$ is the fitted minor axis of the clean beam. The first value is zero.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Method by which scales are set</th>
</tr>
</thead>
<tbody>
<tr>
<td>scalemethod</td>
<td>Method by which scales are set</td>
</tr>
<tr>
<td>allowed: string</td>
<td>Default: nscales uservector nscales</td>
</tr>
<tr>
<td>nscales</td>
<td>Number of scales</td>
</tr>
<tr>
<td>allowed: int</td>
<td>Default: 5</td>
</tr>
<tr>
<td>uservector</td>
<td>Vector of scale sizes in pixels to use, defaults should be 0,3,10</td>
</tr>
<tr>
<td>allowed: doubleArray</td>
<td>Default: 0.0 3.0 10.0</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

1423
- `im.setscales(scalemethod='nscales', nscales=6);`

Here we make six scales automatically using the method described above. Or we could manually choose the scales in pixel numbers as follows:

- `im.setscales(scalemethod='uservector', uservector=[0,3,10,30]);`

Note: 0 pixel is the delta function, so if one were to select scale 0 only it would be equivalent to a Hogbom clean.
imager.setsmallscalebias.html

**imager.setsmallscalebias - Function**

2.3.1 Set bias toward smaller scales for MultiScale Clean

**Description**

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>small scale bias</th>
</tr>
</thead>
<tbody>
<tr>
<td>inbias</td>
<td>float</td>
</tr>
<tr>
<td>allowed</td>
<td>0.6</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```- im.setsmallscalebias(inbias=0.6);```
imager.settaylorterms - Function

2.3.1 Set the number of Taylor series terms for Multi-Frequency Clean

Description

The multi-frequency clean algorithm cleans an image by approximating its spectra by a Taylor series expansion. This function allows the user to set the number of Taylor terms to be used. Options are 1, 2, 3.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Number of Taylor terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntaylorterms</td>
<td>Number of Taylor terms</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Reference Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>reffreq</td>
<td>Reference Frequency</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Returns

bool

Example

- im.settaylorterms(ntaylorterms=3);
imager.setsdoptions.html

**imager.setsdoptions - Function**

### 2.3.1 Set some options for single dish processing

#### Description

Various less-often-used options for single dish processing can be set.

**scale** The overall scale of the single dish data is multiplied by this factor.

**weight** The weight given to the single dish data in the imaging is multiplied by this factor.

**convsupport** This parameter can be used to change the support used in gridding single dish data in imaging. If ‘PB’ or ‘pb’ is used as the 'convtype' in setoptions this parameter is ignored as the support is defined by the primary beam. The default of -1 mans 1 as convsupport is used for 'box' convolution function and 3 is used for 'SF' convolution function.

**pointingcolumntouse** This parameter is NOT to be changed under normal circumstances. This is to be used by those who know what they are doing and want to try to use different columns in the POINTING table especially if they believe their dish direction is wrong. And if any of the OFFSET columns is used do not expect to be able to use a different frame in the image setup in defineimage. Possible values are DIRECTION, TARGET, ENCODER, POINTING_OFFSET, SOURCE_OFFSET

**truncate** The truncation radius as a quantity or a float value. This parameter is effective only when 'GAUSS' or 'GJINC' is used as the 'convtype' in setoptions. You can use an unit 'pixel' to specify truncation radius as pixel value. If float value is set, or quantity without unit is set, its unit will be 'pixel'.

**gwidth** The width of the gaussian beam as a radius of half maximum. This parameter is effective only when 'GAUSS' or 'GJINC' is used as the 'convtype' in setoptions. You can use an unit 'pixel' to specify truncation radius as pixel value. If float value is set, or quantity without unit is set, its unit will be 'pixel'. Note that, when 'GJINC' is used as the 'convtype', gwidth doesn’t directry specify width of the convolution function.
The width of the jinc beam as a parameter \( c \), where \( \text{jinc} = \frac{\sin(\pi x/c)}{(\pi x/c)} \). This parameter is effective only when 'GJINC' is used as the 'convtype' in setoptions. You can use an unit 'pixel' to specify truncation radius as pixel value. If float value is set, or quantity without unit is set, its unit will be 'pixel'. Note that \( \text{jwidth} \) doesn't directly specify width of the convolution function.

**Arguments**

**Inputs**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>scale</td>
<td>Scaling applied to single dish data</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>1.0</td>
</tr>
<tr>
<td>weight</td>
<td>Weights applied to single dish data</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>1.0</td>
</tr>
<tr>
<td>convsupport</td>
<td>number of pixel for convolution support</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>-1</td>
</tr>
<tr>
<td>pointingcolumntouse</td>
<td>Which Pointing Table column to use to get direction</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>DIRECTION</td>
</tr>
<tr>
<td>truncate</td>
<td>truncation radius (effective only for 'GAUSS' or 'GJINC')</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant -1pixel</td>
</tr>
<tr>
<td>gwidth</td>
<td>radius of half maximum for gaussian (effective only for 'GAUSS' or 'GJINC')</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant -1pixel</td>
</tr>
<tr>
<td>jwidth</td>
<td>c-parameter for jinc function (effective only for 'GJINC')</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant -1pixel</td>
</tr>
<tr>
<td>minweight</td>
<td>Minimum weight level to use for weight correction and weight based masking.</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>0.</td>
</tr>
<tr>
<td>clipminmax</td>
<td>Clip minimum and maximum values from each grid</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

bool

1428
Example

- `im.setsdoptions(scale=1.0, weight=1.0, convsupport=5)`
Set the voltage pattern model for subsequent processing

**Description**

Set the voltage pattern model (and hence, the primary beam) used for a telescope. There are currently two ways to set the voltage pattern: by using the extensive list of defaults which the system knows about, or by creating a voltage pattern description with the vpmanager. The default voltage patterns include both a high and a low frequency VP for the WSRT, a VP for each observing band at the AT, several VP’s for the VLA, including the appropriate beam squint for each observing band, and Gaussian for the BIMA dishes. Due to temporary limitations in the internal structure of the visibility buffer, only one telescope’s voltage pattern can be applied to a particular MeasurementSet. This will be corrected shortly.

**Arguments**
### Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Allowed Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dovp</strong></td>
<td>Do voltage pattern (ie, primary beam) correction</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td><strong>usedefaultvp</strong></td>
<td>Look up the default VP for this telescope and frequency?</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td><strong>vptable</strong></td>
<td>If usedefaultvp is False, provide a VP Table made with vpmanager</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td><strong>dosquint</strong></td>
<td>Activate the beam squint in the VP model</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td><strong>parangleinc</strong></td>
<td>Parallactic angle increment for squint application</td>
<td>any</td>
<td>variant 360deg</td>
</tr>
<tr>
<td><strong>skyposthreshold</strong></td>
<td>Sky position threshold</td>
<td>any</td>
<td>variant 180deg</td>
</tr>
<tr>
<td><strong>telescope</strong></td>
<td>Which default telescope to use; if empty use the one in encoded in MS</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td><strong>verbose</strong></td>
<td>If false, suppress some messages from being sent to the logger.</td>
<td>bool</td>
<td>true</td>
</tr>
</tbody>
</table>

### Returns

bool

### Example

```python
im.setvp(dovp=True, usedefaultvp=True, dosquint=False)
```
imager.setweightgrid.html

**imager.setweightgrid - Function**

2.3.1 set the requested weight grids

**Description**

This is a utility function when running multi imager processes in parallel on subsection of an ms/data independently. One would wish to weight the dirty image before averaging or set the imaging weight density (when using uniform or Brigg’s style weighting) to account for all the data being used. This is **NOT** for the general user but for people who are parallelizing at the scripting level.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>weight</td>
<td>Numeric array. Required input.</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
<tr>
<td>type</td>
<td>Type of weight requested</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>imaging</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
wght=im.getweightgrid('imaging')
wght2=im2.getweightgrid('imaging')
wght=wght+wght2

im.setweightgrid(weight=wght, type='imaging')
im2.setweightgrid(weight=wght, type='imaging')
```
imager.smooth.html

**imager.smooth - Function**

2.3.1 Calculate an image smoothed with a Gaussian beam

**Description**

The model images are convolved with the specified Gaussian beam. By default (normalize=T), the beam volume is normalized to unity so that the smoothing is flux preserving. The smoothing used in restoration is not normalized.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of input model</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>stringArray</td>
</tr>
<tr>
<td>image</td>
<td>stringArray</td>
</tr>
<tr>
<td>usefit</td>
<td>bool</td>
</tr>
<tr>
<td>bmaj</td>
<td>any</td>
</tr>
<tr>
<td>bmin</td>
<td>any</td>
</tr>
<tr>
<td>bpa</td>
<td>any</td>
</tr>
<tr>
<td>normalize</td>
<td>variant 0deg</td>
</tr>
<tr>
<td>async</td>
<td>bool</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>async</td>
</tr>
</tbody>
</table>

1434
Example

- im.smooth(model='3C273XC1.clean', image='3C273XC1.clean.restored', bmaj='2.0arcsec', bmin='2.0arcsec')
imager.stop.html

**imager.stop** - Function

2.3.1 stop the currently executing function asap

**Description**

Stop the currently executing function as soon as possible. Note that it is not always possible to stop a function.

**Arguments**

**Returns**

bool
imager.summary.html

**imager.summary - Function**

2.3.1 Summarize the current state of the imager tool

**Description**

Writes a summary of the properties of the imager to the default logger. This includes:

- The name of the MeasurementSet (set in construction or via the open function).
- The parameters of the image (set via defineimage)
- The current beam (set by fitpsf or setbeam.
- The selection of an ms (set via selectvis)
- The general processing options (set via setoptions)

**Arguments**

**Returns**

bool

**Example**

- `im.open('3C273XC1.MS');`
- `im.defineimage(nx=256, ny=256)`
- `im.summary()`
imager.uvrange.html

**imager.uvrange - Function**

### 2.3.1 Select data within the limit of a given range

**Description**

Apply a uvrange so that only points within a given uvrange are selected for further usage. To be noted selectvis if used after uvrange will reset the selected range. So selectvis should be used prior to uvrange or can be used to reset it if one changes one’s mind. The points are not flagged! Further point to be noted for spectral line data the uv distance is calculated using the mean of the wavelengths of the different spectral channels selected.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>uvmin</td>
<td>Minimum uv distance allowed (wavelengths)</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>0.0</td>
</tr>
<tr>
<td>uvmax</td>
<td>Maximum uv distance allowed (wavelengths)</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```plaintext
im.weight('uniform')
im.uvrange(0, 4000.0)
```
imager.weight.html

**imager.weight - Function**

2.3.1 Apply additional weighting to the visibility weights

**Description**

Apply visibility weighting to correct for the local density of sampling in the uv plane. The imaging weights are calculated on the fly when processing the data and can be viewed by plotweights.

To correct for visibility sampling effects, natural, uniform, radial, and Briggs weighting are supported. These work as follows. Then:

- **natural**: minimizes the noise in the dirty image. The weight of the \( i \)-th sample is set to the inverse variance:

  \[
  w_i = \frac{1}{\sigma_i^2}
  \]  

  where \( \sigma_i \) is the noise of the \( i \)’th sample.

- **radial**: approximately minimizes rms sidelobes for an east-west synthesis array. The weight of the \( i \)-th sample is multiplied by the radial distance from the center of the \( u,v \) plane:

  \[
  w_i = w_i \sqrt{u_i^2 + v_i^2}
  \]  

- **uniform**: For Briggs and uniform weighting, we first grid the inverse variance \( w_i \) for all selected data onto a grid of size given by the argument npixels (default to nx) and u,v cell-size given by 2/fieldofview where fieldofview is the specified field of view (defaults to the image field of view). This forms the gridded weights \( W_j \). The weight of the \( i \)-th sample is then changed:

  \[
  w_i = \frac{w_i}{W_j}
  \]  

  where \( W_j \) is the gridded weight of the relevant cell. It may be shown that this minimizes rms sidelobes over the field of view. By changing the field of view, one may suppress the sidelobes over a region different (usually smaller) than the image size.

- **briggs**: \( rmode='norm' \): The weights are changed:

  \[
  w_i = \frac{w_i}{1 + W_j f^2}
  \]
where:

\[ f^2 = \frac{(5 \times 10^{-R})^2}{\sum_k W_k^2} \sum_i w_i^2 } \] (2.8)

and \( R \) is the robust parameter. The scaling of \( R \) is such that \( R = 0 \) gives a good tradeoff between resolution and sensitivity. \( R \) takes value between -2.0 (close to uniform weighting) to 2.0 (close to natural).

**briggs: rmode='abs'** : The weights are changed:

\[ w_{\iota} = \frac{w_{\iota}}{W_k R^2 + 2 \sigma_R^2} \] (2.9)

where \( R \) is the robust parameter and \( \sigma_R \) is the noise parameter.

For more details about Briggs (aka robust) weighting, see the [Briggs thesis](#).

Note that this weighting is *not* cumulative since the imaging weights are calculated from the specified weight (function of noise; usually \( 1/\sigma^2 \)) per visibility (actually stored in the WEIGHT column).

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Type of weighting</td>
<td>string</td>
<td>natural</td>
</tr>
<tr>
<td>rmode</td>
<td>Mode of briggs weighting</td>
<td>string</td>
<td>norm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>abs</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>noise</td>
<td>Noise used in absolute briggs weighting</td>
<td>any</td>
<td>variant 0.0Jy</td>
</tr>
<tr>
<td>robust</td>
<td>Parameter in briggs weighting</td>
<td>double</td>
<td>0.0</td>
</tr>
<tr>
<td>fieldofview</td>
<td>Field of view for uniform weighting</td>
<td>any</td>
<td>variant 0.0arcsec</td>
</tr>
<tr>
<td>npixels</td>
<td>Number of pixels in the u and v directions</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>mosaic</td>
<td>Individually weight the fields of a mosaic</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

| Returns                | bool                                                                       |                                  |                                  |

| Example                | im.weight(type='briggs', rmode='norm', robust=0.5)                         |                                  |                                  |

Applies Briggs (robust) weighting.
imager.mapextent.html

**imager.mapextent - Function**

2.3.1 Compute map extent from given set of MSs

**Description**

TODO: description must be filled

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ref</td>
<td>direction reference</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: J2000</td>
</tr>
<tr>
<td>movingsource</td>
<td>moving source name</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>pointingcolumntouse</td>
<td>POINTING table column to be used for computation</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: DIRECTION</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

TODO: example must be filled.
2.3.2 vpmanager - Tool

Tool for specifying voltage patterns and primary beams

Requires:

Synopsis

Description

The vpmanager tool serves to set up a list of primary beams or voltage patterns (antenna responses) and then select in detail which of them is used for which observatory. The distinction of several antenna types for a given observatory (heterogeneous arrays) is supported.

Antenna responses can be selected from either internally hard-coded ones, or response-groups defined via an AntennaResponses table, or user-defined analytic primary beams.

Imaging and simulation routines pick up the selected response definitions and instantiate them.

The vpmanager can also create a table with the description of one or more voltage patterns (vp) or primary beams (pb). There is a mapping between telescope name and the vp or pb description. The vp description table can be read by imager’s setvp method, which instantiates the corresponding voltage patterns from the descriptions and applies them to the images.

The vpmanager tool is the CASA Python object which constitutes the user interface to the VPManager C++ class. By default it is named ”vp” in casapy. The VPManager class is implemented as a singleton, i.e. internally there is only one instance at all times. This instance accessed via the static VPManager::Instance() method. It is permanent until casapy is exited and can be reinitialised via the VPManager::reset() method.

The vp tool connects to the single instance of VPManager. All settings the user makes with the tool, have effect immediately and are then used by all parts of CASA which access the VPManager class (i.e. eventually all imaging and simulation routines).

The VPManager instance keeps a simple database of available antenna responses, the vplist. This list is initialized at the startup of CASA or by calling the reset() method of the class. In the vp tool, the reset call can be triggered using

   vp.reset()

In order to support heterogeneous interferometer arrays, VPManager permits the use of antenna types in addition to observatory or telescope names.
For defining a simple response which is only spatially scaled by frequency but otherwise constant, a simple call to the vp tool is sufficient, e.g.:

```
vp.setpbairy(telescope='ALMA',
            dishdiam='12.0m',
            blockagediam='0.75m',
            maxrad='1.784deg',
            reffreq='1.0GHz',
            dopb=True)
```

This will create a new entry in the vplist for an analytic Airy disk antenna response and make it the default response for telescope "ALMA". Subsequent requests to VPManager for a ALMA antenna response will get this Airy disk. If whole response systems are to be defined for a given telescope, the use of an `AntennaResponses table` is possible. Such a table can be set up using the vp tool method `createantresp()` and then connected to a telescope using a command like

```
vp.setpbantresptable(telescope='ALMA',
                     antresppath=casa['dirs']['data']+'/alma/responses/AntennaResponses-ALMA-RT',
                     dopb=True)
```

where the value of the `antresppath` parameter indicates the path to the AntennaResponses table. Subsequent requests for ALMA antenna responses to VPManager will start a search in the indicated AntennaResponses table for responses matching given parameters. Presently supported search parameters in VPManager::getvp() and vp.getvp() are:

- antenna type
- observation time (used for versioning and for reference frame transformations)
- frequency (as a Measure, the reference frame is respected)
- observing direction (to support elevation and azimuth dependent responses)

An example of a call to vp.getvp() is

```
myrecord = vp.getvp(telescope='ALMA',
                    antennatype = 'DV',
                    obstime = '2009/07/24/10:00:00',
                    freq = 'TOPO 100GHz',
                    obsdirection = 'AZEL 30deg 60deg')
```

If the default antenna response for the given telescope is not defined via an AntennaResponses table, the observation parameters `obstime`, `freq`, and `obsdirection` are not needed and can be omitted. The parameter `antennatype` defaults to empty string. So if no antenna types are distinguished for the given telescope, the simplest call to getvp becomes
myrecord = vp.getvp(telescope='HATCREEK')

During initialization, VPManager will look for entries in the column "AntennaResponses" of the CASA "Observatories" table. If there are non-blank entries, the string found will be interpreted as the path to the default AntennaResponses table for the given telescope.

Note that the casacore AntennaResponses C++ class (which is used by VPManager to administrate the AntennaResponses tables) also supports the additional search parameters "receiver type" and "beam number". A general interface to the response file name search is available through the vp.getrespimagename() method. But presently this accesses only AntennaResponse tables which are entered as the default table in the Observatories table.

Generally, the vp tool methods provide functionality: to set up new analytic antenna responses, select which antenna responses from the vplist to use for which telescope and antenna type, access the contents of the vplist, create and access an AntennaResponses table, create a voltage pattern table.

Methods

- **vpmanager**: Construct a vpmanager tool (note: the underlying VPManager is a singleton)
- **saveastable**: Save the vp or pb descriptions as a table
- **loadfromtable**: Load the vp or pb descriptions from a table (deleting all previous definitions)
- **summarizevps**: Summarize the currently accumulated VP descriptions
- **setcannedpb**: Select a vp/pb from our library of common pb models
- **setpbairy**: Make an airy disk vp
- **setpbcospoly**: Make a vp/pb from a polynomial of scaled cosines
- **setpbgauss**: Make a Gaussian vp/pb
- **setphinvpoly**: Make a vp/pb as an inverse polynomial
- **setphnumeric**: Make a vp/pb from a user-supplied vector
- **setphimage**: Make a vp/pb from a user-supplied image
- **setphpoly**: Make a vp/pb from a polynomial
- **setpbantrantable**: Declare a reference to an antenna responses table
- **reset**: Reinitialize the VPManager (will erase all VPs and defaults defined on the command line)
- **setuserdefault**: Select the VP which is to be used by the imager for the given telescope and antenna type
- **getuserdefault**: Get the vp list number of the present default VP/PB for the given parameters (-1 = internal PB, -2 = none)
- **getanttypes**: Return the list of available antenna types for the given parameters
- **numvps**: Return the number of vps/pbs available for the given parameters
- **getvp**: Return the default vps/pbs record for the given parameters
- **getvps**: Return the default vps/pbs records for the given antenna list as subrecords in one record
- **createantresp**: Create a standard-format AntennaResponses table
- **getrespimagename**: Get the image name for the given parameters from the given responses table
vpmanager.vpmanager.html

**vpmanager.vpmanager - Function**

**2.3.2** Construct a vpmanager tool (note: the underlying VPManager is a singleton)

**Description**

The vpmanager constructor has no arguments.

**Arguments**
vpmanager.saveastable.html

vpmanager.saveastable - Function

2.3.2 Save the vp or pb descriptions as a table

Description

Save the vp or pb descriptions as a table. Each description is in a different row of the table.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablename</td>
<td>Name of table to save vp descriptions in</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

bool
vpmanager.loadfromtable.html

**vpmanager.loadfromtable - Function**

2.3.2 Load the vp or pb descriptions from a table (deleting all previous definitions)

**Description**

Load the vp or pb descriptions from a table created, e.g., with saveastable().

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablename</td>
<td>Name of table to load vp descriptions from</td>
</tr>
<tr>
<td>allowed</td>
<td>allowed: string</td>
</tr>
<tr>
<td>Default</td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

bool
vpmanager.summarizevps.html

vpmanager.summarizevps - Function

2.3.2 Summarize the currently accumulated VP descriptions

Description

Summarize the currently accumulated VP descriptions to the logger.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>verbose</td>
<td>Print out full record? Other</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td>wise, print summary.</td>
<td></td>
</tr>
<tr>
<td>allowed</td>
<td>bool</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool
vpmanager.setcannedpb.html

vpmanager.setcannedpb - Function

2.3.2 Select a vp/pb from our library of common pb models

Description

We have many vp/pb models ready to go for a variety of telescopes. If 'DEFAULT' is selected, the system default for that telescope and frequency is used.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>telescope</td>
<td>Which telescope in the MS will use this vp/pb?</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>VLA</td>
</tr>
<tr>
<td>othertelescope</td>
<td>If telescope==&quot;OTHER&quot;, specify name here</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>dopb</td>
<td>Should we apply the vp/pb to this telescope’s data?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>true</td>
</tr>
<tr>
<td>commonpb</td>
<td>List of common vp/pb models: DEFAULT code figures it out</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>dosquint</td>
<td>Enable the natural beam squint found in the common vp model</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>paincrement</td>
<td>Increment in Parallactic Angle for asymmetric (ie, squinted) vp application</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant 720deg</td>
</tr>
<tr>
<td>usesymmetricbeam</td>
<td>Not currently used</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>

Returns

1450
record
vpmanager.setpairy.html

vpmanager.setpairy - Function

2.3.2 Make an airy disk vp

Description

Information sufficient to create a portion of the Airy disk voltage pattern. The Airy disk pattern is formed by Fourier transforming a uniformly illuminated aperture and is given by

\[ vp_p(i) = \frac{areaRatio \cdot 2.0 \cdot j_1(x) / x - 2.0 \cdot j_1(x \cdot lengthRatio) / (x \cdot lengthRatio)}{areaNorm}, \]

(2.10)

where areaRatio is the dish area divided by the blockage area, lengthRatio is the dish diameter divided by the blockage diameter, and

\[ x = \frac{i \cdot maxrad \cdot 7.016 \cdot dishdiam / 24.5m}{N_{samples} \cdot 1.566 \cdot 60}. \]

(2.11)

Arguments
## Inputs

**telescope**  
Which telescope in the MS will use this \text{vp/pb}?
- **allowed**: string
- **Default**: VLA

**othertelescope**  
If \text{telescope}="\text{OTHER}\text{"}, specify name here
- **allowed**: string
- **Default**: 

**dopb**  
Should we apply the \text{vp/pb} to this telescope’s data?
- **allowed**: bool
- **Default**: true

**dishdiam**  
Effective diameter of dish
- **allowed**: any
- **Default**: variant 25.0m

**blockagediam**  
Effective diameter of subreflector blockage
- **allowed**: any
- **Default**: variant 2.5m

**maxrad**  
Maximum radial extent of the \text{vp/pb} (scales with 1/freq)
- **allowed**: any
- **Default**: variant 0.8deg

**reffreq**  
Frequency at which \text{maxrad} is specified
- **allowed**: any
- **Default**: variant 1.0GHz

**squintdir**  
Offset (Measure) of RR beam from pointing center, azel frame (scales with 1/freq)
- **allowed**: any
- **Default**: variant

**squintreffreq**  
Frequency at which the \text{squint} is specified
- **allowed**: any
- **Default**: variant 1.0GHz

**dosquint**  
Enable the natural beam \text{squint} found in the common \text{vp} model
- **allowed**: bool
- **Default**: false

**paincrement**  
Increment in Parallactic Angle for asymmetric (ie, squinted) \text{vp} application
- **allowed**: any
- **Default**: variant 720deg

**usesymmetricbeam**  
Not currently used
- **allowed**: bool
- **Default**: false

## Returns

**record**
vpmanager.setpbcospoly.html

vpmanager.setpbcospoly - Function

2.3.2 Make a vp/pb from a polynomial of scaled cosines

Description

A voltage pattern or primary beam of the form

\[ VP(x) = \sum_i (coeff_i \cos(2^i(scale_i,x)). \]  \hspace{1cm} (2.12)

This is a generalization of the WSRT primary beam model.

Arguments
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>telescope</td>
<td>Which telescope in the MS will use this vp/pb?</td>
<td>string</td>
<td>VLA</td>
</tr>
<tr>
<td>othertelescope</td>
<td>If telescope==&quot;OTHER&quot;, specify name here</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>dopb</td>
<td>Should we apply the vp/pb to this telescope’s data?</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>coeff</td>
<td>Vector of coefficients of cosines</td>
<td>doubleArray</td>
<td>-1</td>
</tr>
<tr>
<td>scale</td>
<td>Vector of scale factors of cosines</td>
<td>doubleArray</td>
<td>-1</td>
</tr>
<tr>
<td>maxrad</td>
<td>Maximum radial extent of the vp/pb (scales with 1/freq)</td>
<td>any</td>
<td>variant 0.8deg</td>
</tr>
<tr>
<td>reffreq</td>
<td>Frequency at which maxrad is specified</td>
<td>any</td>
<td>variant 1.0GHz</td>
</tr>
<tr>
<td>isthispb</td>
<td>Do these parameters describe a PB or a VP?</td>
<td>string</td>
<td>PB</td>
</tr>
<tr>
<td>squintdir</td>
<td>Offset (Measure) of RR beam from pointing center, azel frame (scales with 1/freq)</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>squintreffreq</td>
<td>Frequency at which the squint is specified</td>
<td>any</td>
<td>variant 1.0GHz</td>
</tr>
<tr>
<td>dosquint</td>
<td>Enable the natural beam squint found in the common vp model</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>paincrement</td>
<td>Increment in Parallactic Angle for asymmetric (ie, squinted) vp application</td>
<td>any</td>
<td>variant 720deg</td>
</tr>
<tr>
<td>usesymmetricbeam</td>
<td>Not currently used</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>
Returns
record
vpmanager.setpbgauss.html

vpmanager.setpbgauss - Function

2.3.2 Make a Gaussian vp/pb

Description

Make a Gaussian primary beam given by

\[ PB(x) = e^{-\left(x/(\text{halfwidth} \ast \sqrt{1/\log(2)})\right)}, \]  \hspace{1cm} (2.13)

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>telescope</td>
<td>Which telescope in the MS will use this vp/pb?</td>
</tr>
<tr>
<td>allowed: string</td>
<td>Default: VLA</td>
</tr>
<tr>
<td>othertelescope</td>
<td>If telescope==”OTHER”, specify name here</td>
</tr>
<tr>
<td>allowed: string</td>
<td>Default:</td>
</tr>
<tr>
<td>dopb</td>
<td>Should we apply the vp/pb to this telescope’s data?</td>
</tr>
<tr>
<td>allowed: bool</td>
<td>Default: true</td>
</tr>
<tr>
<td>halfwidth</td>
<td>Half power half width of the Gaussian at the reffreq</td>
</tr>
<tr>
<td>allowed: any</td>
<td>Default: variant 0.5deg</td>
</tr>
<tr>
<td>maxrad</td>
<td>Maximum radial extent of the vp/pb (scales with 1/freq)</td>
</tr>
<tr>
<td>allowed: any</td>
<td>Default: variant 1.0deg</td>
</tr>
<tr>
<td>reffreq</td>
<td>Frequency at which maxrad is specified</td>
</tr>
<tr>
<td>allowed: any</td>
<td>Default: variant 1.0GHz</td>
</tr>
<tr>
<td>isthispb</td>
<td>Do these parameters describe a PB or a VP?</td>
</tr>
<tr>
<td>allowed: string</td>
<td>Default: PB</td>
</tr>
<tr>
<td>squintdir</td>
<td>Offset (Measure) of RR beam from pointing center, azel frame (scales with 1/freq)</td>
</tr>
<tr>
<td>allowed: any</td>
<td>Default: variant</td>
</tr>
<tr>
<td>squintreffreq</td>
<td>Frequency at which the squint is specified</td>
</tr>
<tr>
<td>allowed: any</td>
<td>Default: variant 1.0GHz</td>
</tr>
<tr>
<td>dosquint</td>
<td>Enable the natural beam squint found in the common vp model</td>
</tr>
<tr>
<td>allowed: bool</td>
<td>Default: false</td>
</tr>
<tr>
<td>paincrement</td>
<td>Increment in Parallactic Angle for asymmetric (ie, squinted) vp application</td>
</tr>
<tr>
<td>allowed: any</td>
<td>Default: variant 720deg</td>
</tr>
<tr>
<td>usesymmetricbeam</td>
<td>Not currently used</td>
</tr>
<tr>
<td>allowed: bool</td>
<td>Default: false</td>
</tr>
</tbody>
</table>

| Returns                     | record                                                                      |

1459
vpmanager.setpbinvpoly.html

**vpmanager.setpbinvpoly - Function**

2.3.2 Make a vp/pb as an inverse polynomial

**Description**

The inverse polynomial describes the inverse of the VP or PB as a polynomial of even powers:

\[
1/VP(x) = \sum_i \text{coeff}_i * x^{2i}.
\]

(2.14)

**Arguments**
Inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>telescope</td>
<td>Which telescope in the MS will use this vp/pb?</td>
<td>string</td>
<td>VLA</td>
</tr>
<tr>
<td>othertelescope</td>
<td>If telescope==&quot;OTHER&quot;, specify name here</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>dopb</td>
<td>Should we apply the vp/pb to this telescope's data?</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>coeff</td>
<td>Coefficients of even powered terms</td>
<td>doubleArray</td>
<td>-1</td>
</tr>
<tr>
<td>maxrad</td>
<td>Maximum radial extent of the vp/pb (scales with 1/freq)</td>
<td>any</td>
<td>variant 0.8deg</td>
</tr>
<tr>
<td>reffreq</td>
<td>Frequency at which maxrad is specified</td>
<td>any</td>
<td>variant 1.0GHz</td>
</tr>
<tr>
<td>isthispb</td>
<td>Do these parameters describe a PB or a VP?</td>
<td>string</td>
<td>PB</td>
</tr>
<tr>
<td>squintdir</td>
<td>Offset (Measure) of RR beam from pointing center, azel frame (scales with 1/freq)</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>squintreffreq</td>
<td>Frequency at which the squint is specified</td>
<td>any</td>
<td>variant 1.0</td>
</tr>
<tr>
<td>dosquint</td>
<td>Enable the natural beam squint found in the common vp model</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>paincrement</td>
<td>Increment in Parallactic Angle for asymmetric (ie, squinted) vp application</td>
<td>any</td>
<td>variant 720deg</td>
</tr>
<tr>
<td>usesymmetricbeam</td>
<td>Not currently used</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

Returns

record
vpmanager.setpbnumeric.html

vpmanager.setpbnumeric - Function

2.3.2 Make a vp/pb from a user-supplied vector

Description

Supply a vector of vp/pb sample values taken on a regular grid between x=0 and x=maxrad. We perform sinc interpolation to fill in the lookup table.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>telescope</td>
<td>Which telescope in the MS will use this vp/pb?</td>
<td>string</td>
<td>VLA</td>
</tr>
<tr>
<td>othertelescope</td>
<td>If telescope==”OTHER”, specify name here</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>dopb</td>
<td>Should we apply the vp/pb to this telescope’s data?</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>vect</td>
<td>Vector of vp/pb samples uniformly spaced from 0 to maxrad</td>
<td>doubleArray</td>
<td>-1</td>
</tr>
<tr>
<td>maxrad</td>
<td>Maximum radial extent of the vp/pb (scales with 1/freq)</td>
<td>any</td>
<td>variant 0.8deg</td>
</tr>
<tr>
<td>reffreq</td>
<td>Frequency at which maxrad is specified</td>
<td>any</td>
<td>variant 1.0GHz</td>
</tr>
<tr>
<td>isthispb</td>
<td>Do these parameters describe a PB or a VP?</td>
<td>string</td>
<td>PB</td>
</tr>
<tr>
<td>squintdir</td>
<td>Offset (Measure) of RR beam from pointing center, azel frame (scales with 1/freq)</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>squintreffreq</td>
<td>Frequency at which the squint is specified</td>
<td>any</td>
<td>variant 1.0GHz</td>
</tr>
<tr>
<td>dosquint</td>
<td>Enable the natural beam squint found in the common vp model</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>paincrement</td>
<td>Increment in Parallactic Angle for asymmetric (ie, squinted) vp application</td>
<td>any</td>
<td>variant 720deg</td>
</tr>
<tr>
<td>usesymmetricbeam</td>
<td>Not currently used</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

record
vpmanager.setpbimage - Function

2.3.2 Make a vp/pb from a user-supplied image

Description

Experimental: Supply an image of the E Jones elements. The format of the image is:

**Shape** nx by ny by 4 complex polarizations (RR, RL, LR, LL or XX, XY, YX, YY) by 1 channel.

**Direction coordinate** Az, El

**Stokes coordinate** All four “stokes” parameters must be present in the sequence RR, RL, LR, LL or XX, XY, YX, YY.

**Frequency** Only one channel is currently needed - frequency dependence beyond that is ignored.

If a complex image is specified the real and imaginary images is to be left empty.

The other option is to provide the real and imaginary part of the E-Jones as separate float images. On that case one or two images may be specified - the real (must be present) and imaginary parts (optional).

Note that beamsquint must be intrinsic to the images themselves. This will be accounted for correctly by regridding of the images from Az-El to Ra-Dec according to the parallactic angle.

antnames is the Vector of names for which this response pattern apply '*' is for all. The name has to match exactly the name of the Antennas in the ANTENNA table of the MS with which you want to use this VPManager table or object.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>telescope</td>
<td>Which telescope in the MS will use this vp/pb?</td>
<td>string</td>
<td>VLA</td>
</tr>
<tr>
<td>othertelescope</td>
<td>If telescope==&quot;OTHER&quot;, specify name here</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>dopb</td>
<td>Should we apply the vp/pb to this telescope's data?</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>realimage</td>
<td>Real part of vp as an image</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>imagimage</td>
<td>Imaginary part of vp as an image</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>compleximage</td>
<td>complex vp as an image of complex numbers; if specified realimage and imagimage are ignored</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>antnames</td>
<td>antenna names for which this pattern is valid; default is all antennas</td>
<td>stringArray</td>
<td>*</td>
</tr>
</tbody>
</table>

Returns

record
vpmanager.setpbpoly.html

**vpmanager.setpbpoly - Function**

### 2.3.2 Make a vp/pb from a polynomial

#### Description

The VP or PB is described as a polynomial of even powers:

\[ VP(x) = \sum_i coe f_i \cdot x^{2i}. \]  \hspace{1cm} (2.15)

#### Arguments
## Inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>telescope</td>
<td>Which telescope in the MS will use this vp/pb?</td>
<td>string</td>
<td>VLA</td>
</tr>
<tr>
<td>othertelescope</td>
<td>If telescope==&quot;OTHER&quot;, specify name here</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>dopb</td>
<td>Should we apply the vp/pb to this telescope’s data?</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>coeff</td>
<td>Coefficients of even powered terms</td>
<td>doubleArray</td>
<td>-1</td>
</tr>
<tr>
<td>maxrad</td>
<td>Maximum radial extent of the vp/pb (scales with 1/freq)</td>
<td>any</td>
<td>variant 0.8deg</td>
</tr>
<tr>
<td>reffreq</td>
<td>Frequency at which maxrad is specified</td>
<td>any</td>
<td>variant 1.0GHz</td>
</tr>
<tr>
<td>isthispb</td>
<td>Do these parameters describe a PB or a VP?</td>
<td>string</td>
<td>PB</td>
</tr>
<tr>
<td>squintdir</td>
<td>Offset (Measure) of RR beam from pointing center, azel frame (scales with 1/freq)</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>squintreffreq</td>
<td>Frequency at which the squint is specified</td>
<td>any</td>
<td>variant 1.0GHz</td>
</tr>
<tr>
<td>dosquint</td>
<td>Enable the natural beam squint found in the common vp model</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>pmincrement</td>
<td>Increment in Parallactic Angle for asymmetric (ie, squinted) vp application</td>
<td>any</td>
<td>variant 720</td>
</tr>
<tr>
<td>usesymmetricbeam</td>
<td>Not currently used</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

## Returns

record

1470
Declare a reference to an antenna responses table containing a set of VP/PB definitions.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>telescope</td>
<td>Which telescope in the MS will use this vp/pb?</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>othertelescope</td>
<td>If telescope==&quot;OTHER&quot;, specify name here</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>dopb</td>
<td>Should we apply the vp/pb to this telescope’s data?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>true</td>
</tr>
<tr>
<td>antresppath</td>
<td>The path to the antenna responses table (absolute or relative to CASA data dir.)</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool
vpmanager.reset.html

vpmanager.reset - Function

2.3.2 Reinitialize the VPManager (will erase all VPs and defaults defined on the command line)

Description

Reinitialize the VPManager database. Erase all VPs and defaults defined on the command line.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
</table>

Returns

bool
vpmanager.setuserdefault.html

**vpmanager.setuserdefault - Function**

2.3.2 Select the VP which is to be used by the imager for the given telescope and antenna type

**Description**

Selects the VP which is to be used by the imager for the given telescope and antenna type. Overwrites a previous default. Returns True if successful.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>vplistnum</td>
<td>The number of the vp as displayed by summarizevps(), or -1 for internal default, or -2 for unset</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>telescope</td>
<td>Which telescope in the MS will use this vp/pb?</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>antennatypenew</td>
<td>Which antennatypenew will use this vp/pb?</td>
<td>all</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool
vpmanager.getuserdefault.html

vpmanager.getuserdefault - Function

2.3.2 Get the vp list number of the present default VP/PB for the given parameters (-1 = internal PB, -2 = none)

Description

Get the vp list number of the present default VP/PB for the given parameters.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>telescope</td>
<td>Which telescope in the MS will use this vp/pb?</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>anttype</td>
<td>Which antennatype will use this vp/pb? Default: &quot;&quot; = all</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Returns

int
vpmanager.getanttypes.html

**vpmanager.getanttypes - Function**

2.3.2 Return the list of available antenna types for the given parameters

**Description**

Get a list of the available antenna types.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>telescope</td>
<td>Telescope name</td>
<td>string</td>
<td>variant</td>
</tr>
<tr>
<td>obstime</td>
<td>Time of the observation (for versioning and reference frame calculations)</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>freq</td>
<td>Frequency of the observation (may include reference frame, default: LSRK)</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>obsdirection</td>
<td>Direction of the observation (may include reference frame, default: J2000). default: Zenith</td>
<td>any</td>
<td>variant AZEL 0deg 90deg</td>
</tr>
</tbody>
</table>

**Returns**

stringArray
vpmanager.numvps.html

vpmanager.numvps - Function

2.3.2 Return the number of vps/pbs available for the given parameters

Description

Can be used to, e.g., determine the number of antenna types. Note: if a global response is defined for the telescope, this will increase the count of available vps/pbs by 1.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>telescope</td>
<td>Telescope name</td>
<td>string</td>
<td>variant</td>
</tr>
<tr>
<td>obstime</td>
<td>Time of the observation (for versioning and reference frame calculations)</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>freq</td>
<td>Frequency of the observation (may include reference frame, default: LSRK)</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>obsdirection</td>
<td>Direction of the observation (may include reference frame, default: J2000). default: Zenith</td>
<td>any</td>
<td>variant AZEL 0deg 90deg</td>
</tr>
</tbody>
</table>

Returns

int
vpmanager.getvp.html

vpmanager.getvp - Function

2.3.2 Return the default vps/pbs record for the given parameters

Description

Record is empty if no matching vp/pb could be found.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>telescope</td>
<td>Telescope name</td>
<td>string</td>
</tr>
<tr>
<td>antennatype</td>
<td>The antenna type as a string, e.g. &quot;DV&quot;</td>
<td>string</td>
</tr>
<tr>
<td>obstime</td>
<td>Time of the observation (for versioning and reference frame calculations), e.g. 2011/12/12T00:00:00</td>
<td>any</td>
</tr>
<tr>
<td>freq</td>
<td>Frequency of the observation (may include reference frame, default: LSRK)</td>
<td>any</td>
</tr>
<tr>
<td>obsdirection</td>
<td>Direction of the observation (may include reference frame, default: J2000), default:</td>
<td>any</td>
</tr>
</tbody>
</table>

Returns

record
vpmanager.getvps.html

vpmanager.getvps - Function

2.3.2 Return the default vps/pbs records for the given antenna list as subrecords in one record

Description

Record is empty if no matching vp/pb could be found.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>telescope</td>
<td>Telescope name</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>antennas</td>
<td>The antenna names, e.g., as taken from the MS AN-TENNA table</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>obstimestart</td>
<td>Time of the start of the observation (for versioning and</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant 1970/01/01T00:00:00</td>
</tr>
<tr>
<td>obstimeend</td>
<td>Time of the end of the observation (for versioning and</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant 1970/01/01T00:00:00</td>
</tr>
<tr>
<td>minfreq</td>
<td>Minimum frequency of the observation (may include reference frame, default:</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant 1GHz</td>
</tr>
<tr>
<td>maxfreq</td>
<td>Maximum frequency of the observation (may include reference frame, default:</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant 1GHz</td>
</tr>
<tr>
<td>obsdirection</td>
<td>Direction of the observation (may include reference frame, default: J2000),</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant AZEL 0deg 90deg</td>
</tr>
</tbody>
</table>
Returns
record
vpmanager.createantresp.html

vpmanager.createantresp - Function

2.3.2 Create a standard-format AntennaResponses table

Description

The AntennaResponses table serves CASA to look up the location of images describing the response of observatory antennas. Three types of images are supported: "VP" - real voltage patterns, "AIF" - complex aperture illumination patterns, "EFP" - complex electric field patterns. For each image, a validity range can be defined in Azimuth, Elevation, and Frequency. Furthermore, an antenna type (for heterogeneous arrays), a receiver type (for the case of several receivers on the same antenna having overlapping frequency bands), and a beam number (for the case of multiple beams per antenna) are associated with each response image.

The images need to be stored in a single directory DIR of arbitrary name and need to have file names following the pattern

obsname_beamnum_anttype_rectype_azmin_aznom_azmax_elmin_elnom_elmax_freqmin_freqnom_freqmax_frequnit_comment_functype.im

where the individual name elements mean the following (none of the elements may contain the space character, but they may be empty strings if they are not numerical values):

obsname - name of the observatory as in the Observatories table, e.g. "ALMA"

beamnum - the numerical beam number (integer) for the case of multiple beams, e.g. 0

anttype - name of the antenna type, e.g. "DV"

rectype - name of the receiver type, e.g. ""

azmin, aznom, azmax - numerical value (degrees) of the minimal, the nominal, and the maximal Azimuth where this response is valid, e.g. "-10.5,0.,10.5"

elmin, elnom, elmax - numerical value (degrees) of the minimal, the nominal, and the maximal Elevation where this response is valid, e.g. "10.,45.,80."

freqmin, freqnom, freqmax - numerical value (degrees) of the minimal, the nominal, and the maximal Frequency (in units of frequnit) where this response is valid, e.g. "84.,100.,116."
frequnit - the unit of the previous three frequencies, e.g. "GHz"

comment - any string containing only characters permitted in file names and not empty space

functype - the type of the image as defined above ("VP", "AIF", or "EFP")

The createantresp method will then extract the parameters from all the images in DIR and create the lookup table in the same directory.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Path to the directory containing the response images</th>
</tr>
</thead>
<tbody>
<tr>
<td>imdir</td>
<td>Path to the directory containing the response images</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Time from which onwards the response is valid, format YYYY/MM/DD/hh:mm:ss</th>
</tr>
</thead>
<tbody>
<tr>
<td>starttime</td>
<td>Time from which onwards the response is valid, format YYYY/MM/DD/hh:mm:ss</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inputs</th>
<th>List containing the names of the observatory’s frequency bands</th>
</tr>
</thead>
<tbody>
<tr>
<td>bandnames</td>
<td>List containing the names of the observatory’s frequency bands</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inputs</th>
<th>List containing the lower edges of the observatory’s frequency bands, e.g. [&quot;80GHz&quot;,&quot;120GHz&quot;]</th>
</tr>
</thead>
<tbody>
<tr>
<td>bandminfreq</td>
<td>List containing the lower edges of the observatory’s frequency bands, e.g. [&quot;80GHz&quot;,&quot;120GHz&quot;]</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inputs</th>
<th>List containing the upper edges of the observatory’s frequency bands, e.g. [&quot;120GHz&quot;,&quot;180GHz&quot;]</th>
</tr>
</thead>
<tbody>
<tr>
<td>bandmaxfreq</td>
<td>List containing the upper edges of the observatory’s frequency bands, e.g. [&quot;120GHz&quot;,&quot;180GHz&quot;]</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool
vpmanager.getrespimagename.html

**vpmanager.getrespimagename - Function**

2.3.2 Get the image name for the given parameters from the given responses table

**Description**

Given the observatory name, the antenna type, the receiver type, the observing frequency, the observing direction, and the beam number, find the applicable response image and return its name.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>telescope</td>
<td>Which telescope is described by this response?</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>starttime</td>
<td>Time at which the response has to be valid, format YYYY/MM/DD/hh:mm:ss</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>frequency</td>
<td>The frequency at which the response has to be valid, e.g. &quot;100GHz&quot;</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>functype</td>
<td>Type of the responsefunction requested, e.g. &quot;EFP&quot;</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>ANY</td>
</tr>
<tr>
<td>anttype</td>
<td>Antenna type (observatory-dependent)</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>azimuth</td>
<td>Azimuth of the observation (at the location of the observatory, 0 is North), e.g. &quot;5deg&quot;</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>0deg</td>
</tr>
<tr>
<td>elevation</td>
<td>Elevation of the observation (at the location of the observatory, 0 is North), e.g. &quot;60deg&quot;</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>45deg</td>
</tr>
<tr>
<td>rectype</td>
<td>Receiver type (observatory-dependent)</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>beamnumber</td>
<td>Beam number (for the case of multiple beams per receiver)</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

---

simulator-Module.html

1484
2.4 simulator - Module

Module for simulation of telescope data

Description

simulator provides a unified interface for simulation of telescope processing. It can create a MeasurementSet from scratch or read in an existing MeasurementSet. It can predict synthesis data onto the (u,v) coordinates or single dish data onto (ra,dec) points, and it can corrupt this data through Gaussian errors or through specified errors reading in (anti-) calibration tables.

In the observing phase, simulator tries to act like a (simple) telescope. You first open the name of the MeasurementSet that you wish to construct. Next you use the various set* methods to setup the observing (sources, spectral windows, etc). Each such setup should be given a unique name that will be used in the next step. Then you call the observe method for each observing scan you wish to make. Here you specify the source name, spectral window name, and observing times. After this, you have a MeasurementSet that is complete but empty. In the next phase, you fill the MeasurementSet with data from a model and then corrupt the measurements (if desired). To fill it in with a model, use the predict method. Finally, to apply errors, first set up the various effects using the relevant set* methods and then call corrupt.

Some important points:

- One call to observe generates one scan (all rows have the same SCAN_NUMBER).
- The start and stop times specified to observe need not be contiguous and so one can simulate antenna drive times.
- Currently there is no facility for patterns of observing, such as mosaicing, since it is easy to do this via sequences of calls of observe.

The following columns of the MeasurementSet are particularly important:

- **DATA** The original observed visibilities are in a column called DATA. These are normally not altered by any processing in CASA. simulator does write this column when it creates observations.
- **CORRECTED\_DATA** During a calibration process, the visibilities may be corrected for calibration effects. These corrected visibilities are stored in CORRECTED\_DATA which is created upon demand.
- **MODEL\_DATA** During various phases of processing, the visibilities as predicted from some model are required. These model visibilities are stored in the column MODEL\_DATA. The 'ft' task can be used to calculate the model visibility for a model image.

The available tool in this module is:

- **simulator** - tool for simulation
2.4.1 simulator - Tool

Tool for simulation

Requires:

Synopsis

Description

simulator provides a unified interface for simulation of telescope processing. It can create a MeasurementSet from scratch or read in an existing MeasurementSet, it can predict synthesis data onto the (u,v) coordinates or single dish data onto (ra,dec) points, and it can corrupt this data through Gaussian errors or through specific errors residing in (anti-) calibration tables.

In the observing phase, simulator tries to act like a (simple) telescope. You first make a simulator tool, with the name of the MeasurementSet that you wish to construct. Next you use the various set* methods to set up the observing (sources, spectral windows, etc.). Each such setup should be given a unique name that will be used in the next step. Then you call the observe method for each observing scan you wish to make. Here you specify the source name, spectral window name, and observing times. After this, you have a MeasurementSet that is complete but empty. In the next phase, you fill the MeasurementSet with data from a model and then corrupt the measurements (if desired). To fill it in with a model, use the predict method. NOTE: sm.predict assumes the model image units are Jy/pixel, and in fact will overwrite the brightness units of the image itself! Finally, to apply errors, first set up the various effects using the relevant set* methods, and then call corrupt.

Some important points (mostly for the cognoscenti):

- One call to observe generates one scan (all rows have the same SCAN_NUMBER).
- The start and stop times specified to observe need not be contiguous and so one can simulate antenna drive times.
- Currently there is no facility for patterns of observing, such as mosaicing, since it is easy to do this via sequences of calls of observe.
- The heavy duty columns (DATA, FLAG, IMAGING_WEIGHT, etc. are tiled. New tiles are generated for each scan. Thus the TSM files will not get very large.
simulator changes some columns to the MeasurementSet to store results of processing. The following columns in the MS are particularly important:

**DATA** The original observed visibilities are in a column called DATA. These are normally not altered by any processing in CASA. However, this simulation program does overwrite these values.

**CORRECTED_DATA** During a calibration process, as carried out by *e.g.* calibrator, the visibilities may be corrected for calibration effects. This corrected visibilities are stored in a column CORRECTED_DATA which is created on demand.

**MODEL_DATA** During various phases of processing, the visibilities as predicted from some model are required. These model visibilities are stored in a column MODEL_DATA. The ft function of the imager tool should be used to calculate the model visibility for a model image or component models.

Standard tools such as the table module and the ms can be used to access and possibly change these (and all other) columns.

simulator is a tool that performs simulation of synthesis data, including (optionally) creation of a MeasurementSet, prediction of model data, and corruption by various physical effects.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>simulator</td>
<td>Construct a simulator tool</td>
</tr>
<tr>
<td>open</td>
<td>Construct a simulator tool and creating a new MeasurementSet</td>
</tr>
<tr>
<td>openfromms</td>
<td>Construct a simulator tool using an already existing MS</td>
</tr>
<tr>
<td>close</td>
<td>Close the newsimulator tool</td>
</tr>
<tr>
<td>done</td>
<td>Close the newsimulator tool</td>
</tr>
<tr>
<td>name</td>
<td>Provide the name of the attached MeasurementSet</td>
</tr>
<tr>
<td>summary</td>
<td>Summarize the current state</td>
</tr>
<tr>
<td>type</td>
<td>Return the type of this tool</td>
</tr>
<tr>
<td>settimes</td>
<td>Set integration time, <em>etc.</em></td>
</tr>
<tr>
<td>observe</td>
<td>Observe a given configuration</td>
</tr>
<tr>
<td>observemany</td>
<td>Observe a given configuration</td>
</tr>
<tr>
<td>setlimits</td>
<td>Set limits for observing</td>
</tr>
<tr>
<td>setauto</td>
<td>Set autocorrelation weight</td>
</tr>
<tr>
<td>setconfig</td>
<td>Set the antenna configuration</td>
</tr>
<tr>
<td>setknownconfig</td>
<td>Set the antenna configuration to a known array</td>
</tr>
<tr>
<td>setfeed</td>
<td>Set the feed parameters</td>
</tr>
<tr>
<td>setfield</td>
<td>Set one or more observed fields</td>
</tr>
<tr>
<td>setmosaicfield</td>
<td>Set observed mosaic fields</td>
</tr>
<tr>
<td>setswindow</td>
<td>Set one or more spectral windows</td>
</tr>
<tr>
<td>setdata</td>
<td>Set the data parameters selection for subsequent processing</td>
</tr>
<tr>
<td>predict</td>
<td>Predict astronomical data from an image</td>
</tr>
</tbody>
</table>

1487
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setoptions</td>
<td>Set various processing options</td>
</tr>
<tr>
<td>setvp</td>
<td>Set the voltage pattern model for subsequent processing</td>
</tr>
<tr>
<td>corrupt</td>
<td>Corrupt the data with visibility errors</td>
</tr>
<tr>
<td>reset</td>
<td>Reset the corruption terms</td>
</tr>
<tr>
<td>setbandpass</td>
<td>Set the bandpasses</td>
</tr>
<tr>
<td>setapply</td>
<td>Arrange for corruption by existing cal tables</td>
</tr>
<tr>
<td>setgain</td>
<td>Set the gains</td>
</tr>
<tr>
<td>settrop</td>
<td>Set tropospheric gain corruptions</td>
</tr>
<tr>
<td>setpointingerror</td>
<td>Set the Pointing error</td>
</tr>
<tr>
<td>setleakage</td>
<td>Set the polarization leakage</td>
</tr>
<tr>
<td>oldsetnoise</td>
<td>Set the noise level fixed sigma (mode=simplenoise) or Brown’s equation (mode=calculate)</td>
</tr>
<tr>
<td>setnoise</td>
<td>Set the noise level fixed sigma (mode=simplenoise) or Brown’s equation using the ATM model for atmospheric parameters</td>
</tr>
<tr>
<td>setpa</td>
<td>Corrupt phase by the parallactic angle</td>
</tr>
<tr>
<td>setseed</td>
<td>Set the seed for the random number generator</td>
</tr>
</tbody>
</table>
simulator.simulator.html

**simulator.simulator - Function**

2.4.1 Construct a simulator tool

**Description**

Create a simulator tool.

**Arguments**

**Returns**

simulatorobject

**Example**

```python
# create a simulator tool
mysim = simulator();
```
simulator.open.html

**simulator.open - Function**

2.4.1 Construct a simulator tool and creating a new MeasurementSet

**Description**

This is used to construct simulator tools. A simulator tool can either be instantiated from an existing MeasurementSet, predicting and/or corrupting data on the given coordinates, or it can be used to create a fundamentally new MeasurementSet from descriptions of the array configuration and the observational parameters. This is useful for making a simulator tool which will make a MeasurementSet from scratch. In order to do this, you must also run setconfig, setfield, setspwindow, setfeed, and settimes. Creating the actual MS is performed by observe. Data can be predict-ed and then corrupted-ed. NOTE: sm.predict assumes the model image units are Jy/pixel, and in fact will overwrite the brightness units of the image itself! In this example, we read in the antenna coordinates from an ASCII file:

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ms</td>
<td>MeasurementSet to be created</td>
</tr>
<tr>
<td>ms</td>
<td>allowed: string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```plaintext
tabname = 'VLAC.LOCAL.TAB'
asciifile = 'VLAC.LOCAL.STN'
mytab=table.create()
mytab.fromascii(tabname, asciifile);
xx=[]; yy=[]; zz=[]; diam=[];
```

1490
```python
xx = mytab.getcol('X');
yy = mytab.getcol('Y');
zz = mytab.getcol('Z');
diam = mytab.getcol('DIAM');
#
sm.open('NEW1.ms')
# do configuration
posvla = me.observatory('vla');  # me.observatory('ALMA') also works!
sm.setconfig(telescopename='VLA', x=xx, y=yy, z=zz, dishdiameter=diam,
              mount='alt-az', antname='VLA',
              coordsystem='local', referencelocation=posvla);

# Initialize the spectral windows
sm.setspwindow(spwname='CBand', freq='5GHz',
                deltafreq='50MHz',
                freqresolution='50MHz',
                nchannels=1,
                stokes='RR RL LR LL');
sm.setspwindow(spwname='LBand', freq='1.420GHz',
                deltafreq='3.2MHz',
                freqresolution='3.2MHz',
                nchannels=32,
                stokes='RR LL');

# Initialize the source and calibrater
sm.setfield(sourcename='My cal',
            sourcedirection=['J2000','00h0m0.0','+45.0.0.000'],
            calcode='A');
sm.setfield(sourcename='My source',
            sourcedirection=['J2000','01h0m0.0','+47.0.0.000']);

sm.setlimits(shadowlimit=0.001, elevationlimit='8.0deg');
sm.setauto(autocorrwt=0.0);
sm.settimes(integrationtime='10s', usehourangle=F,
            referencetime=me.epoch('utc', 'today'));
sm.observe('My cal', 'LBand', starttime='0s', stoptime='300s');
sm.observe('My source', 'LBand', starttime='310s', stoptime='720s');
sm.observe('My cal', 'CBand', starttime='720s', stoptime='1020s');
sm.observe('My source', 'CBand', starttime='1030s', stoptime='1500s');
sm.setdata(spwid=1, fieldid=1);
sm.predict(imagename='M31.MOD');
sm.setdata(spwid=2, fieldid=2);
sm.predict(imagename='BigLBand.MOD');
```
sm.close();
simulator.openfromms.html

**simulator.openfromms** - Function

Construct a simulator tool using an already existing MS

**Description**

This is used to construct simulator tools operating on an existing MS. Data can be predicted and/or corrupted on the MS's given coordinates. NOTE: sm.predict assumes the model image units are Jy/pixel, and in fact will overwrite the brightness units of the image itself!

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ms</code></td>
<td>MeasurementSet to be processed</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>'MS'</td>
</tr>
</tbody>
</table>

**Returns**

`bool`

**Example**

```python
sm.openfromms('3C273XC1.MS');
sm.predict('3C273XC1.imagename');
sm.setnoise(simplenoise='10mJy');
sm.setgain(interval='100s', amplitude=0.01);
sm.corrupt();
sm.close();
```
simulator.close.html

simulator.close - Function

2.4.1 Close the newsimulator tool

Description

This is used to close `newsimulator` tools. Note that the data is written to disk. This is a synonym for done.

Arguments

Returns

bool
**simulator.done - Function**

2.4.1 Close the newsimulator tool

**Description**

This is used to close `newsimulator` tools. Note that the data is written to disk. This is a synonym for close.

**Arguments**

**Returns**

`bool`
simulator.name.html

**simulator.name - Function**

2.4.1 Provide the name of the attached MeasurementSet

**Description**

Returns the name of the attached MeasurementSet.

**Arguments**

**Returns**

string
simulator.summary.html

**simulator.summary - Function**

[2.4.1] Summarize the current state

**Description**

Writes a summary of the properties of the simulator to the default logger.

**Arguments**

**Returns**

bool
simulator.type.html

**simulator.type - Function**

2.4.1 Return the type of this tool

**Description**

This function returns the string ‘simulator’. It is used so that in a script, you can make sure this variable is a simulator tool.

**Arguments**

**Returns**

string
**simulator.settimes** - Function

2.4.1 Set integration time, *etc.*

**Description**

The start and stop times are referenced to `referencetime`. Use either `starttime/stoptime` or `startha/stopha`. If the hour angles are specified, then the start and stop times are calculated such that the start time is later than the reference time, but less than one day later. The hour angles refer to the first source observed.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>integrationtime</td>
<td>Integration time</td>
</tr>
<tr>
<td>usehourangle</td>
<td>Use starttime/stoptime as hour angles - else they are referenced to referencetime</td>
</tr>
<tr>
<td>reference time</td>
<td>Reference time for starttime and stoptime. Epoch Measure. E.g me.epoch('UTC', '50000.0d')</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>integrationtime</td>
<td>any</td>
</tr>
<tr>
<td>usehourangle</td>
<td>true</td>
</tr>
<tr>
<td>reference time</td>
<td>variant 50000.0d</td>
</tr>
</tbody>
</table>

**Returns**

`bool`
simulator.observe.html

**simulator.observe - Function**

2.4.1 Observe a given configuration

**Description**

Observe a given source with a given spectral window for the specified times, including start, stop, integration, and gap times. The start and stop times are referenced to `referencetime`. Use either `starttime/stoptime` or `startha/stopha`. If the hour angles are specified, then the start and stop times are calculated such that the start time is later than the reference time, but less than one day later. The hour angles refer to the first source observed.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourcename</td>
<td>Name of source or field (must be specified)</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: None</td>
</tr>
<tr>
<td>spwname</td>
<td>Unique user-supplied name for this spectral window</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: None</td>
</tr>
<tr>
<td>starttime</td>
<td>Start time referenced to referenceepoch</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant 0s</td>
</tr>
<tr>
<td>stoptime</td>
<td>Stop time referenced to referenceepoch</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant 3600s</td>
</tr>
<tr>
<td>add_observation</td>
<td>Add a new line to the OBSERVATION subtable for this call</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: false</td>
</tr>
<tr>
<td>state_sig</td>
<td>A new line will be added to STATE if the following don’t match</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: true</td>
</tr>
<tr>
<td>state_ref</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: false</td>
</tr>
<tr>
<td>state_cal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: double</td>
</tr>
<tr>
<td></td>
<td>Default: 0.0</td>
</tr>
<tr>
<td>state_load</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: double</td>
</tr>
<tr>
<td></td>
<td>Default: 0.0</td>
</tr>
<tr>
<td>state_sub_scan</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
<tr>
<td>state_obs_mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: OBSERVE_TARGET.ON_SOURCE</td>
</tr>
<tr>
<td>observer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: CASA simulator</td>
</tr>
<tr>
<td>project</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: CASA simulation</td>
</tr>
</tbody>
</table>

**Returns**

bool
Description

Observe given sources with a given spectral window for the specified times, including start, stop, integration, and gap times. The start and stop times are referenced to `referencetime`. Use either `starttime/stoptime` or `startha/stopha`. If the hour angles are specified, then the start and stop times are calculated such that the start time is later than the reference time, but less than one day later. The hour angles refer to the first source observed.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourcenames</td>
<td>Name of sources</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td>None</td>
</tr>
<tr>
<td>spwname</td>
<td>Unique user-supplied name for this spectral window</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>None</td>
</tr>
<tr>
<td>starttimes</td>
<td>Start times referenced to referenceepoch</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td>0s</td>
</tr>
<tr>
<td>stoptimes</td>
<td>Stop time referenced to referenceepoch</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td>3600s</td>
</tr>
<tr>
<td>directions</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>add_observation</td>
<td>Add a new line to the OBSERVATION subtable for this call</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>state_sig</td>
<td>A new line will be added to STATE if the following don’t match</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>true</td>
</tr>
<tr>
<td>state_ref</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td>state_cal</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>0.0</td>
</tr>
<tr>
<td>state_load</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>0.0</td>
</tr>
<tr>
<td>state_sub_scan</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td>state_obs_mode</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>OBSERVE_TARGET#ON_SOURCE</td>
</tr>
<tr>
<td>observer</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>CASA simulator</td>
</tr>
<tr>
<td>project</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>CASA simulation</td>
</tr>
</tbody>
</table>
Returns
bool
Description

Data are flagged for two conditions:

**Below elevation limit** If either of the antennas point below the specified elevation limit then the data are flagged. The elevation is calculated correctly for antennas at different locations (such as occurs in VLBI).

**Shadowing** If one antenna shadows another such that the fractional (geometric) blockage is greater than the specified limit then the data are flagged. No correction for blockage is made for shadowed but non-flagged points.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shadowlimit</td>
<td>Maximum fraction of geometrically shadowed area before flagging occurs</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>1e-6</td>
</tr>
<tr>
<td>elevationlimit</td>
<td>Minimum elevation angle before flagging occurs</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant 10deg</td>
</tr>
</tbody>
</table>

Returns

bool
**simulator.setauto - Function**

2.4.1 Set autocorrelation weight

**Description**

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Weight to assign autocorrelations (0=none)</th>
</tr>
</thead>
<tbody>
<tr>
<td>autocorrwt</td>
<td>allowed: double</td>
</tr>
<tr>
<td></td>
<td>Default: 0.0</td>
</tr>
</tbody>
</table>

**Returns**

bool
simulator.setconfig.html

**simulator.setconfig - Function**

2.4.1 Set the antenna configuration

**Description**

Set the positions of the antennas. Note that the name of the telescope will control which voltage pattern is applied to the data.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>telescopename</strong></td>
<td>Name of the telescope we are simulating (determines VP)</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>VLA</td>
</tr>
<tr>
<td>'VLA'</td>
<td></td>
</tr>
<tr>
<td><strong>x</strong></td>
<td>Vector of x values of all antennas [currently m]</td>
</tr>
<tr>
<td>allowed:</td>
<td>doubleArray</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>y</strong></td>
<td>Vector of y values of all antennas [currently m]</td>
</tr>
<tr>
<td>allowed:</td>
<td>doubleArray</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>z</strong></td>
<td>Vector of z values of all antennas [currently m]</td>
</tr>
<tr>
<td>allowed:</td>
<td>doubleArray</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>dishdiameter</strong></td>
<td>Vector of diameters of all antennas [currently m]</td>
</tr>
<tr>
<td>allowed:</td>
<td>doubleArray</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>offset</strong></td>
<td>Vector of offset of all antennas [currently m]</td>
</tr>
<tr>
<td>allowed:</td>
<td>doubleArray</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>mount</strong></td>
<td>Vector of mount types of all antennas (recognized mounts are 'ALT-AZ', 'EQUATORIAL', 'X-Y', 'ORBITING', 'BIZARRE')</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td>ALT-AZ</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>antname</strong></td>
<td>Vector of names of all antennas</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>padname</strong></td>
<td>Vector of names of pads or stations</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td>P</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>coordsystem</strong></td>
<td>Coordinate system of antenna positions [x,y,z], possibilities are 'global', 'local', 'longlat'</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>global</td>
</tr>
<tr>
<td>'global'</td>
<td></td>
</tr>
<tr>
<td><strong>referencelocation</strong></td>
<td>Reference location [required for local coords] Position Measure of Coordinates of array location. E.g me.position('ITRF', '30.5deg', -20.2deg', 6000km') or me.observatory('ALMA')</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant ALMA</td>
</tr>
<tr>
<td>position measure</td>
<td></td>
</tr>
</tbody>
</table>
Returns
bool

Example

diam := [25, 25, 25, 25, 25]
xx := [50, 100, 150, 200, 250]
yy := [2, -5, -20, -50, -100]
zz := [-0.5, -1.0, -1.5, -2.0, -2.5]
posvla := dm.observatory('vla');
sm.setconfig(telescopename='VLA', x=xx, y=yy, z=zz, dishdiameter=diam,
            mount='alt-az', antname='VLA',
            coordsystem='local', referencelocation=posvla);

1510
## simulator.setknownconfig - Function

### 2.4.1 Set the antenna configuration to a known array

### Description

Sets the configuration to a known array such as VLAA, VLBA, EVN or ATCA6.0A. The arrays are those known to simhelper. All the information needed by setconfig is filled in.

### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrayname</td>
<td>Name of the telescope configuration we are simulating</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>VLA</td>
</tr>
<tr>
<td></td>
<td>'VLA'</td>
</tr>
</tbody>
</table>

### Returns

bool

### Example

```plaintext
sm.setknownconfig('ATCA6.0A');
```
**simulator.setfeed - Function**

2.4.1 Set the feed parameters

**Description**

The goal is to let the feed parameters be specified for each antenna and each spectral window. At this moment, you only have the choice between 'perfect R L' and 'perfect X Y' (i.e., you cannot invent your own corrupted feeds yet). Doesn’t need to be run if you want perfect R and L feeds.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>Mode for specifying feed parameters (currently, perfect only)</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>x</td>
<td>Some very secretive feed array parameter x</td>
</tr>
<tr>
<td></td>
<td>allowed: doubleArray</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
<tr>
<td>y</td>
<td>Some more very secretive feed array parameter y</td>
</tr>
<tr>
<td></td>
<td>allowed: doubleArray</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
<tr>
<td>pol</td>
<td>Guess its the polarization of feed arrays... your guess is as good as mine....if you know better let us know please!</td>
</tr>
<tr>
<td></td>
<td>allowed: stringArray</td>
</tr>
<tr>
<td></td>
<td>Default: R</td>
</tr>
</tbody>
</table>

**Returns**

bool
**simulator.setfield.html**

**simulator.setfield - Function**

2.4.1 Set one or more observed fields

**Description**

Set one or more observed fields, including name, coordinates, calibration code. Can be invoked multiple times for a complex observation. Must be invoked at least once before `observe`. If the distance to the object is set then the phase term includes a curvature for the near-field effect at the center of the image.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of source or field (must be specified)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourcename</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>SOURCE</td>
</tr>
<tr>
<td>'unknown'</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sourcedirection</th>
<th>Direction Measure of Coordinates of source to be observed. E.g me.direction('J2000', '30.5deg','-20.2deg').</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>calcode</th>
<th>Calibration code</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>'OBJ'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>distance</th>
<th>Distance to the object</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant 0m</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

1513
sm.setconfig(telescopename='VLA', x=xx, y=yy, z=zz, dishdiameter=diam, mount='alt-az', antname='VLA', coordsystem='local', referencelocation=dm.observatory('vla'));

sm.setspwindow(spwname='XBAND', freq='8GHz', deltafreq='50MHz', freqresolution='50MHz', nchannels=1, stokes='RR LL');
dir0 = me.direction('B1950', '16h00m0.0', '50d0m0.000')
sm.setfield(sourcename='SIMU1', sourcedirection=dir0);
sm.observe('SIMU1', 'XBAND', integrationtime='10s', usehourangle=T, starttime='0s', stoptime='3600s', referencetime=reftime);
**simulator.setmosaicfield - Function**

Set observed mosaic fields

**Description**

Set mosaic fields by internally invoking `setfield` multiple times. Currently only handle a rectangular mosaicing pattern. Either setfield or setmosaicfield must be invoked at least once before `observe`.

If the distance to the object is set then the phase term includes a curvature for the near-field effect at the center of the image.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourcename</td>
<td>Name of source or field (must be specified).</td>
<td>string</td>
<td>SOURCE 'unknown'</td>
</tr>
<tr>
<td>calcode</td>
<td>Calibration code</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>fieldcenter</td>
<td>Coordinates of mosaic field center</td>
<td>any</td>
<td>variant MDirection</td>
</tr>
<tr>
<td>xmosp</td>
<td>Number of mosaic pointing in horizontal direction</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>ymosp</td>
<td>Number of mosaic pointing in vertical direction</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>mospacing</td>
<td>Spacing between mosaic pointings</td>
<td>any</td>
<td>variant 1arcsec</td>
</tr>
<tr>
<td>distance</td>
<td>Distance to the object</td>
<td>any</td>
<td>variant 0m</td>
</tr>
</tbody>
</table>

**Returns**

1515
bool

Example

sm.setconfig(telescopename='VLA', x=xx, y=yy, z=zz, dishdiameter=diam,
mount='alt-az', antname='VLA',
coordsystem='local', referencelocation=dm.observatory('vla'));

sm.setspwindow(spwname='XBAND', freq='8GHz', deltafreq='50MHz',
freqresolution='50MHz', nchannels=1, stokes='RR LL');
dir0 = me.direction('B1950', '16h00m0.0', '50d0m0.000')
sm.setmosaicfield(sourcename='SIMU1', fieldcenter=dir0,
xmosp=2, ymosp=2, mossspacing='154.5arcsec');
sm.settimes(integrationtime='10s');
sm.observe('SIMU1_1', 'XBAND', starttime='0s', stoptime='100s');
sm.observe('SIMU1_2', 'XBAND', starttime='110s', stoptime='210s');
sm.observe('SIMU1_3', 'XBAND', starttime='220s', stoptime='320s');
sm.observe('SIMU1_4', 'XBAND', starttime='330s', stoptime='430s');
**simulator.setspwindow** - Function

2.4.1 Set one or more spectral windows

**Description**

Set one or more spectral windows for the observations, including starting frequency, number of channels, channel increment and resolution, and stokes parameters observed. Can be invoked multiple times for a complex observation. Must be invoked at least once before `observe`.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>spwname</code></td>
<td>Unique user-supplied name for this spectral window</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: <code>XBAND</code></td>
</tr>
<tr>
<td></td>
<td>'XBAND'</td>
</tr>
<tr>
<td><code>freq</code></td>
<td>Starting frequency</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant 8.0e9Hz</td>
</tr>
<tr>
<td><code>deltafreq</code></td>
<td>Frequency increment per channel</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant 50e6Hz</td>
</tr>
<tr>
<td><code>freqresolution</code></td>
<td>Frequency resolution per channel</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant 50.e6Hz</td>
</tr>
<tr>
<td><code>refcode</code></td>
<td>Spectral reference code e.g. LSRK, TOPO, BARY</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: TOPO</td>
</tr>
<tr>
<td><code>nchannels</code></td>
<td>Number of channels</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 1</td>
</tr>
<tr>
<td><code>stokes</code></td>
<td>Stokes types to simulate</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: RR LL</td>
</tr>
<tr>
<td></td>
<td>'RR LL'</td>
</tr>
</tbody>
</table>

**Returns**

1517
bool

Example

To simulate a two spectral window (or two IF’s in VLA jargon) data set, use setpwid as follows (here we are simulating 16 channels, 50MHz wide channel for each spectral window)

```python
sm.setspwindow(spwname='CBAND', freq='2GHz', deltafreq='50MHz',
               freqresolution='50MHz', nchannels=16, stokes='RR LL');
sm.setspwindow(spwname='SBAND', freq='5GHz', deltafreq='50MHz',
               freqresolution='50MHz', nchannels=16, stokes='RR LL');
```

Note that the spwname is used in `{\tt observe}` to determine which spectral window is used.
simulator.setdata.html

**simulator.setdata - Function**

2.4.1 Set the data parameters selection for subsequent processing

**Description**

This setup tool function selects which data are to be used subsequently. After invocation of setdata, only the selected data are operated on.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>spwid</td>
<td>Spectral Window Ids (0 relative) to select</td>
<td>intArray</td>
<td>0</td>
</tr>
<tr>
<td>fieldid</td>
<td>Field Ids (0 relative) to select</td>
<td>intArray</td>
<td>0</td>
</tr>
<tr>
<td>msselect</td>
<td>TQL select string applied as a logical &quot;and&quot; with the other selections</td>
<td>string</td>
<td>String</td>
</tr>
</tbody>
</table>

**Returns**

bool

1519
**simulator.predict - Function**

2.4.1 Predict astronomical data from an image

**Description**

Predict astronomical data from an image. The (u,v) coordinates already exist, either from a MeasurementSet we have read in or by generating the MeasurementSet coordinates and empty data through `create()`. We simply predict onto these coordinates. NOTE: sm.predict assumes the model image units are Jy/pixel, and in fact will overwrite the brightness units of the image itself!

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>imagename</td>
<td>Name of image from which to predict visibilities</td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>complist</td>
<td>Name of component list</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>String</td>
</tr>
<tr>
<td>incremental</td>
<td>Add this model to the existing Data Visibilities?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

bool
simulator.setoptions.html

**simulator.setoptions** - Function

2.4.1 Set various processing options

**Description**

For most of these, set the options for `predict` details. See also `imager` help for more details.
To simulate single dish data, use `gridft=SD` and `gridfunction=PB`.

**Arguments**
Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftmachine</td>
<td>Fourier transform machine. Possibilities are 'ft', 'sd'</td>
<td>string</td>
<td>ft</td>
</tr>
<tr>
<td>cache</td>
<td>Size of gridding cache in complex pixels</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>tile</td>
<td>Size of a gridding tile in pixels (in 1 dimension)</td>
<td>int</td>
<td>16</td>
</tr>
<tr>
<td>gridfunction</td>
<td>Gridding function. String: 'SF'—'BOX'—'PB'</td>
<td>string</td>
<td>SF</td>
</tr>
<tr>
<td>location</td>
<td>Location used in phase rotations. Position Measure of Coordinates of array location. E.g me.position('ITRF', '30.5deg', '-20.2deg', '6000km') or me.observatory('ALMA')</td>
<td>any</td>
<td>variant ALMA</td>
</tr>
<tr>
<td>padding</td>
<td>Padding factor in image plane (&gt;=1.0)</td>
<td>double</td>
<td>1.3</td>
</tr>
<tr>
<td>facets</td>
<td>Number of facets</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>maxdata</td>
<td>Maximum data to write to a single TSM file (MB)</td>
<td>double</td>
<td>2000.0</td>
</tr>
<tr>
<td>wprojplanes</td>
<td>Number of projection planes when using wproject as the ft-machine</td>
<td>int</td>
<td>1</td>
</tr>
</tbody>
</table>

Returns

bool

Example
sm.setoptions(cache=1000000, tile=32, gridfunction='BOX', me.location('vla'))
simulator.setvp.html

**simulator.setvp - Function**

2.4.1 Set the voltage pattern model for subsequent processing

**Description**

Set the voltage pattern model (and hence, the primary beam) used for a Telescope. There are currently two ways to set the voltage pattern: by using the extensive list of defaults which the system knows about, or by creating a voltage pattern description with the vpmanager. The default voltage patterns include both a high and a low frequency VP for the WSRT, a VP for each observing band at the AT, several VP’s for the VLA, including the appropriate beam squint for each observing band, and Gaussian for the BIMA dishes. If you are simulating a telescope which doesn’t yet exist, you will need to supply a model voltage pattern using the vpmanager.

**Arguments**
### Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dovp</code></td>
<td>Multiply by the voltage pattern (ie, primary beam) when simulating</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td><code>usedefaultvp</code></td>
<td>Look up the default VP for this telescope and frequency?</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td><code>vptable</code></td>
<td>If <code>usedefaultvp</code> is false, provide a VP Table made with vpmanger</td>
<td>string</td>
<td>Table</td>
</tr>
<tr>
<td><code>dosquint</code></td>
<td>Activate the beam squint in the VP model</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td><code>parangleinc</code></td>
<td>Parallactice angle increment for squint application</td>
<td>any</td>
<td>variant 360deg</td>
</tr>
<tr>
<td><code>skyposthreshold</code></td>
<td>Position threshold on the sky for feed arrays ??</td>
<td>any</td>
<td>variant 180deg</td>
</tr>
<tr>
<td><code>pblimit</code></td>
<td>Primary beam limit to use in feed arrays</td>
<td>double</td>
<td>1.0e-2</td>
</tr>
</tbody>
</table>

### Returns

`bool`

### Example

```python
sm.setvp(dovp=T, usedefaultvp=F, vptable='MyAlternateVLAPBModel.TAB', dosquint=F);
```
simulator.corrupt.html

**simulator.corrupt - Function**

2.4.1 Corrupt the data with visibility errors

**Description**

Add errors specified by the `set` functions (such as noise, gains, polarization leakage, bandpass, etc) to the visibility data. The errors are applied to the `MODEL_DATA`, and written to the `DATA` and `CORRECTED_DATA` columns. Note that `corrupt` handles only visibility-plane effects, not image-plane effects such as pointing errors and voltage patterns, which get applied in `predict`. Note, the function applies errors to both cross- and auto-correlation data; The auto-correlation data are corrupted properly only for the thermal noise set by `setnoise`. NOTE: `sm.predict` assumes the model image units are Jy/pixel, and in fact will overwrite the brightness units of the image itself!

**Arguments**

**Returns**

`bool`

**Example**

```python
sm.openfromms('3C273XC1.MS');
sm.predict('3C273XC1.FAKE.IMAGE');
sm.setnoise( mode='simplenoise', simplenoise='0.1Jy');
sm.setpa( mode='calculate');
sm.corrupt();
```
Reset the corruption terms

Description

Reset the visibility corruption terms: this means that corrupt introduces no errors.

Arguments

Returns

bool
**simulator.setbandpass - Function**

2.4.1 Set the bandpasses

**Description**

Set the level of bandpass errors. The error distributions are normal, mean zero, with the variances as specified. (Not yet implemented).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>mode</strong></td>
<td>Mode of operation. String: 'calculate'—'table'</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>'calculate'</td>
</tr>
<tr>
<td>table</td>
<td>Name of table</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>interval</td>
<td>Coherence interval e.g. '1h'</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant 3600s</td>
</tr>
<tr>
<td>amplitude</td>
<td>Variances errors in amplitude and phase</td>
</tr>
<tr>
<td>allowed:</td>
<td>doubleArray</td>
</tr>
<tr>
<td>Default:</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Returns**

bool
simulator.setapply.html

**simulator.setapply - Function**

2.4.1 Arrange for corruption by existing cal tables

**Description**

Arrange for corruption by existing cal tables, in a manner exactly analogous to calibrater.setapply.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>Calibration table name</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>string</td>
</tr>
<tr>
<td>type</td>
<td>Component type</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>string</td>
</tr>
<tr>
<td>t</td>
<td>Interpolation interval (seconds)</td>
</tr>
<tr>
<td>allowed</td>
<td>double</td>
</tr>
<tr>
<td>Default</td>
<td>0.0</td>
</tr>
<tr>
<td>field</td>
<td>Select on field</td>
</tr>
<tr>
<td>allowed</td>
<td>any</td>
</tr>
<tr>
<td>Default</td>
<td>variant</td>
</tr>
<tr>
<td>interp</td>
<td>Interpolation type (in time)</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>string</td>
</tr>
<tr>
<td>calwt</td>
<td>Calibrate weights?</td>
</tr>
<tr>
<td>allowed</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>spwmap</td>
<td>Spectral windows to apply</td>
</tr>
<tr>
<td>allowed</td>
<td>intArray</td>
</tr>
<tr>
<td>Default</td>
<td>-1</td>
</tr>
<tr>
<td>opacity</td>
<td>Array-wide zenith opacity (for type='TOPAC')</td>
</tr>
<tr>
<td>allowed</td>
<td>double</td>
</tr>
<tr>
<td>Default</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returns</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.4.1 Set the gains

**Description**

Set the level of gain errors. Gain drift is implemented as fractional brownian motion with rms amplitude as specified. Interval is not currently used, but future statistical models for gain errors (e.g. simple Gaussian) will use it.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>Mode of operation. String: 'fbm'</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>'fbm'</td>
</tr>
<tr>
<td>table</td>
<td>Optional name of table to write</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>'fbm'</td>
</tr>
<tr>
<td>interval</td>
<td>timescale for gain variations NOT USED</td>
</tr>
<tr>
<td>allowed</td>
<td>any</td>
</tr>
<tr>
<td>Default</td>
<td>variant 10s</td>
</tr>
<tr>
<td>amplitude</td>
<td>amplitude scale (RMS) for gain variations [real,imag] or scalar</td>
</tr>
<tr>
<td>allowed</td>
<td>doubleArray</td>
</tr>
<tr>
<td>Default</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>[]</td>
</tr>
</tbody>
</table>

**Returns**

bool
simulator.settrop.html

**simulator.settrop** - Function

**2.4.1 Set tropospheric gain corruptions**

**Description**

Set the atmosphere.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Mode of operation - screen or individual antennas</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>mode</strong></td>
<td><strong>Mode of operation - screen or individual antennas</strong></td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>screen</td>
</tr>
<tr>
<td></td>
<td>'screen'</td>
</tr>
<tr>
<td><strong>table</strong></td>
<td>Name of cal table</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>''</td>
</tr>
<tr>
<td><strong>pwv</strong></td>
<td>total precipitable water vapour in mm</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>3.0</td>
</tr>
<tr>
<td><strong>deltapwv</strong></td>
<td>RMS PWV fluctuations <em>as a fraction of PWV parameter</em></td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>0.15</td>
</tr>
<tr>
<td><strong>beta</strong></td>
<td>exponent of fractional brownian motion</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>1.1</td>
</tr>
<tr>
<td><strong>windspeed</strong></td>
<td>wind speed for screen type corruption (m/s)</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>7.</td>
</tr>
</tbody>
</table>

**Returns**

bool
**simulator.setpointingerror.html**

**simulator.setpointingerror - Function**

### 2.4.1 Set the Pointing error

**Description**

Set the pointing error from a calpointing table

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>epjtablename</td>
<td>Name of a table that has E-Jones errors for Pointing</td>
</tr>
<tr>
<td>applypointingoffsets</td>
<td>Apply pointing offsets</td>
</tr>
<tr>
<td>dopbcorrection</td>
<td>apply primary beam correction</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>allowed</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
</tr>
</tbody>
</table>

| Returns       | bool                                                                 |

1533
**simulator.setleakage.html**

**simulator.setleakage - Function**

2.4.1 Set the polarization leakage

**Description**

Set the level of polarization leakage between feeds. Currently, no time dependence is available.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>mode</strong></td>
<td>Mode of operation. String: 'constant'</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>constant</td>
<td>'constant'</td>
</tr>
<tr>
<td><strong>table</strong></td>
<td>Optional name of table to write</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td><strong>amplitude</strong></td>
<td>Magnitude of pol leakage [real,imag]</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>doubleArray</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>0.01</td>
<td>[]</td>
</tr>
<tr>
<td><strong>offset</strong></td>
<td>Mean of pol leakage [real,imag]</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>doubleArray</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>0.</td>
<td>[]</td>
</tr>
</tbody>
</table>

**Returns**

bool
simulator.oldsetnoise - Function

2.4.1 Set the noise level fixed sigma (mode=simplenoise) or Brown’s equation (mode=calculate) OBSOLETE VERSION

Description

Set various system parameters from which the thermal (ie, random additive) noise level will be calculated.

For mode=simplenoise, one specifies the standard deviation for the noise to be added to real and imaginary parts of the visibility.

For mode=calculate, the noise will vary with dish diameter, antenna efficiency, system temperature, opacity, sky temperature, etc. The noise will increase with the airmass if $\tau$ is greater than zero. The noise is calculated according to the Brown Equation (ie, R.L. Brown’s calculation of MMA sensitivity, 3Oct95):

$$
\Delta S = \frac{4\sqrt{2} [T_{rx} e^{\tau A} + T_{atm} (e^{\tau A} - \epsilon_t) + T_{cmb}]}{\epsilon_q \epsilon_a \pi D^2 \sqrt{\Delta \nu \Delta t}}
$$

(2.16)

Arguments
### Inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Allowed Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>Mode of operation. String: 'simplenoise'—'calculate'</td>
<td>string</td>
<td>calculate 'simplenoise', 'calculate'</td>
</tr>
<tr>
<td>table</td>
<td>Name of noise table - not currently implemented</td>
<td>string</td>
<td>&quot;</td>
</tr>
<tr>
<td>simplenoise</td>
<td>Level of noise (if mode=simplenoise)</td>
<td>any</td>
<td>variant 0.0 Jy</td>
</tr>
<tr>
<td>antiefficiency</td>
<td>antenna efficiency</td>
<td>double</td>
<td>0.8</td>
</tr>
<tr>
<td>coriefficiency</td>
<td>Correlation efficiency</td>
<td>double</td>
<td>0.85</td>
</tr>
<tr>
<td>spillefficiency</td>
<td>Forward spillover efficiency</td>
<td>double</td>
<td>0.85</td>
</tr>
<tr>
<td>tau</td>
<td>Atmospheric Opacity</td>
<td>double</td>
<td>0.1</td>
</tr>
<tr>
<td>trx</td>
<td>Receiver temp (ie, all non-atmospheric Tsys contributions) [K]</td>
<td>double</td>
<td>50</td>
</tr>
<tr>
<td>tatmos</td>
<td>(Physical, not Brightness) Temperature of atmosphere [K]</td>
<td>double</td>
<td>230.0</td>
</tr>
<tr>
<td>tcmb</td>
<td>Temperature of cosmic microwave background [K]</td>
<td>double</td>
<td>2.7</td>
</tr>
</tbody>
</table>

### Returns

bool
simulator.setnoise.html

**simulator.setnoise** - Function

2.4.1 Set the noise level fixed sigma (mode=simplenoise) or Brown’s equation using the ATM model for frequency-dependent atmospheric opacity (mode=tsys-atm) or Brown’s equation, manually specifying the zenith opacity (constant across the band) and atmospheric temperature (mode=tsys-manual)

**Description**

Set various system parameters from which the thermal (ie, random additive) noise level will be calculated.

For mode=simplenoise, one specifies the standard deviation for the noise to be added to real and imaginary parts of the visibility.

For mode=tsys-atm or tsys-atm, the noise will vary with dish diameter, antenna efficiency, system temperature, opacity, sky temperature, etc. The noise will increase with the airmass if tau is greater than zero. The noise is calculated according to the Brown Equation (ie, R.L. Brown’s calculation of MMA sensitivity, 3Oct95): 

\[
\text{dS} = 4 \sqrt{2} \left[ T_{rx} \eta \tau + T_{atm} (\eta \tau \epsilon_l) + T_{cmb} \right] / \left[ \epsilon_q \epsilon_a \pi D^2 \sqrt{\Delta \nu \Delta t} \right]
\]

For mode=tsys-atm, the sky brightness temperature is calculated using an atmospheric model created for the user-input PWV. For mode=tsys-manual, the user specifies the sky brightness temperature manually.

**Arguments**
**Inputs**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>Mode of operation.</td>
<td>string</td>
<td>simplenoise</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: simplenoise</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>'simplenoise'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>'tsys-atm'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>'tsys-manual'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>table</td>
<td>Name of optional cal table to write</td>
<td>string</td>
<td>&quot;</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: &quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>simplenoise</td>
<td>Level of noise if not calculated</td>
<td>any</td>
<td>variant 0.1Jy</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pground</td>
<td>Ground pressure for ATM model (if tsys-atm)</td>
<td>any</td>
<td>variant 560mbar</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
<td></td>
<td></td>
</tr>
<tr>
<td>relhum</td>
<td>Ground relative humidity for ATM model (if tsys-atm)</td>
<td>double</td>
<td>20.0</td>
</tr>
<tr>
<td></td>
<td>allowed: double</td>
<td></td>
<td></td>
</tr>
<tr>
<td>altitude</td>
<td>Site altitude for ATM model (if tsys-atm)</td>
<td>any</td>
<td>variant 5000m</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
<td></td>
<td></td>
</tr>
<tr>
<td>waterheight</td>
<td>Height of water layer for ATM model (if tsys-atm)</td>
<td>any</td>
<td>variant 200m</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pwv</td>
<td>Precipitable Water Vapor ATM model (if tsys-atm)</td>
<td>any</td>
<td>variant 1mm</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tatmos</td>
<td>Temperature of atmosphere [K] (if tsys-manual)</td>
<td>double</td>
<td>250.0</td>
</tr>
<tr>
<td></td>
<td>allowed: double</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tau</td>
<td>Zenith Atmospheric Opacity (if tsys-manual)</td>
<td>double</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>allowed: double</td>
<td></td>
<td></td>
</tr>
<tr>
<td>antefficiency</td>
<td>Antenna efficiency</td>
<td>double</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>allowed: double</td>
<td></td>
<td></td>
</tr>
<tr>
<td>spillefficiency</td>
<td>Forward spillover efficiency</td>
<td>double</td>
<td>0.85</td>
</tr>
<tr>
<td></td>
<td>allowed: double</td>
<td></td>
<td></td>
</tr>
<tr>
<td>correfficiency</td>
<td>Correlation efficiency</td>
<td>double</td>
<td>0.88</td>
</tr>
<tr>
<td></td>
<td>allowed: double</td>
<td></td>
<td></td>
</tr>
<tr>
<td>trx</td>
<td>Receiver temp (ie, all non-atmospheric Tsys contributions) [K]</td>
<td>double</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>allowed: double</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tground</td>
<td>Temperature of ground/spill [K]</td>
<td>double</td>
<td>270.0</td>
</tr>
<tr>
<td></td>
<td>allowed: double</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tcmb</td>
<td>Temperature of cosmic microwave background [K]</td>
<td>double</td>
<td>2.73</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OTF</td>
<td>calculate noise on-the-fly (WARNING: only experts with high-RAM machines should use False)</td>
<td>bool</td>
<td></td>
</tr>
</tbody>
</table>
Returns
bool
**simulator.setpa** - Function

**2.4.1 Corrupt phase by the parallactic angle**

**Description**

Corrupt phase by the parallactic angle

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>mode</strong></td>
<td>Mode of operation. String: 'calculate'—'table'</td>
<td>allowed: string; Default: calculate</td>
</tr>
<tr>
<td><strong>table</strong></td>
<td>Name of table</td>
<td>allowed: string; Default: &quot;</td>
</tr>
<tr>
<td><strong>interval</strong></td>
<td>Interval for parallactic angle application, e.g. '10s'</td>
<td>allowed: any; Default: variant 10s</td>
</tr>
</tbody>
</table>

**Returns**

bool
simulator.setseed.html

**simulator.setseed - Function**

Set the seed for the random number generator

**Description**

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Seed</th>
<th>allowed:</th>
<th>int</th>
</tr>
</thead>
<tbody>
<tr>
<td>seed</td>
<td>Seed</td>
<td>Default:</td>
<td>185349251</td>
</tr>
<tr>
<td></td>
<td></td>
<td>185349251</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

---

UtilityPackage.html
Chapter 3

Package Utility

Utilities useful beyond astronomical processing misc-Module.html

3.1 misc - Module

Miscellaneous tools module
3.1.1 logsink - Tool

tool for logsink

Requires:

Synopsis

Description

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logsink</td>
<td>Construct a logsink tool</td>
</tr>
<tr>
<td>origin</td>
<td>Set the origin of the message</td>
</tr>
<tr>
<td>processorOrigin</td>
<td>Set the CASA processor origin</td>
</tr>
<tr>
<td>filter</td>
<td>Set the filter level</td>
</tr>
<tr>
<td>filterMsg</td>
<td>Add messages to the filter out list</td>
</tr>
<tr>
<td>clearFilterMsgList</td>
<td>Clear list of messages to be filter out</td>
</tr>
<tr>
<td>post</td>
<td>Post a message</td>
</tr>
<tr>
<td>postLocally</td>
<td>Post locally</td>
</tr>
<tr>
<td>localId</td>
<td>Get local ID</td>
</tr>
<tr>
<td>version</td>
<td>version of CASA</td>
</tr>
<tr>
<td>id</td>
<td>Get ID</td>
</tr>
<tr>
<td>setglobal</td>
<td>Set this logger to be the global logger</td>
</tr>
<tr>
<td>setlogfile</td>
<td>Set the name of file for logger output</td>
</tr>
<tr>
<td>showconsole</td>
<td>Choose to send messages to the console/terminal</td>
</tr>
<tr>
<td>logfile</td>
<td>Returns the full path of the log file</td>
</tr>
<tr>
<td>ompNumThreadsTest</td>
<td>Determines the number of OpenMP threads in the current parallel region</td>
</tr>
<tr>
<td>ompGetNumThreads</td>
<td>Returns the number of OpenMP threads in the current parallel region</td>
</tr>
<tr>
<td>ompSetNumThreads</td>
<td>Specifies the number of OpenMP threads used by default in subsequent parallel region</td>
</tr>
<tr>
<td>setMemoryTotal</td>
<td>Sets the memory total value to be returned by HostInfo</td>
</tr>
<tr>
<td>setMemoryFraction</td>
<td>Sets the memory fraction value to be returned by HostInfo</td>
</tr>
<tr>
<td>setNumCPUs</td>
<td>Sets the number of CPUs to be returned by HostInfo</td>
</tr>
<tr>
<td>getMemoryTotal</td>
<td>Returns the memory total value from HostInfo</td>
</tr>
<tr>
<td>getNumCPUs</td>
<td>Returns the number of CPUs from HostInfo</td>
</tr>
</tbody>
</table>
logsink logsink - Function

3.1.1 Construct a logsink tool

Description

Arguments

Inputs

Returns
logsink

Returns
bool

Example
logsink.origin.html

logsink.origin - Function

3.1.1 Set the origin of the message

Description

Sets the origin of messages to be displayed

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>The origin of a log messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>fromwhere</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

bool
logsink.processorOrigin.html

**logsink.processorOrigin - Function**

3.1.1 Set the CASA processor origin

**Description**

Sets the CASA processor origin which is shown at the end of each log origin

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fromwhere</td>
<td>Input CASA processor origin name</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

bool

1547
set the filter level

Description

Set the filter level of logging messages to be displayed. This will determine what log messages go into the log file. The logger itself can adjust what gets displayed so you could set INFO5 and then filter in the logger everything above INFO1.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Level of messages to display to the console/log file</th>
</tr>
</thead>
<tbody>
<tr>
<td>level</td>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default: ERROR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WARN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INFO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INFO1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INFO2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INFO3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INFO4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INFO5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEBUG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEBUG1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEBUG2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INFO</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool
logsink.filterMsg - Function

3.1.1 Add messages to the filter out list

Description

Add messages to the filter out list

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>msgList</th>
<th>Array of strings identifying messages to filter out</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>allowed: stringArray</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

void
3.1.1 Clear list of messages to be filtered out

**Description**

Clear list of messages to be filtered out

**Arguments**

**Returns**

void
logsink.post.html

**logsink.post - Function**

3.1.1 Post a message

**Description**

If the message passes the filter, write it (same as postLocally)

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>Message to be posted</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>priority</td>
<td>Priority of message to be posted</td>
<td>string</td>
<td>INFO</td>
</tr>
<tr>
<td>origin</td>
<td>Origin of message to be posted</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool
logsink.postLocally - Function

3.1.1 Post locally

Description

If the message passes the filter, write it

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>Message to be posted</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>priority</td>
<td>Priority of message to be posted</td>
<td>string</td>
<td>INFO</td>
</tr>
<tr>
<td>origin</td>
<td>Origin of message to be posted</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool

Example
**logsink.localId.html**

**logsink.localId - Function**

3.1.1 Get local ID

**Description**

Returns the id for this class

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
</table>

**Returns**

string

**Example**

1553
logsink.version.html

**logsink.version - Function**

3.1.1 version of CASA

**Description**

Returns the version of CASA as well as sending it to the log

**Arguments**

**Inputs**

**Returns**

string

**Example**

casalog.version()
**logsink.id - Function**

3.1.1 Get ID

**Description**

Returns the ID of the LogSink in use

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
</table>

**Returns**

string

**Example**

1555
logsink.setglobal.html

**logsink.setglobal - Function**

3.1.1 Set this logger to be the global logger

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isglobal</td>
<td>Use as global logger</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: true</td>
</tr>
</tbody>
</table>

**Returns**

bool
logsink.setlogfile.html

**logsink.setlogfile - Function**

3.1.1 Set the name of file for logger output

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>filename for logger</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>casapy.log</td>
</tr>
</tbody>
</table>

**Returns**

bool
logsink.showconsole - Function

3.1.1 Choose to send messages to the console/terminal

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onconsole</td>
<td>All messages to the console as well as log file</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: false</td>
</tr>
</tbody>
</table>

Returns

bool
logsink.logfile.html

logsink.logfile - Function

3.1.1 Returns the full path of the log file

Description

Returns the full path of the log file

Arguments

Inputs

Returns
string

Example

logfile = casalog.logfile()
**logsink.ompNumThreadsTest - Function**

3.1.1 Determines the number of OpenMP threads in the current parallel region using an OpenMP reduction pragma

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
</table>

**Returns**

int

**Example**

```python
omp_num_thread = casalog.ompNumThreadsTest()
```
logsink.ompGetNumThreads - Function

3.1.1 Returns the number of OpenMP threads in the current parallel region

Arguments

Returns

int

Example

omp_num_thread = casalog.ompNumThreadsTest()
logsink.ompSetNumThreads.html

**logsink.ompSetNumThreads - Function**

3.1.1 Specifies the number of OpenMP threads used by default in subsequent parallel regions

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>allowed: int</th>
<th>Default: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>numThreads</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```
casalog.ompSetNumThreads(2)
```
logsink.setMemoryTotal.html

**logsink.setMemoryTotal - Function**

Sets the memory total value to be returned by HostInfo

### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>allowed: int</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory</td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

### Returns

int

### Example

casalog.setMemoryTotal(4*1024)

1563
logsink.setMemoryFraction.html

**logsink.setMemoryFraction - Function**

3.1.1 Sets the memory fraction value to be returned by HostInfo

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>memfrac</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
</tbody>
</table>

**Returns**

int

**Example**

```
casalog.setMemoryFraction(50)
```
logsink.setNumCPUs.html

**logsink.setNumCPUs - Function**

3.1.1 Sets the number of CPUs to be returned by HostInfo

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>cores</th>
<th>allowed: int</th>
<th>Default: 0</th>
</tr>
</thead>
</table>

**Returns**

int

**Example**

casalog.setNumCPUs(4)
logsink.getMemoryTotal.html

**logsink.getMemoryTotal - Function**

3.1.1 Returns the memory total value from HostInfo

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>use_aipsrc</td>
<td>bool</td>
<td>true</td>
</tr>
</tbody>
</table>

**Returns**

int

**Example**

```
casalog.getMemoryTotal(true)
```
logsink.getNumCPUs.html

**logsink.getNumCPUs - Function**

3.1.1 Returns the number of CPUs from HostInfo

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>use_aipsrc</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>true</td>
</tr>
</tbody>
</table>

**Returns**

int

**Example**

casalog.getNumCPUs(true)
3.1.2 deconvolver - Tool

deconvolver tool
Requires:
  Synopsis

Description

deconvolver is a tool that deconvolves a known point spread function from an image. A deconvolver must be constructed for each dirty image and point spread function for which one wishes to do processing. Multiple copies of deconvolver may be made at any time (provide they are given different names).

Methods

deconvolver Construct a deconvolver tool
open Open a new dirty image and PSF
reopen Reopen the dirty image and PSF
close Close the deconvolver tool
done Terminate the deconvolver process
summary Summarize the current state
boxmask Construct a mask from blc, trc
regionmask Construct a mask image from a region
clipimage Zero all pixels where Stokes I is below a threshold
clarkclean Make a clean image using the Clark Clean a threshold
fullclarkclean Make a clean image using the Clark Clean a threshold
dirtyname Return the name of the dirty-image table
psfname Return the name of the PSF-image table
make Make an empty image
convolve Convolves an image with the PSF
makegaussian Make an image with a single gaussian component
state Return the “state” of the tool
updatestate [A GUI builders related function]Update the GUI to reflect the current state
clean Make a clean image with Hogbom or MultiScale Clean
naclean Make a clean image with Hogbom with self masking
setscales Set the scale sizes for MultiScale Clean
fft Fourier transform the specified model
restore Restore the residuals
residual Find the residuals
smooth smooth the image
mem Make the mem image
makeprior Make the mem’s prior image, or make a mask
mtopen Init : Make a series of images using a Multi-Term Clean algorithm
mtclean Make a series of images using a Multi-Term Clean algorithm
mtrestore Restore the Multi-Term residuals
mtcalcpowerlaw Interpret Taylor coefficients as a power law, and compute spectral index
deconvolver.deconvolver.html

**deconvolver.deconvolver - Function**

3.1.2 Construct a deconvolver tool

**Description**

This is used to construct deconvolver tools associated with a dirty image and point spread function. The deconvolver tool may then be used to generate various types of images. Note that a new executable is started every time the constructor is called.

Example:

```python
mydecon=casac.deconvolver() mydecon.open('dirty.image', 'psf.image')
```

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dirtyname</td>
<td>Dirty image to be processed. Table name.</td>
<td>string</td>
<td>dirtyname</td>
</tr>
<tr>
<td>psfname</td>
<td>Point spread function to be processed. Table name</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

casadeconvolver

**Example**

```python
deco=casac.deconvolver()
deco.open('3C273XC1.dirty', '3C273XC1.psf')
deco.clean(model='3C273XC1.clean', niter=10000, gain=0.2)
deco.close()
```
deconvolver.open.html

**deconvolver.open - Function**

3.1.2 Open a new dirty image and PSF

**Description**

Close the current images and open a new dirty image and PSF instead. The current state of `deconvolver` is retained, except for the data selection.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dirty</td>
<td>Dirty image to be processed</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>psf</td>
<td>Point spread function to be processed</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>warn</td>
<td>Produce warning messages if psf is not provided</td>
<td>bool</td>
<td>true</td>
</tr>
</tbody>
</table>

**Returns**

bool
3.1.2 Reopen the dirty image and PSF

Description

Close and reopen the current dirty and PSF images, and make new convolvers and cleaners. The main benefit of this method is to flush the residual image and replace it with the dirty image.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
</table>

Returns

bool
deconvolver.close.html

**deconvolver.close - Function**

3.1.2 Close the deconvolver tool

**Description**

This is used to close deconvolver tools. Note that the data is written to disk. The deconvolver process keeps running until a done tool function call is performed.

**Arguments**

**Returns**

bool

**Example**

dc.open('3C273XC1.dirty', '3C273XC1.psf')
dc.clean(model='3C273XC1.clean');
dc.close()
3.1.2 Terminate deconvolver process

**Description**

This is used to totally stop the deconvolver process. It is a good idea to conserve memory use on your machine by stopping the process once you no longer need it.

**Arguments**

**Returns**

bool

**Example**

```python
dc.open('3C273XC1.dirty', '3C273XC1.psf')
dc.clean(model='3C273XC1.clean');
dc.close()
dc.done()
```
deconvolver.summary.html

**deconvolver.summary - Function**

3.1.2 Summarize the current state

**Description**

Writes a summary of the properties of the deconvolver to the default logger. This includes:

- The names of the dirty image and PSF (set in construction or via the open function.
- The current beam fit

**Arguments**

**Returns**

bool

**Example**

```python
dc.open('3C273XC1.dirty', '3C273XC1.psf')
dc.summary()
```
3.1.2 Construct a mask from blc, trc

Description

A mask image is an image with the same shape as the other images but with values between 0.0 and 1.0 as a pixel value. Mask images are used in deconvolver to control the region selected in a deconvolution. In the Clark CLEAN, the mask image can usefully have any value between 0.0 and 1.0. Intermediate value discourage but do not rule out selection of clean components in that region. This is accomplished by multiplying the residual image by the mask prior to entering the minor cycle. Note that if you do use a mask for the Clark or Hogbom Clean, it must cover only a quarter of the image. boxmask does not enforce this requirement.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>mask</strong></td>
<td>name of mask image</td>
<td>string</td>
</tr>
<tr>
<td><strong>blc</strong></td>
<td>Bottom left corner</td>
<td>intArray</td>
</tr>
<tr>
<td><strong>trc</strong></td>
<td>Top right corner</td>
<td>intArray</td>
</tr>
<tr>
<td><strong>fillvalue</strong></td>
<td>Value to fill in</td>
<td>any</td>
</tr>
<tr>
<td><strong>outsidevalue</strong></td>
<td>outside value</td>
<td>any</td>
</tr>
</tbody>
</table>

Returns

bool
Example

dc.boxmask(mask='bigmask', blc=[56,45,1,1], trc=[87,93,4,1])
dc.clean(mask='bigmask', model='3C273XC1.clean.masked', niter=1000)
deconvolver.regionmask.html

**deconvolver.regionmask - Function**

**3.1.2** Construct a mask image from a region

**Description**

A mask image is an image with the same shape as the other images but with values between 0.0 and 1.0 as a pixel value. Mask images are used in imager to control the region selected in a deconvolution.

In Clark CLEAN, the mask image can usefully have any value between 0.0 and 1.0. Intermediate value is discouraged but do not rule out selection of clean components in that region. This is accomplished by multiplying the residual image by the mask prior to entering the minor cycle. Note that if you do use a mask for the Clark or Hogbom Clean, it must cover only a quarter of the image. `regionmask` does not enforce this requirement.

The function regionmask also allows multiple regions to be used. A record of the regions can be made as in the example below.

Regions can be made in many different ways using the regionmanager functions. An example using wbox function is given below. The default regionmanager tool 'rg' can be used for cases the user want to have flexibility in manipulating regions. The `region` parameter takes a record that comes from the regionmanager output. The parameter boxes allow the user to sent in a list of 4 elements numbers representing blc’s and trc’s.

If both the parameters, `regions` and `boxes` are used the a union is done with the two sets of region thus defined.

**Arguments**

1578
Inputs

**mask**
- Name of mask image
- Allowed: string
- Default:

**region**
- Region record usually from regionmanager
- Allowed: record
- Default: unset

**boxes**
- List of 4 elements lists e.g. 
  - [[xblc1, yblc1, xtrc1, ytrc1],
  - [[xblc2, yblc2, xtrc2, ytrc2]]
- Allowed: any
- Default: variant

**value**
- Value to set the mask to
- Allowed: double
- Default: 1.0

Returns

bool

Example

Makes a mask then cleans using it.

dc.open(dirty.image', 'psf.image')

a=[100.0, 100.0, 200, 200.0]
b=[50, 50, 80, 80]
dc.regionmask(mask='bigmask', boxes[a, b])
dc.clean(algorithm='hogbom', mask='bigmask', model='model.clean.masked', niter=1000)

Another example using rg.wbox function:

ia.open('dirty')
cs:=ia.coordsys()
rg.setcoordinates(cs.record())
r1:=dg.wbox(blc=['173pix', '347pix'], trc=['183pix', '370pix'])
c.regionmask(mask='bigmask', region=r1)

Or using a dict of regions:

r2=rg.wbox(blc=['180pix', '344pix'], trc=['191pix', '369pix'])
If quantities are to be used to define regions the following is an example:

deconvolver.clipimage.html

**deconvolver.clipimage - Function**

Zero all pixels where Stokes I is below a threshold

**Description**

All pixels in the image with Stokes I less than some threshold are set to zero. This is useful prior to self-calibration where one oftens wishes to remove negative pixels from the model. Note that if the image has polarization information, then the polarized part of a pixel is also set to zero if Stokes I is less than the threshold.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>name of clipped image</th>
</tr>
</thead>
<tbody>
<tr>
<td>clippedimage</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>inputimage</td>
<td>name of input image</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>threshold</td>
<td>Threshold</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant 0.0Jy</td>
</tr>
</tbody>
</table>


**Returns**

bool

**Example**

dc.clipimage(image='clean', threshold='50mJy')

###the ft that model into an MS and do gaincal for e.g
deconvolver.clarkclean.html

**deconvolver.clarkclean - Function**

3.1.2 Make a clean image using the Clark Clean a threshold

**Description**

In the Clark Clean algorithm, the cleaning is split into minor and major cycles. In the minor cycles only the brightest points are cleaned, using a subset of the point spread function. In the major cycle, the points thus found are subtracted correctly by using an FFT-based convolution.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>niter</td>
<td>Number of iterations</td>
</tr>
<tr>
<td>allowed: int</td>
<td>int</td>
</tr>
<tr>
<td>Default: 1000</td>
<td></td>
</tr>
<tr>
<td>gain</td>
<td>Loop Gain for CLEANing</td>
</tr>
<tr>
<td>allowed: double</td>
<td>double</td>
</tr>
<tr>
<td>Default: 0.1</td>
<td></td>
</tr>
<tr>
<td>threshold</td>
<td>Flux level at which to stop CLEANing</td>
</tr>
<tr>
<td>allowed: any</td>
<td>any</td>
</tr>
<tr>
<td>Default: variant 0Jy</td>
<td></td>
</tr>
<tr>
<td>displayprogress</td>
<td>Display the progress of the cleaning?</td>
</tr>
<tr>
<td>allowed: bool</td>
<td>bool</td>
</tr>
<tr>
<td>Default: false</td>
<td></td>
</tr>
<tr>
<td>model</td>
<td>Name of images</td>
</tr>
<tr>
<td>allowed: string</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>mask</td>
<td>Name of mask images used for CLEANing</td>
</tr>
<tr>
<td>allowed: string</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>histbins</td>
<td>Number of bins in the pixel-flux histogram</td>
</tr>
<tr>
<td>allowed: int</td>
<td>int</td>
</tr>
<tr>
<td>Default: 500</td>
<td></td>
</tr>
<tr>
<td>psfpatchsize</td>
<td>Size of PSF for minor cycle</td>
</tr>
<tr>
<td>allowed: intArray</td>
<td>intArray</td>
</tr>
<tr>
<td>Default: 51 51</td>
<td></td>
</tr>
<tr>
<td>maxextpsf</td>
<td>maximum external sidelobe, used to set depth of minor cycle clean</td>
</tr>
<tr>
<td>allowed: double</td>
<td>double</td>
</tr>
<tr>
<td>Default: 0.2</td>
<td></td>
</tr>
<tr>
<td>speedup</td>
<td>Cleaning speedup exponent</td>
</tr>
<tr>
<td>allowed: double</td>
<td>double</td>
</tr>
<tr>
<td>Default: 0.0</td>
<td></td>
</tr>
<tr>
<td>maxmumpix</td>
<td>Maximum number of pixels used in each minor cycle</td>
</tr>
<tr>
<td>allowed: int</td>
<td>int</td>
</tr>
<tr>
<td>Default: 10000</td>
<td></td>
</tr>
<tr>
<td>maxnummajcycles</td>
<td>Max number of major cycles; -1 = no restrictions</td>
</tr>
<tr>
<td>allowed: int</td>
<td>int</td>
</tr>
<tr>
<td>Default: -1</td>
<td></td>
</tr>
<tr>
<td>maxnumminoriter</td>
<td>Max number of minor iterations; -1 = no restrictions</td>
</tr>
<tr>
<td>allowed: int</td>
<td>int</td>
</tr>
<tr>
<td>Default: -1</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool
3.1.2 Make a clean image using the Clark Clean a threshold

Description

This is similar to clarkclean except that it accepts casa standard images. I.e it has to have 4 axes with the canonical order of (direction1, direction2, stokes, spectral). This function further will clean more than the quarter of the image if the mask coverage is larger than a quarter of the image size. It is useful for CLI level parallelization, i.e use imager to make a dirty image and psf and use this function to deconvolve. If in doubt use clarkclean

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>niter</td>
<td>Number of iterations</td>
<td>int</td>
<td>1000</td>
</tr>
<tr>
<td>gain</td>
<td>Loop Gain for CLEANing</td>
<td>double</td>
<td>0.1</td>
</tr>
<tr>
<td>threshold</td>
<td>Flux level at which to stop CLEANing</td>
<td>any</td>
<td>variant 0Jy</td>
</tr>
<tr>
<td>model</td>
<td>Name of model image that will contain the clean components</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>mask</td>
<td>Name of mask image used for CLEANing</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>cyclefactor</td>
<td>Factor to determine how deep to go in a Clark minor cycle</td>
<td>double</td>
<td>1.5</td>
</tr>
</tbody>
</table>

Returns

record
deconvolver.dirtyname.html

**deconvolver.dirtyname - Function**

3.1.2 Return the name of the dirty-image table

**Arguments**

**Returns**
string
deconvolver.psfname.html

**deconvolver.psfname** - Function

3.1.2 Return the name of the PSF-image table

**Arguments**

**Returns**

string
deconvolver.make.html

**deconvolver.make - Function**

3.1.2 Make an empty image

**Description**

Make an empty image with the properties (co-ordinate system etc.) borrowed from the dirty image.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>Name of the new image on the disk</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background?</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: false</td>
</tr>
</tbody>
</table>

**Returns**

bool
deconvolver.convolve.html

**deconvolver.convolve - Function**

3.1.2 Convolves an image with the PSF

---

**Description**

Convolves an image (e.g., the model image) with the PSF

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>convolvemodel</td>
<td>Name of the output image on the disk to hold the result of the convolution</td>
</tr>
<tr>
<td>model</td>
<td>The input image to be convolved with the PSF</td>
</tr>
</tbody>
</table>

| allowed: | string |
| Default: |

**Returns**

bool
**deconvolver.makegaussian.html**

**deconvolver.makegaussian - Function**

3.1.2 Make an image with a single gaussian component

**Description**

Make a model image with the a single gaussian. The properties of the output image (e.g. the co-ordinate system, etc.) are borrowed from the dirty image. The image is made as follows:

\[ I(x,y) = \text{Delta function of unit amplitude at (0,0)} \]
\[ \text{Temp}(x,y) = \text{Gaussian}(x,y,\text{Amplitude, Center, Sigma, PA}) \]
\[ I(x,y) = \text{Convolution of Temp}(x,y) \text{ with } I(x,y). \]

If `normalize=T` \[ I(x,y) = I(x,y)/(\text{area under the gaussian}). \]

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>gaussianimage</td>
<td>Name of the output image on the disk</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>bmaj</td>
<td>The major axis of the gaussian</td>
<td>any</td>
<td>variant 0rad</td>
</tr>
<tr>
<td>bmin</td>
<td>The minor axis of the gaussian</td>
<td>any</td>
<td>variant 0rad</td>
</tr>
<tr>
<td>bpa</td>
<td>The Position Angle of the gaussian</td>
<td>any</td>
<td>variant 0deg</td>
</tr>
<tr>
<td>normalize</td>
<td>Normalize the area under the gaussian to 1.0?</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background?</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

bool

1591
deconvolver.state.html

**deconvolver.state - Function**

3.1.2 Return the “state” of the tool

**Description**

Prints the name of the Dirty Image and the PSF and the parameters of the gaussian fitted to the main lobe of the PSF (the “Clean Beam”).

**Arguments**

**Returns**

bool
deconvolver.updatestate.html

**deconvolver.updatestate - Function**

3.1.2 [A GUI builders related function] Update the GUI to reflect the current state

**Description**

Updates the GUI to reflect the current state of the tool. This function is used by toolmanager. See documentation of the toolmanager for details about “methods” used to update the GUI.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>Glish variable for the GUI to be updated</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>method</td>
<td>The method to be used for updating</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>DONE</td>
</tr>
<tr>
<td>close</td>
<td>INIT</td>
</tr>
</tbody>
</table>

**Returns**

bool
3.1.2 Make a clean image with Hogbom or MultiScale Clean

**Description**

Makes a clean image using either the Hogbom or MultiScale algorithms. The MultiScale algorithm is the default. The clean is performed on the residual image calculated from the dirty image minus the point spread function convolved with the model currently selected. Thus if you want to restart a clean, simply set the model to the model that was previously produced by clean.

Rather than explicit CLEAN boxes, mask images are used to constrain the region that is to be deconvolved. To make mask images, use either boxmask (to define a mask via the corner locations blc and trc) or mask (to define a mask via thresholding an existing image). The default mask is the inner quarter of the image.

The CLEAN deconvolution is joint in whatever Stokes parameters are present. Thus it searches for peaks in either $I$ or $I + |V|$ or $I + \sqrt{Q^2 + U^2 + V^2}$, the rationale for the latter two forms being to be biased towards finding strongly polarized pixels first (these forms are also the maximum eigenvalue of the coherency matrix). The PSF is constrained to be the same in all polarizations (a feature of this implementation, not of the Hamaker-Bregman-Sault formalism).

The clean algorithms possible are:

**Hogbom** The classic algorithm: delta function units of emission are found iteratively by searching for the peak. Each point is subtracted from the full residual image using the shifted and scaled point spread function.

**Multi-Scale** As the Multi-Scale Clean algorithm is quite new, we provide extensive information on its use.

In the Multi-scale Clean, the image is cleaned simultaneously with several different beams given by the point spread function convolved with components of various shapes and sizes. The components we use in this implementation are upside-down paraboloids multiplied by first order spheroidal functions (i.e., the same functions used in gridding Fourier plane data). The paraboloids are truncated at zero, and the multiplication by the spheroidal function results in a smooth shape with minimal power at long baselines. This shape is scaled to the component sizes specified in setscales. As these functions have finite extent (unlike a Gaussian), they can easily be used with mask images.
For each iteration, the scale size which is able to subtract the most flux is chosen (but with a caveat, see below). The model is then built up out of the spheroidal functions of the various scale sizes. The scale sizes are set by the setscales function, which will permit the user to specify the scale sizes explicitly, or will optionally take the number of scale sizes to clean for and calculate the scale sizes themselves from a power law.

Most images deconvolved with Multi-scale Clean will be dominated by extended structure, and the largest scale size will initially remove the most flux from the dirty image. As the algorithm reduces the residuals on the largest scale, the residuals on the smaller scales will also be reduced, even without cleaning on those size scales (a falling sea sinks all boats). However, at some point, the residual image will be dominated by features on smaller size scales. These smaller features will be both positive and negative (i.e., to correct for the largest size scale being the wrong shape for the true emission features). Later in the algorithm, the magnitude of the residuals on all scales will be approximately equal. At this stage, most of the deconvolvable flux has been assimilated into the largest scale size components and detailed corrections to the large scale components must be made. At this point, the user may consider switching to a faster algorithm such as the Clark Clean.

Masking is fully available with Multi-scale Clean. No component is permitted to place any of its wings outside of the user-supplied mask. If the masking were based upon the different scale components’ center positions, then the large scale components could place their wings outside the mask, but the smaller scale components would not be able to make fine scale corrections. Hence, the Multi-scale Clean uses a different mask for each different size scale internally. If the mask is too restrictive or the scales are too large, the algorithm may not be able to fit the large scales into the mask at all, and the user is warned of this condition.

Traditional Clean algorithms use a small loop gain such as 0.1 to avoid confusing emission and sidelobes when extended emission is present. However, as MultiScale Clean can image large extended structure in a single spheroidal component, a loop gain in the range 0.5 to 1.0 can be used. If the largest residual oscillates between positive and negative with iteration number, as it can for some brightness distributions which include point sources, a lower loop gain will improve the imaging and the convergence.

A mild bias favoring cleaning small scale emission has been built into the Multi-scale algorithm. To illustrate the requirement of this bias, consider the case of a bright point source with very faint extended emission. Each scale may find its optimal component to subtract at the position of the bright point source, but each successively larger component will integrate more extended flux. Hence, the largest scale component will be removed from the residuals. If most of the flux were in the point source, then several smaller negative components must be subtracted from the largest
component, and finally the point component itself may be removed after
the extended emission has been taken care of. To prevent this situation
from occurring, we bias the selection of small-sized components.

Note that for all of these functions except fullmsclean, only a quarter of the
image may be cleaned. If no mask is set, then the cleaned region defaults to
the inner quarter. If a mask larger than a quarter of the image is set, then
only the quarter starting at the bottom left corner is used. Algorithm
fullmsclean will deconvolve the entire field. This is useful when performing a
limited accuracy deconvolution (as needed for example in wide-field imaging)
but will diverge if pushed too deep.
The clean threshold may be either absolute ('0.5Jy') or relative ('1%').

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>algorithm</strong> Algorithm to use</td>
</tr>
<tr>
<td>allowed: string</td>
</tr>
<tr>
<td>Default: fullmsclean</td>
</tr>
<tr>
<td>msclean</td>
</tr>
<tr>
<td>hogbom</td>
</tr>
<tr>
<td><strong>niter</strong> Number of Iterations, set to zero for no CLEANing</td>
</tr>
<tr>
<td>allowed: int</td>
</tr>
<tr>
<td>Default: 1000</td>
</tr>
<tr>
<td><strong>gain</strong> Loop Gain for CLEANing, try 0.7 for msclean or fullmsclean</td>
</tr>
<tr>
<td>allowed: double</td>
</tr>
<tr>
<td>Default: 0.1</td>
</tr>
<tr>
<td><strong>threshold</strong> Flux level at which to stop CLEANing</td>
</tr>
<tr>
<td>allowed: any</td>
</tr>
<tr>
<td>Default: variant 0Jy</td>
</tr>
<tr>
<td><strong>displayprogress</strong> Display progress</td>
</tr>
<tr>
<td>allowed: bool</td>
</tr>
<tr>
<td>Default: false</td>
</tr>
<tr>
<td><strong>model</strong> Name of images</td>
</tr>
<tr>
<td>allowed: string</td>
</tr>
<tr>
<td>Default:</td>
</tr>
<tr>
<td><strong>mask</strong> Name of mask images used for CLEANing</td>
</tr>
<tr>
<td>allowed: string</td>
</tr>
<tr>
<td>Default:</td>
</tr>
<tr>
<td><strong>async</strong> Run asynchronously in the background?</td>
</tr>
<tr>
<td>allowed: bool</td>
</tr>
<tr>
<td>Default: false</td>
</tr>
</tbody>
</table>

**Returns**

1596
Example

dc.clean(image='3C273XC1.clean.image', model='3C273XC1.clean.model', mask='3C283XC1.mask', niter=1000, gain=0.25, threshold=0.03)
deconvolver.naclean.html

**deconvolver.naclean - Function**

3.1.2 Make a clean image with Hogbom with self masking

**Description**

The clean is performed on the residual image calculated from the dirty image minus the point spread function convolved with the model currently selected. Thus if you want to restart a clean, simply set the model to the model that was previously produced by clean.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>niter</td>
<td>Number of Iterations, set to zero for no CLEANing</td>
</tr>
<tr>
<td>allowed: int</td>
<td></td>
</tr>
<tr>
<td>Default: 1000</td>
<td></td>
</tr>
<tr>
<td>gain</td>
<td>Loop Gain for CLEANing, try 0.7 for msclean or fullm-sclean</td>
</tr>
<tr>
<td>allowed: double</td>
<td></td>
</tr>
<tr>
<td>Default: 0.1</td>
<td></td>
</tr>
<tr>
<td>threshold</td>
<td>Flux level at which to stop CLEANing</td>
</tr>
<tr>
<td>allowed: any</td>
<td></td>
</tr>
<tr>
<td>Default: variant 0Jy</td>
<td></td>
</tr>
<tr>
<td>model</td>
<td>Name of images</td>
</tr>
<tr>
<td>allowed: string</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>mask</td>
<td>Name of mask images to return region cleaned</td>
</tr>
<tr>
<td>allowed: string</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>masksuppp</td>
<td>Number of pixels on each side of peak to set the memory mask</td>
</tr>
<tr>
<td>allowed: int</td>
<td></td>
</tr>
<tr>
<td>Default: 3</td>
<td></td>
</tr>
<tr>
<td>memory</td>
<td>this define memory function to use (none, weak, medium, strong)</td>
</tr>
<tr>
<td>allowed: string</td>
<td></td>
</tr>
<tr>
<td>Default: medium</td>
<td></td>
</tr>
<tr>
<td>numsigma</td>
<td>remember positions of peak above this number of sigmas</td>
</tr>
<tr>
<td>allowed: double</td>
<td></td>
</tr>
<tr>
<td>Default: 5.0</td>
<td></td>
</tr>
</tbody>
</table>
Returns
record

Example

dc.naclean(image='3C273XC1.clean.image', model='3C273XC1.clean.model',
mask='3C283XC1.mask', niter=1000, gain=0.25, threshold=0.03)
**deconvolver.setscales - Function**

3.1.2 Set the scale sizes for MultiScale Clean

**Description**

Set the scale sizes, all required PSF’s and Dirty Images for MultiScale Clean will be calculated. You can either give the number of scales, in which case the the scale sizes are set via a power law, or give a vector of scale sizes in pixels.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Method by which scales are set</th>
</tr>
</thead>
<tbody>
<tr>
<td>scalemethod</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>uservector</td>
</tr>
<tr>
<td>nscales</td>
<td>Number of scales</td>
</tr>
<tr>
<td>allowed</td>
<td>int</td>
</tr>
<tr>
<td>Default</td>
<td>5</td>
</tr>
<tr>
<td>uservector</td>
<td>Vector of scale sizes to use</td>
</tr>
<tr>
<td>allowed</td>
<td>doubleArray</td>
</tr>
<tr>
<td>Default</td>
<td>0.0 3.0 10.0</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```
dc.setscales(6);
```

1600
deconvolver.ft.html

**deconvolver.ft - Function**

3.1.2 Fourier transform the specified model

**Description**

Fourier transform the specified model to an image.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of image</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>Name of image</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>transform</td>
<td>Name of transform image</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background?</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: false</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

deco.ft(model='3C273XC1.nnls.model', transform='3C273XC1.nnls.model.ft')
deconvolver.restore.html

**deconvolver.restore** - Function

3.1.2 Restore the residuals

**Description**

Restore the residuals to a smoothed version of the model. The model images are convolved with the specified Gaussian beam and then the residual images are added. If the beam is not supplied, one will be fit to the PSF.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>Name of input model</td>
</tr>
<tr>
<td>image</td>
<td>Name of output restored image</td>
</tr>
<tr>
<td>bmaj</td>
<td>Major axis of beam</td>
</tr>
<tr>
<td>bmin</td>
<td>Minor axis of beam</td>
</tr>
<tr>
<td>bpa</td>
<td>Position angle of beam</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

1602
deco.restore(model='3C273XC1.clean', image='3C273XC1.clean.restored',
bmaj='2.0arcsec', bmin='2.0arcsec')
**deconvolver.residual.html**

**deconvolver.residual - Function**

### 3.1.2 Find the residuals

**Description**

Calculate the residuals corresponding to the model componentlist.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>Names of input models</td>
</tr>
<tr>
<td><strong>allowed:</strong> string</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>image</td>
<td>Names of output residual images</td>
</tr>
<tr>
<td><strong>allowed:</strong> string</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background</td>
</tr>
<tr>
<td><strong>allowed:</strong> bool</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```
decco.residual(model='3C273XC1.clean', complist='3C273XC1.cl', image='3C273XC1.clean.residual')
```
**deconvolver.smooth** - Function

3.1.2 smooth the image

**Description**

The model image is convolved with the specified Gaussian beam. By default (normalize=T), the beam volume is normalized to unity so that the smoothing is flux preserving. The smoothing used in restoration is not normalized.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of input model</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>Name of input model</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>image</td>
<td>Name of output smoothed image</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>bmaj</td>
<td>Major axis of beam</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant 0rad</td>
</tr>
<tr>
<td>bmin</td>
<td>Minor axis of beam</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant 0rad</td>
</tr>
<tr>
<td>bpa</td>
<td>Position angle of beam</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant 0deg</td>
</tr>
<tr>
<td>normalize</td>
<td>Normalize volume of psf to unity</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: true</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: false</td>
</tr>
</tbody>
</table>

**Returns**

bool
Example

- deco.smooth(model='3C273XC1.clean', image='3C273XC1.clean.restored', bmaj='2.0arcsec', bmin='2.0arcsec')

----------------------------------------------------------------------------
deconvolver.mem.html

**deconvolver.mem - Function**

3.1.2 Make the mem image

**Description**

Makes a mem image using the Cornwell-Evans algorithm, using either maximum entropy (entropy) or maximum emptiness (emptiness). The maximum entropy algorithm is the default. You can restart a MEM deconvolution on an existing model image, but the alpha and beta parameters are not yet saved.

Mask images can be used to restrict where the algorithm puts flux. A prior, or bias, image can provide a priori information to the algorithm and effectively limit the support as well as a mask. The prior image can be constructed by smoothing an existing estimate for the brightness distribution and clipping. Any pixel values below 1e-6 will be clipped to this level, so zero or negative pixels will not cause problems.

Currently, only one Stokes parameter may be deconvolved at a time. Stokes $I$ images can be deconvolved with either maximum entropy or maximum emptiness. Stokes $Q$, $U$, or $V$ should be deconvolved with maximum emptiness, which permits negative pixel values. Joint polarization MEM deconvolution is planned for the future.

The mem entropies possible are:

**entropy** The smoothness of the image, relative to some prior (also called default or bias) image is maximized. The functional form of the entropy is $H = \sum I \ln(I/M)$, where $I$ is the mem image brightness and $M$ is the prior image. As the prior image is positive definite, the entropy constrains the mem image pixels to be positive, hence only stokes $I$ can be imaged.

**emptiness** The number of pixels with absolute value of the flux greater than the noise level is minimized. This treats positive and negative pixel values equally, so it is appropriate for any Stokes image.

This MEM algorithm works in the image plane (ie, is ignorant of visibility data), but performs the convolution by multiplication in the Fourier plane. Not to be confused with this usage of the term "image plane", some problems are "image plane" problems, such as a single dish performing On-The-Fly mapping. Independent noise is added at each integration as the beam sweeps over the object (ie, in the image plane). This can lead to a noise signal at non-physically large spatial frequencies. This non-physical signal can be
removed by convolving the residual image with the PSF. Also key to this
problem is that the PSF is of finite extent, permitting the deconvolution of
nearly the entire dirty image rather than just the inner quarter. These options
are accessed by setting imageplane to T.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>entropy</td>
<td>entropy to use</td>
<td>string</td>
<td>emptiness</td>
</tr>
<tr>
<td>niter</td>
<td>Number of Iterations, set to zero for no MEMing</td>
<td>int</td>
<td>20</td>
</tr>
<tr>
<td>sigma</td>
<td>Noise level to try to achieve</td>
<td>any</td>
<td>0.001 Jy</td>
</tr>
<tr>
<td>targetflux</td>
<td>Total image flux to try to achieve</td>
<td>any</td>
<td>1.0 Jy</td>
</tr>
<tr>
<td>constrainflux</td>
<td>Use targetflux as a constraint? (or starting flux)</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>displayprogress</td>
<td>Display progress</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>model</td>
<td>Name of input/output model image</td>
<td>string</td>
<td>false</td>
</tr>
<tr>
<td>prior</td>
<td>Name of prior (default) image used for mem</td>
<td>string</td>
<td>false</td>
</tr>
<tr>
<td>mask</td>
<td>Mask image restricting emission (all pixels 0 or 1)</td>
<td>string</td>
<td>false</td>
</tr>
<tr>
<td>imageplane</td>
<td>Is this an image plane problem (like single dish)?</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background?</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

Returns

1608
Example

deco.mem(entropy='entropy', niter=30, sigma=0.01, targetflux=10.0,
model='3C273XC1.mem.image', prior='3C283XC1.prior')
deconvolver.makeprior - Function

3.1.2 Make the mem’s prior image, or make a mask

Description

Makes a prior image for the mem function. A general way to make a prior image is to start with a low resolution image, obtained from a smaller array configuration or a lower frequency observation, from another image which has been smoothed, or from a single dish image. The low resolution image can then be doctored via clipping and regioning to make it acceptable for the mem function. Currently, only one Stokes parameter may be used at a time.

Arguments
### Inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>prior</td>
<td>output prior image</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>templateimage</td>
<td>starting point for prior image</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>lowclipfrom</td>
<td>Clip any pixel below this level</td>
<td>any</td>
<td>variant 0.0Jy</td>
</tr>
<tr>
<td>lowclipto</td>
<td>Any clipped pixel will be given this value</td>
<td>any</td>
<td>variant 0.0Jy</td>
</tr>
<tr>
<td>highclipfrom</td>
<td>Clip any pixel above this level</td>
<td>any</td>
<td>variant 9e20Jy</td>
</tr>
<tr>
<td>highclipto</td>
<td>Any clipped pixel will be given this value</td>
<td>any</td>
<td>variant 9e20Jy</td>
</tr>
<tr>
<td>blc</td>
<td>Bottom left hand corner for box; outside box is clipped</td>
<td>intArray</td>
<td>-1</td>
</tr>
<tr>
<td>trc</td>
<td>Top right hand corner for box; outside box is clipped</td>
<td>intArray</td>
<td>-1</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background?</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

### Returns

bool

### Example

```python
decco.makeprior(prior='3C283XC1.prior', templateimage='3C283XC1.mem.smooth',
clipfrom='0.01Jy', clipto='0.0001Jy', blc=[100,100], trc=[150,150])
```
deconvolver.mtopen.html

**deconvolver.mtopen** - Function

### 3.1.2 Init : Make a series of images using a Multi-Term Clean algorithm

**Description**

Makes a series of images.
For N terms in the polynomial, supply a list of 2N-1 Hessian elements (psfs), and the scale sizes.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntaylor</td>
<td>Number of terms in the taylor polynomial</td>
<td>int</td>
<td>2</td>
</tr>
<tr>
<td>scalevector</td>
<td>Vector of scale sizes to use</td>
<td>doubleArray</td>
<td>0.0 3.0 10.0</td>
</tr>
<tr>
<td>psfs</td>
<td>Intpu : List of names of 2N-1 psfs. This is valid only for a Taylor-polynomial model.</td>
<td>stringArray</td>
<td></td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background?</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

xxx
### deconvolver.mtclean - Function

**3.1.2** Make a series of images using a Multi-Term Clean algorithm

#### Description

Makes a series of images.
Supply a list of N residual images and a corresponding list of 2N-1 Hessian elements (psfs).
This way, for each field or partial-run, one can choose how many terms to operate with, changing this number depending on the signal-to-noise level.

#### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>residuals</td>
<td>Input : List of names of N residual images</td>
</tr>
<tr>
<td>models</td>
<td>Output : List of names of N model images</td>
</tr>
<tr>
<td>niter</td>
<td>Number of Iterations, set to zero for no CLEANing</td>
</tr>
<tr>
<td>gain</td>
<td>Loop Gain for CLEANing, try 0.7 for msclean or fullmsclean</td>
</tr>
<tr>
<td>threshold</td>
<td>Flux level at which to stop CLEANing</td>
</tr>
<tr>
<td>displayprogress</td>
<td>Display progress</td>
</tr>
<tr>
<td>mask</td>
<td>Name of mask images used for CLEANing</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background?</td>
</tr>
</tbody>
</table>

**Inputs**
- residuals: Input : List of names of N residual images
  - allowed: stringArray
  - Default: 
- models: Output : List of names of N model images
  - allowed: stringArray
  - Default: 
- niter: Number of Iterations, set to zero for no CLEANing
  - allowed: int
  - Default: 1000
- gain: Loop Gain for CLEANing, try 0.7 for msclean or fullmsclean
  - allowed: double
  - Default: 0.1
- threshold: Flux level at which to stop CLEANing
  - allowed: any
  - Default: variant 0Jy
- displayprogress: Display progress
  - allowed: bool
  - Default: false
- mask: Name of mask images used for CLEANing
  - allowed: string
  - Default: 
- async: Run asynchronously in the background?
  - allowed: bool
  - Default: false
Returns
record

Example

xxx
deconvolver.mtrestore - Function

3.1.2 Restore the Multi-Term residuals

Description

The model images are smoothed by the specified Gaussian beam. The principal solution is computed from the residuals. The smoothed models are added to these new residuals. This ensures that any undeconvolved source flux has been transformed into the Taylor-coefficient basis, before being added into the smoothed Taylor-coefficient model images. (If the beam is not supplied, one will be fit to the PSF).

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>models</td>
<td>Input : Name of input model</td>
<td>stringArray</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>residuals</td>
<td>Input : Name of residual image</td>
<td>stringArray</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>images</td>
<td>Output : Name of output restored image</td>
<td>stringArray</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bmaj</td>
<td>Major axis of beam</td>
<td>any</td>
<td>variant 0rad</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bmin</td>
<td>Minor axis of beam</td>
<td>any</td>
<td>variant 0rad</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bpa</td>
<td>Position angle of beam</td>
<td>any</td>
<td>variant 0deg</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

Returns

1615
bool

Example

deco.restore(model='3C273XC1.clean', image='3C273XC1.clean.restored',
bmaj='2.0arcsec', bmin='2.0arcsec')
deconvolver.mtcalcpowerlaw.html

**deconvolver.mtcalcpowerlaw - Function**

3.1.2 Interpret Taylor coefficients as a power law, and compute spectral index

**Description**

Take ratios of restored images to compute alpha and beta

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>images</td>
<td>Input : Names of input restored images</td>
<td>stringArray</td>
<td></td>
</tr>
<tr>
<td>residuals</td>
<td>Input : Names of input residuals images (for error calcs)</td>
<td>stringArray</td>
<td></td>
</tr>
<tr>
<td>alphaname</td>
<td>Output : Name of output spectral-index image</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>betaname</td>
<td>Output : Name of output spectral-curvature image</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>threshold</td>
<td>Threshold</td>
<td>any</td>
<td>variant 0.0Jy</td>
</tr>
<tr>
<td>calcerro</td>
<td>Calculate an error image for spectral index</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>async</td>
<td>Run asynchronously in the background</td>
<td>bool</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

1617
deco.restore(model='3C273XC1.clean', image='3C273XC1.clean.restored', bmaj='2.0arcsec', bmin='2.0arcsec')
3.1.3  linearmosaic - Tool

combining images in a weighted fashion

Requires:

Synopsis

Description

The linearmosaic tool (lm) provides a toolkit for stitching images in a weighted fashion.

The default linear mosaic equation is defined by Equation (6) in Cornwell, Holdaway et al (Astronomy and Astrophysics, Vol. 271, p. 697 (1993)).

\[
I_{lm}(\theta) = \frac{\sum_p A_p(\theta)(I_p(\theta)A_p(\theta))w_p}{\sum_p A_p^2(\theta)w_p}
\]  

(3.1)

where \( A_p(\theta) \) is the primary beam (PB) of a given pointing \( p \), \( w_p \) is a sensitivity weight and the image of that pointing is \( I_p(\theta) \); the linear mosaic being \( I_{lm}(\theta) \)

Methods

- linearmosaic: Construct a linearmosaic tool
- defineoutputimage: Set the output direction image parameters and name
- setoutputimage: Use this function to mosaic on an existing image and weight image
- saultweightimage: Use this function to create a Sault-weighted image
- setlinmostype: Use this function to set if you want to mosaic in flatnoise domain
- makemosaic: mosaic images in a weighted fashion onto output image
linearmosaic.linearmosaic - Function

Construct a linearmosaic tool

Description

Create a `linearmosaic` tool.

Arguments

Returns

`linearmosaic`

Example

```r
lm=casac.linearmosaic()
```
linearmosaic.defineoutputimage - Function

3.1.3 Set the output direction image parameters and name

Description

Define the direction axes output image parameters. The output image will get the same number of spectral and polarization planes as the input images. This function creates a fresh new output image. If an image of the same name exists on disk it will be erased. The spectral and polarization part of the image will be identical to the images that are being mosaiced.

The output image will by default be flux correct and the weight image will be $\sum_p A_p^2(\theta)$ where the primary beam is $A_p(\theta)$

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nx</td>
<td>Total number of spatial pixels in x</td>
</tr>
<tr>
<td>allowed: int</td>
<td>int</td>
</tr>
<tr>
<td>Default: 128</td>
<td></td>
</tr>
<tr>
<td>ny</td>
<td>Total number of spatial pixels in y</td>
</tr>
<tr>
<td>allowed: int</td>
<td>int</td>
</tr>
<tr>
<td>Default: -1</td>
<td></td>
</tr>
<tr>
<td>cellx</td>
<td>Cellsize in x (e.g. '1arcsec')</td>
</tr>
<tr>
<td>allowed: any</td>
<td>any</td>
</tr>
<tr>
<td>Default: variant 1.0</td>
<td></td>
</tr>
<tr>
<td>celly</td>
<td>Cellsize in y (e.g. '1arcsec')</td>
</tr>
<tr>
<td>allowed: any</td>
<td>any</td>
</tr>
<tr>
<td>Default: variant</td>
<td></td>
</tr>
<tr>
<td>imagecenter</td>
<td>Direction of phase center as a direction measure</td>
</tr>
<tr>
<td>allowed: any</td>
<td>any</td>
</tr>
<tr>
<td>Default: variant 0</td>
<td></td>
</tr>
<tr>
<td>outputimage</td>
<td>output image name</td>
</tr>
<tr>
<td>allowed: string</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>outputweight</td>
<td>output weight image name</td>
</tr>
<tr>
<td>allowed: string</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>
Returns
bool

Example

```
lm.defineoutputimage(nx=3000, cellx='3arcsec', imagecenter='19h13m13.469 4d50m09.649', outputimage='test.linmos', outputweight='test.weightlinmos')
```
linemosaic.setoutputimage.html

**linemosaic.setoutputimage - Function**

3.1.3 Use this function to mosaic on an existing image and weight image

**Description**

Use this function if the mosaicing is to be done onto a previous mosaic or image. For now the stokes and spectral characteristic of the images to be mosaic and the output image has to be similar (i.e the user has to regrid them prior to linemosaic if necessary). The weightimage represents the sensitivity image of the image (for example the weighted primary beam coverage of a mosaic)

- **imageweighttype parameter:**
  If the image is of the type that has been normalized to be flux correct then the imageweighttype should be 0. If the image has been apodized by a primary beam then imageweighttype should be 1 and if the image is multiplied by $PB^2$ then it should be 2.

- **weighttype parameter:** This should be 1 if the weight image is the sum of Primary beams or equivalent and it should be 2 if it is the sum of $PB^2$

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputimage</td>
<td>Existing mosaic imagename</td>
<td><strong>allowed:</strong> string <strong>Default:</strong></td>
</tr>
<tr>
<td>outputweight</td>
<td>Existing weight image for mosaic</td>
<td><strong>allowed:</strong> string <strong>Default:</strong></td>
</tr>
<tr>
<td>imageweighttype</td>
<td>what kind of weight was already applied to image 0: flux correct 1: Primary beam equivalent weight 2: $PB^2$ equivalent weight</td>
<td><strong>allowed:</strong> int <strong>Default:</strong> 1</td>
</tr>
<tr>
<td>weighttype</td>
<td>what kind of weight is weight image 1: Primary beam equivalent weight 2: $PB^2$ equivalent</td>
<td><strong>allowed:</strong> int <strong>Default:</strong> 1</td>
</tr>
</tbody>
</table>

**Returns**

1623
bool

**Example**

```python
lm.setoutputimage(outputimage='test0.linmos', outputweight='test0.linmos.weight', imageweight=0, weighttype=2)
```
**Description**

"Sault weighted" image is one which is more pleasant to view (without high noise at the edges of mosaic images), it is flux correct upto a where the beam coverage becomes low and is tapered off onwards just to keep the noise from rising in the overall image (see Eq[2] from Sault, Staveley-Smith and Brouw (1996), Astron. Astrophys. Suppl, 120, 375)

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputimage</td>
<td>Name of image to create the Sault-Weighted image</td>
</tr>
<tr>
<td>fracpeak</td>
<td>Upto what fraction of peak of coverage should the image be flux correct</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>allowed</th>
<th>double</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default:</td>
<td>0.1</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```plaintext
lm.defineoutputimage(nx=3000, cellx='3arcsec', imagecenter='19h13m13.469 4d50m09.649', outputimage='test.linmos')
lm.makemosaic(images=['FIELD_31/img.image', 'FIELD_33/img.image'], weightimages=['FIELD_31/img.pb', 'FIELD_33/img.pb'])

###the above will make a flux correct mosaic of the 2 images in a flux correct fashion in test.linmos

#Now make a Sault weighted image to use in the viewer
lm.saultweightimage('test_sault.linmos')
```
Use this function to set if you want to mosaic in flatnoise domain.

**Description**

Use this function if the mosaicing is to be done using a non optimal weighting mode.

For now **optimal** (which is the default) follows this equation

\[
I_{lm}(\theta) = \frac{\sum_p A_p(\theta) (I_p(\theta) A_p(\theta)) w_p}{\sum_p A_p^2(\theta) w_p}
\]  

(3.2)

And **pbweight** follows this one

\[
I_{lm}(\theta) = \frac{\sum_p (I_p(\theta) A_p(\theta)) w_p}{\sum_p A_p(\theta) w_p}
\]  

(3.3)

where \( A_p(\theta) \) is the primary beam (PB) of a given pointing \( p \), \( w_p \) is a sensitivity weight and the image of that pointing is \( I_p(\theta) \); the linear mosaic being \( I_{lm}(\theta) \). For now \( w_p = 1 \)

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Type of weighting for linear mosaic</th>
</tr>
</thead>
<tbody>
<tr>
<td>linmostype</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: optimal</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**
linearmosaic.makemosaic.html

**linearmosaic.makemosaic - Function**

3.1.3 mosaic images in a weighted fashion onto output image

**Description**

Put the list of images onto the mosaic image using the weight images

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of images to mosaic</th>
</tr>
</thead>
<tbody>
<tr>
<td>images</td>
<td>allowed: any Default: variant</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of images to mosaic</th>
</tr>
</thead>
<tbody>
<tr>
<td>weightimages</td>
<td>allowed: any Default: variant</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inputs</th>
<th>what kind of weight is already applied to image 0: flux correct 1: Primary beam equivalent weight 2: PB² equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>imageweighttype</td>
<td>allowed: int Default: 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inputs</th>
<th>what kind of weight image it is 1: Primary beam equivalent weight 2: PB² equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>weighttype</td>
<td>allowed: int Default: 1</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```
lm.defineoutputimage(nx=3000, cellx='3arcsec', imagecenter='19h13m13.469 4d50m09.649', outputimage='test.linmos')
```

1627
### make a mosaic of two fields

```python
lm.makemosaic(images=['FIELD_31/img.image', 'FIELD_33/img.image'], weightimages=['FIELD_31/img.pb', 'FIELD_33/img.pb'])
```

### now we have another submosaic (which is flux corrected) we want to add to this mosaic

```python
lm.makemosaic(images='submosaic.image', weightimages='submosaic.flux', imageweighttype=0, weighttype=1)
```

---

**3.2 table - Module**

Casapy interface to table system

**Description** CASA stores all data inside CASA tables which are stored on disk. A CASA table consists of an unlimited number of columns of data, with optional column keywords and optional table keywords. Columns are named and rows are numbered (starting at 0). The columns hold data, such as visibilities and uv coordinates, while the keywords hold general information such as units or revision numbers or table author or even other tables.

To make this concrete, examples of columns might be:

<table>
<thead>
<tr>
<th>U</th>
<th>V</th>
<th>W</th>
<th>TIME</th>
<th>ANT1</th>
<th>ANT2</th>
<th>VISIBILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>124.011</td>
<td>54560.0</td>
<td>3477.1</td>
<td>43456789.0990</td>
<td>1</td>
<td>2</td>
<td>4.327 -0.1132</td>
</tr>
<tr>
<td>34561.0</td>
<td>45629.3</td>
<td>3900.5</td>
<td>43456789.0990</td>
<td>1</td>
<td>3</td>
<td>5.398 0.4521</td>
</tr>
</tbody>
</table>

and examples of keywords might be:

```
REVISION=2.01
AUTHOR="Tim Cornwell"
INSTRUMENT="VLA"
```

Everything in a CASA table (and thus all data stored in CASA) is potentially accessible and changable from casapy. The table module provides a convenient way of accessing and changing CASA tables from inside casapy. To do so one uses `tb`, the table tool, inside casapy. The table tool provides functions for:

- Opening and copying of existing tables, (using `open` and `copy`).
- Get and put of table cells, columns and keywords,
- Selection and sorting with TaQL (using the query or calc functions),
- Get and put of table information strings,
- Browsing of tables (using the browse function),
- Printing of a summary of a table (using summary function),
- Reading or writing a table from or to an ASCII format (using fromascii and toasciifmt).

All operations are done inside the casapy client and are not written to disk until an explicit flush command is performed.

The most typical operation on a CASA table is to open it, load a column from the table into casapy, alter it using casapy capabilities, and then write it back to the table. For this only a few commands are relevant: see the example below.

Sorting and selecting of tables is possible. The table thus produced is a reference table, and points back to the original table.

Example

```python
tb.open('3C273XC1.MS')
tb.summary();
uvw=tb.getcol("UVW");
tb.close()
tb.open("3C273XC1.MS/SPECTRAL_WINDOW")
freq=tb.getcell("REF_FREQUENCY", 0)
tb.close()
for i in range(len(uvw)):
    for j in range(len(uvw[0])):
        uvw[i][j] = uvw[i][j]*1.420E9/freq
tb.open('3C273XC1.MS', nomodify=False)
tb.putcol("UVW", uvw);
tb.close()
```

See Also
tableplot
3.2.1 table - Tool

Access tables from casapy
Requires:
Synopsis

Description

table is the tool that contains all the functions relevant for table handling.

Methods

- fromfits: Create a CASA table from a binary FITS file
- fromascii: Create a CASA table from a file containing data in ASCII format
- open: open an existing table
- create: create a new table
- flush: flush the current contents to disk
- fromASDM: Create an CASA table from an ASDM table
- resync: resync the table tool with table file
- close: close the table tool
- copy: copy a table
- copyrows: copy rows from this table to another
- done: end the table tool
- iswritable: is the table writable?
- endianformat: get the endian format used for this table
- lock: acquire a lock on the table
- unlock: unlock and flush the table
- datachanged: has data changed in table?
- haslock: has this process a lock on the table?
- lockoptions: get the lock options used for this table
- ismultiused: is the table in use in another process?
- browse: browse a table using a graphical browser
- name: return name of table on disk
- createmultitable: Create a virtually concatenated table
- toascii: Write CASA table into an ASCII format
- taql: Make a table from a TaQL command.
- query: Make a table from a query
- calc: TaQL expression with calc to calculate an expression on a table
- selectrows: Make a table from a selection of rows
get the info record
set the info record
add a readme line to the info record
summarize the contents of the table
return the names of the columns
!!!INPUT PARAMETERS IGNORED!!! return the row numbers in the (reference) table
set maximum cache size for column in the table
is the specified column scalar?
tell if column contains variable shaped arrays
return the column data type
return the column array type
return number of columns
return number of rows
add a specified number of rows
remove the specified rows
!!!REQUIRES COLUMN DESCRIPTION FUNCTIONS THAT HAVE NOT BEEN IMPLEMENTED!!! rename a column
remove one or more columns
test if a specific cell contains a value
get a specific cell
get a slice from a specific cell
get a specific column
get a specific column (for variable arrays)
get a slice from a specific columnarray
put a specific cell
put a slice into a specific cell
put a specific column
put a slice into a specific column
put a specific column (for variable arrays)
put a slice into a specific column
get shape of arrays in a specific column
get value of specific table keyword
get values of all table keywords
get value of specific column keyword
get values of all keywords for a column
put a specific table keyword
!!!BROKEN!!! put multiple table keywords
put a specific keyword for a column
put multiple keywords for a column
remove a specific table keyword
remove a specific keyword for a column
get the info about data managers
get the names of all table keywords
get the names of fields in a table keyword
get the names of all keywords in a column
get the names of fields in a keyword in a column
get the table description
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getcoldesc</code></td>
<td>get the description of a specific column</td>
</tr>
<tr>
<td><code>ok</code></td>
<td>Is the table tool ok?</td>
</tr>
<tr>
<td><code>clearlocks</code></td>
<td>Clears any table lock associated with the current process</td>
</tr>
<tr>
<td><code>listlocks</code></td>
<td>Lists any table lock associated with the current process</td>
</tr>
<tr>
<td><code>statistics</code></td>
<td>Get statistics on the selected table column</td>
</tr>
<tr>
<td><code>showcache</code></td>
<td>show the contents of the table cache</td>
</tr>
<tr>
<td><code>testincrsm</code></td>
<td>Checks consistency of an Incremental Store Manager bucket layout</td>
</tr>
</tbody>
</table>
Create a CASA table from a binary FITS file

Description

Create a table from binary FITS format. This generates a CASA table from the binary FITS table in the given HDU (header unit) of the FITS file. Note that other FITS formats (e.g. Image FITS and UVFITS) are read by other means.

It is possible to specify the storage manager to use for the table:
- `standard` is the default storage manager.
- `incremental` is efficient for slowly varying data.
- `memort` is for in memory use for e.g to grab given columns via getcol.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablename</td>
<td>Name of table to be created</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>fitsfile</td>
<td>Name of FITS file to be read</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>whichhdu</td>
<td>Which HDU to read (0-relative to primary HDU i.e. 1 is the smallest valid</td>
</tr>
<tr>
<td></td>
<td>value)</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 1</td>
</tr>
<tr>
<td>storage</td>
<td>Storage manager to use (standard or incremental or memory)</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: standard</td>
</tr>
<tr>
<td>convention</td>
<td>Convention to use (sdfits or none)</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: none</td>
</tr>
<tr>
<td>nomodify</td>
<td>Open Read-only?</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: true</td>
</tr>
<tr>
<td>ack</td>
<td>Acknowledge creations, etc</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: true</td>
</tr>
<tr>
<td>Returns</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>table</td>
<td></td>
</tr>
</tbody>
</table>
table.fromascii.html

**table.fromascii - Function**

3.2.1 Create a CASA table from a file containing data in ASCII format

**Description**

Create a table from an ASCII file. Columnar data as well as table and column keywords may be specified. Once the table is created from the ASCII data, it is opened in the specified mode by the table tool.

The table columns are filled from a file containing the data values separated by a separator (one line per table row). The default separator is a blank. Blanks after the separator are ignored.

If a non-blank separator is used, values can be empty. Such values default to 0, empty string, or F depending on the data type. E.g. 1,,2, has 4 values of which the 2nd and 4th are empty and default to 0. Similarly if fewer values are given than needed, the missing values get the default value.

Either the data format can be explicitly specified or it can be found automatically. The former gives more control in ambiguous situations. Both scalar and array columns can be generated from the ASCII input. The format string determines the type and optional shape.

In automatic mode (autoheader=True) the first line of the ASCII data is analyzed to deduce the data types. Only the types I, D, and A can be recognized. A number without decimal point or exponent is I (integer), otherwise it is D (double). Any other string is A (string). Note that a number may contain a leading sign (+ or -). The autoshape argument can be used to specify if the input should be stored as multiple scalars (the default) or as a single array. In the latter case one axis in the shape can be defined as variable length by giving it the value 0. It means that the actual array shape in a row is determined by the number of values in the corresponding input line.

Columns get the names Column1, Column2, etc..

For example:

1. autoshape=[] (which is the default) means that all values are to be stored as scalar columns.

2. autoshape=0 means that all values in a row are to be stored as a variable length vector.

3. autoshape=10 defines a fixed length vector. If an input line contains less than 10 values, the vector is filled with default values. If more than 10 values, the latter values are ignored.
4. autoshape=[5,0] defines a 2-dim array of which the 2nd axis is variable. Note that if an input line does not contain a multiple of 5 values, the array is filled with default values.

If the format of the table is explicitly specified, it has to be done either in the first two lines of the data file (named by the argument filename), or in a separate header file (named by the argument headerfile). In both forms, table keywords may also be specified before the column definitions. The column names and types can be described by two lines:

1. The first line contains the names of the columns. These names may be enclosed in quotes (either single or double).

2. The second line contains the data type and optionally the shape of each column. Valid types are:
   - S for Short data
   - I for Integer data
   - R for Real data
   - D for Double Precision data
   - X for Complex data (Real followed by Imaginary)
   - Z for Complex data (Amplitude then Phase)
   - DX for Double Precision Complex data (Real followed by Imaginary)
   - DZ for Double Precision Complex data (Amplitude then Phase)
   - A for ASCII data (a value must be enclosed in single or double quotes if it contains whitespace)
   - B for Boolean data (False are empty string, 0, or any string starting with F, f, N, or n).

If a column is an array, the shape has to be given after the data type without any whitespace. E.g. I10 defines an integer vector of length 10. A2,5 defines a 2-dim string array with shape [2,5]. Note that I is not the same as I1 as the first one defines a scalar and the other one a vector with length 1. The last column can have one variable length axis denoted by the value 0. It "consumes" the remainder of the input line.

If the argument headerfile is set then the header information is read from that file instead of the first lines of the data file.

To give a simple example of the form where the header information is located at the top of the data file:

<table>
<thead>
<tr>
<th>COLI</th>
<th>COLF</th>
<th>COLD</th>
<th>COLX</th>
<th>COLZ</th>
<th>COLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>R</td>
<td>D</td>
<td>X</td>
<td>Z</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>1.1</td>
<td>1.11</td>
<td>1.12</td>
<td>1.13</td>
<td>1.14 1.15 Str1</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15    16   &quot;&quot;</td>
</tr>
</tbody>
</table>

1636
Note that a complex number consists of 2 numbers.
Also note that an empty string can be given.
Let us now give an example of a separate header file that one might use to get interferometer data into CASA:

```
U  V  W  TIME  ANT1  ANT2  DATA
R  R  R  D  I   I   X1,0
```

The data file would then look like:

```
124.011 54560.0 3477.1 43456789.0990 1 2 4.3270 -0.1132
34561.0 45629.3 3900.5 43456789.0990 1 3 5.3980 0.4521
```

Note that the DATA column is defined as a 2-dim array of 1 correlation and a variable number of channels, so the actual number of channels is determined by the input. In this example both rows will have 1 channel (note that a complex value contains 2 values).

Tables may have keywords in addition to the columns. The keywords are useful for holding information that is global to the entire table (such as author, revision, history, etc).

The keywords in the header definitions must precede the column descriptions. They must be enclosed between a line that starts with ".key..." and a line that starts with ".endkey..." (where ... can be anything). Between these two lines each line should contain the following as listed below. A table keyword set and column keyword sets can be specified. The latter can be specified by specifying the column name after the .keywords string.

- The keyword name, e.g., ANYKEY
- The datatype and optional shape of the keyword (cf. list of valid types above)
- The value or values for the keyword (the keyword may contain a scalar or an array of values). e.g., 3.14159 21.78945

Thus to continue the example above, one might wish to add keywords as follows:

```
.DATE A "97/1/16"
.REVISION D 2.01
.AUTHOR A "Tim Cornwell"
.INSTRUMENT A "VLA"
```

Thus to continue the example above, one might wish to add keywords as follows:

```
.DATE A "97/1/16"
.REVISION D 2.01
.AUTHOR A "Tim Cornwell"
.INSTRUMENT A "VLA"
```

```
U  V  W  TIME  ANT1  ANT2  DATA
R  R  R  D  I   I   X1,0
```

1637
Similarly to the column format string, the keyword formats can also contain shape information. The only difference is that if no shape is given, a keyword can have multiple values (making it a vector).

It is possible to ignore comment lines in the header and data file by giving the `commentmarker`. It indicates that lines starting with the given marker are ignored. Note that the marker can be a regular expression (e.g. `texttt' *//'` tells that lines starting with `//` and optionally preceded by blanks have to be ignored).

With the arguments `firstline` and `lastline` one can specify which lines have to be taken from the input file. A negative value means 1 for `firstline` or end-of-file for `lastline`. Note that if the headers and data are combined in one file, these line arguments apply to the whole file. If headers and data are in separate files, these line arguments apply to the data file only. Also note that ignored comment lines are counted, thus are used to determine which lines are in the line range.

The number of rows is determined by the number of lines read from the data file.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>tablename</td>
<td>Name of table to be created</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>asciifile</td>
<td>Name of ASCII file to be read</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>headerfile</td>
<td>Name of an optional file defining the format</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>autoheader</td>
<td>Determine header information automatically</td>
<td>bool</td>
<td>false</td>
</tr>
<tr>
<td>autoshape</td>
<td>Shape to be used if autoheader=True</td>
<td>intArray</td>
<td>-1</td>
</tr>
<tr>
<td>sep</td>
<td>Value separator</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>commentmarker</td>
<td>Regex indicating comment line</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>firstline</td>
<td>First line to use</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>lastline</td>
<td>Last line to use</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>nomodify</td>
<td>Open Read-only?</td>
<td>bool</td>
<td>true</td>
</tr>
<tr>
<td>columnnames</td>
<td>Column Names</td>
<td>stringArray</td>
<td></td>
</tr>
<tr>
<td>datatypes</td>
<td>Data types</td>
<td>stringArray</td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool
table.open.html

**table.open - Function**

3.2.1 open an existing table

**Description**

Opens a disk file containing an existing CASA Table. Most of the time you just need to specify the tablename and perhaps nomodify. A table can be shared by multiple processes by using the appropriate locking options. The possible options are:
- auto: let the system take care of locking. At regular time intervals these autolocks are released to give other processes the opportunity to access the table.
- autonoread: as auto, but no read locking is needed. This must be used with care, because it means that reading can be done while the table tool is not synchronized with the table file (as is normally done when a lock is acquired). The function `resync` can be used to explicitly synchronize the table tool
- user: the user takes care by explicit calls to lock and unlock
- usernoread: as user and the no readlocking behaviour of autonoread.
- permanent: use a permanent lock; the constructor fails when the table is already in use in another process
- permanentwait: as above, but wait until the other process releases its lock
- default: this is the default option. If the given table is already open, the locking option in use is not changed. Otherwise it reverts to auto.

When auto locking is used, it is possible to give a record containing the fields option, interval, and/or maxwait. In this way advanced users have full control over the locking options. In practice this is hardly ever needed.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tablename</td>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>lockoptions</td>
<td>locking dictionary to be used: dict keys are 'option', 'interval', 'maxwait'</td>
<td>allowed: record</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>nomodify</td>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td>true</td>
</tr>
</tbody>
</table>
Returns

bool

Example

# First let's make a table for testing
def maketesttable():
    # Get path to CASA home directory by stripping name from '$CASAPATH'
    pathname=os.environ.get("CASAPATH").split()[0]
    # This is where the 3C273XC1.fits data should be
    fitsdata=pathname+"/data/demo/3C273XC1.fits"
    # Remove old table if present
    !rm -rf 3C273XC1.MS
    ms.fromfits("3C273XC1.MS",fitsdata)
    ms.close()

maketesttable()
tb.open("3C273XC1.MS")
tb.browse()
tb.close()

The first line opens an existing table 3C273XC1.MS, the second browses it using the browse function.

    tb.open("3C273XC1.MS", nomodify=False, lockoptions={"option":"user"})
    tb.lock();
    tb.addrows();
    tb.unlock();

In this example explicit user locking is used. The function lock is needed to acquire a (write) lock before the addrows is done. Thereafter the lock is released to give other processes the chance to operate on the table.
\Note that releasing a lock implies flushing the table, so doing that very often can be quite expensive.
table.create.html

**table.create - Function**

### 3.2.1 create a new table

**Description**

Create a new CASA Table. Most of the time you just need to specify the table’s name and a description of its format.

A table can be shared by multiple processes by using the appropriate locking options. The possible options are:

- **auto**: let the system take care of locking. At regular time intervals these autolocks are released to give other processes the opportunity to access the table.
- **autonoread**: as auto, but no read locking is needed. This must be used with care, because it means that reading can be done while the table tool is not synchronized with the table file (as is normally done when a lock is acquired). The function `resync` can be used to explicitly synchronize the table tool.
- **user**: the user takes care by explicit calls to lock and unlock
- **usernoread**: as user and the no readlocking behaviour of autonoread.
- **permanent**: use a permanent lock; the constructor fails when the table is already in use in another process
- **permanentwait**: as above, but wait until the other process releases its lock
- **default**: this is the default option. If the given table is already open, the locking option in use is not changed. Otherwise it reverts to auto.

When auto locking is used, it is possible to give a record containing the fields option, interval, and/or maxwait. In this way advanced users have full control over the locking options. In practice this is hardly ever needed.

**Arguments**
Inputs

tablename
  allowed: string
  Default:
tabledesc
description of the table’s format
  allowed: record
  Default:
lockoptions
locking to be used
  allowed: record
  Default: default
endianformat
  allowed: string
  Default:
memtype
  allowed: string
  Default:
nrow
  allowed: int
  Default: 0
dminfo
  Data Manager information
  allowed: record
  Default:

Returns

bool

Example

# First let’s get sample descriptions of a table and its data managers.
import os, shutil
def get_tabledesc_and_dminfo(tabname="3C273XC1.MS"):
    made_copy = False

    # Fetch new table if tabname not present
    if not os.path.isdir(tabname):
        # Get path to CASA root directory by stripping name from '$CASAPATH'
        pathname = os.environ.get("CASAPATH").split()[0]
# There should be some data here
fitsdata = pathname + "/data/demo/3C273XC1.fits"
tabname = "3C273XC1.MS"

ms.fromfits(tabname, fitsdata)
ms.close()
made_copy = True

tb.open(tabname)
tabdesc = tb.getdesc()
dminfo = tb.getdminfo()
print tabname, "has", tb.nrows(), "rows."
tb.close()

# Clean up
if made_copy:
    shutil.rmtree(tabname)

return tabdesc, dminfo

tabdesc, dmi = get_tabledesc_and_dminfo()
tabdesc # prints tabdesc
dmi # prints dmi

# You could alter tabdesc and/or dmi at this point.

# Unnecessary, but just to show there is nothing up my sleeve...
tb.close()

tb.create("myempty.ms", tabdesc, dminfo=dmi)
tb.nrows() # OL
tb.addrows(5) # Add the rows _before_ filling the columns.
tb.putcol('ARRAY_ID', numpy.array([0 for i in range(5)]))
tb.putcol('ANTENNA1', numpy.array(range(5)))
tb.putcol('ANTENNA2', numpy.array(range(1,6)))
tb.browse() # Still mostly, but not completely, empty.
tb.close()

This creates a CASA table using a description of a table and its data managers from an existing MS.

1645
table.flush.html

**table.flush - Function**

3.2.1 flush the current contents to disk

**Description**

Until a flush is performed, the results of all operations are not reflected in any change to the disk file. Hence you *must* do a flush to write the changes to disk.

**Arguments**

**Returns**

bool
table.fromASDM.html

**table.fromASDM - Function**

3.2.1 Create an CASA table from an ASDM table

**Description**

The main function for this task is to create a CASA::Table from a XML ASDM Table. The classes asdmCasaXMLUtil and asdmCasaSaxHandler are the main objects which implement the task. The asdmCasaSaxHandler encapsulate all the operations returning a reference to a CASA::Table. The class uses xerces-c to parse the XML table and creates the CASA::Table. The implementation assumes the integrity of the XML data, it not attempting to check whether the XML data meets a column format or not. In detail, an ArrayString column should agree with the following format: nd nx ... data, where nd is the number of dimensions, nx is the size of the first dimension (implemented upto a cube, i.e. nx,ny,nz), and data is the array itself which should have the appropriate number of elements. For example, a VectorString column could be: 1 2 "I" "Q" or dimension 1, size 2, and two string elements. Due to the lack of data type specification in the XML tables, the column names are hardcoded into the asdmCasaSaxHandler based on the ASDM specification (see http://aramis.obspm.fr/~alma/ASDM/ASDMEntities/index.html). While missing data from a table column will be accepted by the task, any new column beyond the specification has to be added into the class, also, any change in data types form the specification will produce a crash, CASA is picky with data types integrity. So far, the list of tables included in the class is: AlmaCorrelatorMode.xml, Antenna.xml, ConfigDescription.xml, DataDescription.xml, ExecBlock.xml, Feed.xml, Field.xml, Main.xml, Polarization.xml, Processor.xml, Receiver.xml, SBSummary.xml, Scan.xml, Source.xml, SpectralWindow.xml, State.xml, Station.xml, Subscan.xml, SwitchCycle.xml, CalCurve.xml, CalData.xml, CalPhase.xml more tables will follow. The usage of fromASDM is simple, it gets two string, tablename and xmlfile, where tablename is the CASA::Table to be written and xmlfile represents the ASDM XML table. To call it do:

tb.fromasdm(tablename,xmlfile)

**Arguments**
Inputs

<table>
<thead>
<tr>
<th>tablename</th>
<th>Name of table to be created</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>xmlfile</td>
<td>Name of the XML file to be read</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

bool
table.resync.html

**table.resync - Function**

resync the table tool with table file

**Description**

Acquiring a read or write lock automatically synchronizes the internals of the table tool with the actual contents of the table files. In this way different processes accessing the same table always use the same table data. However, a table can be used without read locking. In that case the table tool internals are not synchronized automatically. The resync function offers a way to do explicit synchronization. It is only useful if the table is opened with locking mode `autonoread` or `usernoread`.

**Arguments**

**Returns**

bool
table.close.html

**table.close - Function**

3.2.1 close the table tool

**Description**

First a flush is done, then the table is closed inside casapy and is no longer available for use.

**Arguments**

**Returns**

bool
table.copy.html

**table.copy - Function**

3.2.1 copy a table

**Description**

Copy the table. All subtables are also copied. References to another table are preserved.

The argument **deep** determines how a reference table (i.e. the result of a query) is copied. By default a file copy is made, thus the resulting table still contains references and no actual data. If, however, **deep=True** is given, a deep copy is made which means that the actual data are copied. Also all subtables are copied.

Normally a plain table is copied by copying the files. However, if **deep=True** and **valuecopy=True** are given, a plain table is copied by copying all its values and subtables. This is useful to reorganize the tables, i.e. to regain file space that is wasted by frequent updates to a table.

The argument **dminfo** can be used to specify explicit data manager info for the columns in the new plain table. It can be used to change, for example, a storage manager from IncrStMan to StandardStMan. The **dminfo** is a record as returned by the getdminfo. If **dminfo** is a non-empty record, it forces **valuecopy=True**.

The standard operation is make the copy to a plain table. It is, however, possible to copy to a memory table by giving **memorytable=True**.

The endian format for the newly created table can be specified. This is only meaningful if a deep copy is made to a plain table. The possible values are:
- big: big endian format (as used on e.g. SUN)
- little: little endian format (as used on e.g. PC)
- local: use the endian format of the machine being used
- aipsrc: use the endian format specified in aipsrc variable table.endianformat (which defaults to big).

The default is aipsrc.

Normally the **copy** function only copies the table and does not create a new table tool object. The user can do that by opening the newly created table in the standard way. However, it is possible to get an object back by using **returnobject=True**. An object is always returned if the copy is made to a memory table.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>newtablename</code></td>
<td>Name of new table on disk</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><code>deep</code></td>
<td>Make a deep copy of a reference table?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td><code>valuecopy</code></td>
<td>Make a deep copy of any table?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td><code>dminfo</code></td>
<td>Data manager info for new table</td>
</tr>
<tr>
<td>allowed:</td>
<td>record</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><code> endian</code></td>
<td>Endian format of new table</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>aipsrc</td>
</tr>
<tr>
<td><code>memorytable</code></td>
<td>Hold new table in memory?</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td><code>returnobject</code></td>
<td>Return a tool object for the new table</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
<tr>
<td><code>norows</code></td>
<td>Don’t copy any rows (useful for copying only the table structure)</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returns</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>table</code></td>
<td></td>
</tr>
</tbody>
</table>
table.copyrows.html

**table.copyrows - Function**

3.2.1 copy rows from this table to another

**Description**

Copy rows from this table to another. By default all rows of this table are appended to the output table. It is possible though to control which rows are copied.

Rows are added to the output table as needed. Because no rows can be added to a reference table, it is only possible to overwrite existing rows in such tables. Only the data of columns existing in both tables will be copied. Thus by making a reference table consisting of a few columns, it is possible to copy those columns only.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Type</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outtable</td>
<td>table object of output table</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>startrowin</td>
<td>First row to take from input table</td>
<td>int</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>startrowout</td>
<td>First row to write in output table, -1 (=end)</td>
<td>int</td>
<td></td>
<td>-1</td>
</tr>
<tr>
<td>nrow</td>
<td>Nr of rows to copy, -1 (=all)</td>
<td>int</td>
<td></td>
<td>-1</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

1653
This example appends rows to the table itself, thus doubles the number of rows.

```python
tb.open('3C273XC1.MS', nomodify=False)
tb.copyrows('3C273XC1.MS')
tb.close()
```

This example copies 10 rows of the selected subset of the MS to the beginning of the output MS.

```bash
!rm -rf in.MS out.MS
ms.fromfits('in.MS', '3C273XC1.fits')  # Make two MSs
ms.fromfits('out.MS', '3C273XC1.fits')  # For example
ms.close()
tb.open('in.MS')
t1 = tb.query('ANTENNA1==0')
tb.close()
t1.copyrows('out.MS', nrow=10, startrowout=0)
t1.close()
```
table.done.html

**table.done - Function**

3.2.1 end the table tool

**Description**

Effectively a synonym for function close.

**Arguments**

**Returns**

bool
table.iswritable - Function

3.2.1 is the table writable?

Description

Test if the table is opened for write.

Arguments

Returns

bool
Get the endian format used for this table. It returns a string with value 'big' or 'little'.

Arguments

Returns

string
**table.lock - Function**

3.2.1 acquire a lock on the table

**Description**

Try to acquire a read or write lock on the table. Nothing will be done if the table is already correctly locked by this process. It is only needed when user locking is used. When the lock is acquired, the internal caches will be synchronized with the (possibly changed) contents of the table. It is possible to specify the number of attempts to do (1 per second) in case the table is locked by another process. The default 0 is trying indefinitely.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>write</td>
<td>Write lock? (F=read lock)</td>
</tr>
<tr>
<td>nattempts</td>
<td>Nr of attempts</td>
</tr>
</tbody>
</table>

- **write**
  - allowed: bool
  - Default: true

- **nattempts**
  - allowed: int
  - Default: 0

**Returns**

bool
table.unlock.html

### table.unlock - Function

3.2.1 unlock and flush the table

#### Description

The table is flushed and the lock on the table is released. This function is only needed when user locking is used. However, it is also possible to use it with auto locking. In that case the lock will automatically be re-acquired before the next table operation.

#### Arguments

#### Returns

bool
table.datachanged.html

**table.datachanged** - **Function**

Has data changed in table?

**Description**

This function tests if data in the table have changed (by another process) since the last call to this function.

**Arguments**

**Returns**

bool
table.haslock.html

**table.haslock - Function**

3.2.1 has this process a lock on the table?

**Description**

Has this process a read or write lock on the table?

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>write</td>
<td>Has it a write lock? (F=read lock)</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>true</td>
</tr>
</tbody>
</table>

**Returns**

bool
table.lockoptions - Function

get the lock options used for this table

Description

Get the lock options used for this table. It returns a record with the fields: option, interval and maxwait. The record can be used as the lockoptions argument when opening a table.

Arguments

Returns

record
table.ismultiused.html

**table.ismultiused - Function**

3.2.1 is the table in use in another process?

**Description**

Is the table still in use in another process? If so, the table cannot be deleted.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>check if subtables are multiused?</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

bool
table.browse.html

**table.browse - Function**

3.2.1 browse a table using a graphical browser

**Description**

To start the browser, the environment variable DISPLAY must be set.

**Arguments**

**Returns**

bool
table.name.html

**table.name - Function**

3.2.1 return name of table on disk

**Description**

Gives the name of the **CASA** table on disk that the table tool has open.

**Arguments**

**Returns**

string

**Example**

```plaintext
tb.open("3C273XC1.MS")
tb.name()
# 3C273XC1.MS
```
table.createmultitable.html

**table.createmultitable - Function**

3.2.1 Create a virtually concatenated table

**Description**

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputTableName</td>
<td>name of the concatenated table</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>tables</td>
<td>list of the names of the tables to be concatenated</td>
<td>stringArray</td>
<td></td>
</tr>
<tr>
<td>subdirname</td>
<td>optional name of the subdirectory into which the input tables are moved</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

---

1666
table.toascifmt - Function

3.2.1 Write CASA table into an ASCII format

Description

Write a table into an ASCII format approximately compatible with fromascii except that in order to permit variable shaped arrays (as they often occur in MSs), array values are output enclosed in square brackets. The separator between values can be specified and defaults to a blank. Note that columns containing invalid data or record type data are ignored and a warning is issued. If the argument headerfile is set then the header information is written to that file instead of the first two lines of the data file.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of ASCII file to be written</th>
</tr>
</thead>
<tbody>
<tr>
<td>asciifile</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>headerfile</td>
<td>Name of an optional file defining the format</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>columns</td>
<td>Names of columns to be written, default is all</td>
</tr>
<tr>
<td></td>
<td>allowed: stringArray</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>sep</td>
<td>Value separator, default is one blank</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

bool

Example

1667
tb.toasciifmt(asciifile='myfile3.dat', headerfile='myfile3.head', columns=['SOURCE_ID', 'NAME', 'PROPER_MOTION'], sep=',

will produce a comma separated ASCII output of the three columns 'SOURCE_ID', 'NAME', and 'PROPER_MOTION' in file 'myfile3.dat' and a format description in 'myfile3.head'.

tb.toasciifmt(asciifile='myfile.dat')

will produce a space separated ASCII output of all table columns into file 'myfile.dat' with the first two lines containing a format description.
table.taql.html

**table.taql - Function**

3.2.1 Make a table from a TaQL command.

**Description**

This method Expose TaQL to the user. Details on TaQL maybe found at http://www.astron.nl/aips++/docs/notes/199

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>TaQL expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>taqlcommand</td>
<td>TaQL expression</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>TaQL expression</td>
</tr>
</tbody>
</table>

**Returns**

table

**Example**

For more information on TaQL see http://www.astron.nl/aips++/docs/notes/199


**Description**

Make a table from a query applied to the current table. It is possible to specify column(s) and/or expressions to sort on and to specify the columns to be contained in the output table. See the example below. A new ”on-the-fly” table tool is returned. The new (reference) table can be given a name and will then be written to disk. Note that the resulting table is just a reference to the original table. One can make a deep copy of the query result using the copy function (see example).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>query</strong></td>
<td>Query string</td>
<td>string</td>
<td>String</td>
</tr>
<tr>
<td><strong>name</strong></td>
<td>Name of resulting reference table</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td><strong>sortlist</strong></td>
<td>Sort string (one or more expressions separated by commas)</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td><strong>columns</strong></td>
<td>List of column names separated by commas</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td><strong>style</strong></td>
<td>How to handle numeric ranges and order axes</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

table

**Example**
tb.open("3C273XC1.MS")
subt=tb.query("OBSERVATION_ID==0",
             sortlist="ARRAY_ID", columns="TIME, DATA, UVW")
print subt.ncols()
# 23
tb.close()
copyt = subt.copy ("3C273XC1_spw1.MS", True)
subt.close()
copyt.close()

From the original table corresponding to the disk file 3C273XC1.MS, only rows with OBSERVATION_ID equal to 0 are selected and sorted by ARRAY_ID. Only the columns TIME DATA UVW are written. Thereafter a deep copy of the result is made. This table query command is equivalent to the Table Query Language (TaQL) command

    SELECT TIME, DATA, UVW  
    FROM 3C273XC1.MS  
    WHERE OBSERVATION_ID==0  
    ORDERBY ARRAY_ID

See http://www.astron.nl/casacore/trunk/casacore/doc/notes/199.html for an explanation of TaQL.

If "style" is not blank, "using style <style> " is prepended to the query. See http://www.astron.nl/casacore/trunk/casacore/doc/notes/199.html#x1-50002.2 for an explanation and list of choices for style. The default (glish) style is 1-based, inclusive end, and Fortran ordering. You may prefer python (0-based, exclusive end, and C order) style.

    tb.open('any_data')
    tsel = tb.selectrows([0])
    print tsel.nrows()  # returns 1
    tsel = tb.query('ROWNUMBER()==0')
    print tsel.nrows()  # returns 0
    tsel = tb.query('ROWNUMBER()==0', style='python')
    print tsel.nrows()  # returns 1
    tb.close()

    Note that style had no effect on the "OBSERVATION_ID==0" query above.
Example

The sortlist argument can be used to sort in ascending or descending order (or a mix of them) on one or more columns. Default is ascending. It is also possible to remove duplicate values using the word NODUPLICATES at the beginning.

E.g.:

- `sortlist='TIME desc'`
- `sortlist='noduplicates ANTEENA1,ANTEENA2'`
- `sortlist='ANTEENA1 desc, ANTEENA2 asc'`
- `sortlist='desc ANTEENA1, ANTEENA2, TIME'`
table.calc.html

**table.calc - Function**

3.2.1 TaQL expression with calc to calculate an expression on a table

**Description**

Get the result from the calculation of an expression on a table. The expression can be any expression that can be given in the WHERE clause of a SELECT expression (thus including subqueries). The given expression determines if the result is a scalar, a vector, or a record containing arrays. See the examples below.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>expr</code></td>
<td>Expression string</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td><code>prefix</code></td>
<td>TaQL prefix for style and ordering etc ...check TaQL</td>
</tr>
<tr>
<td>note 199 for usage</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>using style base0, endincl, fortranorder</td>
</tr>
<tr>
<td><code>showtaql</code></td>
<td>Show the full taql command used</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

anyvariant

**Example**

```plaintext
tb.calc('[select from ngc5921.ms giving [mean(abs(DATA))]]')
```

find the mean of the abs of each row of the DATA column of the MeasurementSet ngc5921.ms
returns a (potentially enormous) record where a field contains the value of the expression for the row with that number. Note that it returns a record because for each row the expression results in an array. It should be clear that this example is useless. However, something like this could be useful for a column with (very) small arrays.

```
tb.calc('*[select from ngc5921.ms.contsub giving [ntrue(FLAG)]]*')
```

returns for each row the number of flags set. The result is a vector, because for each row the expression results in a scalar.

```
tb.calc('sum([select from ngc5921.ms.contsub giving [ntrue(FLAG)]])')
```

returns the total number of flags set in the table (in a single scalar).

using subrow array

```
tb.calc('median([select from ngc5921.ms where ANTENNA1==3 && ANTENNA2==5 giving [abs(DATA[0,31])]])', prefix='using style python')
```

The above will find the median channel 31 and 0th pol of the requested baseline formed with antennas 3 and 5. Note that the default casa order of arrays is fortran order ...pol axis is before channel axis

```
tb.calc('median([select from ngc5921.ms where ANTENNA1==3 && ANTENNA2==5 giving [abs(DATA[31,0])]])', prefix='using style python')
```

Now the same is as the above but using the python style of axis ordering access
table.selectrows.html

**table.selectrows - Function**

3.2.1 Make a table from a selection of rows

**Description**

Create a (reference) table containing a given subset of rows. It is, for instance, useful when a selection is done on another table containing the row numbers in the main table. It can be useful to apply the `casapy` function `unique` to those row numbers, otherwise the same row might be included multiple times (see example).

It is possible to give a name to the resulting table. If given, the resulting table is made persistent with that table name. Otherwise the table is transient and disappears when closed or when `casapy` exits.

The `rownumbers` function returns a vector containing the row number in the main table for each row in the selection table. Thus given a row number vector `rownrs`, the following is always true.

\[
\text{rownrs} \equiv \text{tb.selectrows(rownrs).rownumbers()}
\]

However, it is not true when `selectrows` is used on a selection table. because `rownumbers` does not return the row number in that selection table but in the main table.

It means that one has to take great care when using `selectrows` on a selection table.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>rownrs</strong></td>
<td>0-based Row Numbers</td>
</tr>
<tr>
<td></td>
<td>allowed: intArray</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td><strong>name</strong></td>
<td>Name of resulting table</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

`table`
Example

# EXAMPLE NOT VERIFIED SINCE query IS BROKEN
# Do the query on the main table.
tb.open('SOMENAME')
scantable = tb.query(command)
# Get the column containing the 0-based row numbers in the BACKEND table.
# Make the row numbers unique. NEED TO REPLACE GLISH unique FUNCTION HERE!
backrows = unique(scantable.getcol('NS_GBT_BACKEND_ID'))
# Form the table subset of the BACKEND table containing those rows.
tb.close()
tb.open('SOMENAME/GBT_BACKEND')
scanback = tb.selectrows(backrows);
# Do something with that table.
print scanback.nrows();
table.info.html

**table.info - Function**

3.2.1 get the info record

**Description**

The info record contains information on the table.

**Arguments**

**Returns**

record
table.putinfo.html

**table.putinfo - Function**

3.2.1 set the info record

**Description**

The info record contains information on the table. It is written by applications, and used to determine what type of information is stored in a table.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Value</th>
<th>Info record</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>record</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool
**Description**

A readme line is part of the info record associated with a table. It is to inform the user, and is not used by any application directly.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>readme line</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool
table.summary.html

**table.summary - Function**

3.2.1 summarize the contents of the table

**Description**

A (terse) summary of the table contents is sent to the defaultlogger.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Summarize subtables recursively</th>
</tr>
</thead>
<tbody>
<tr>
<td>recurse</td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: false</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
    tb.open("tcal")
    tb.summary()
    # successful nomodify open of table tcal : 9 columns, 11 rows
    # Table summary: tcal
    # Shape: 9 columns by 11 rows
    # Info: [type=Calibration, subType=T Jones, readme=]
    # Table keywords: [Type=T Jones, Interval=30, DeltaT=1]
    # Columns: StartTime StopTime Gain SolutionOK Fit FitWeight
    # iSolutionOK iFit iFitWeight
```
table.colnames.html

**table.colnames - Function**

3.2.1 return the names of the columns

**Description**

The names of the columns in the table are returned as a vector of Strings.

**Arguments**

**Returns**

stringArray

**Example**

```plaintext
tb.open("tcal")
tb.colnames()
# StartTime StopTime Gain SolutionOK Fit FitWeight iSolutionOK iFit iFitWeight
```
table.rownumbers.html

**table.rownumbers** - Function

3.2.1 !!!INPUT PARAMETERS IGNORED!!! return the row numbers in the (reference) table

**Description**

!!!NOTE INPUT PARAMETERS IGNORED!!!
This function can be useful after a selection or a sort. It returns the row numbers of the rows in this table with respect to the given table. If no table is given, the original table is used.

For example:

!!!NOTE INPUT PARAMETERS IGNORED!!!

```python
tb.open('3C273XC1.MS')
t1=tb.selectrows([1,3,5,7,9])
t1.rownumbers()  # [1L, 3L, 5L, 7L, 9L]
t2=t1.selectrows([2,4])
t2.rownumbers(t1)  # [2L, 4L]
t2.rownumbers(tb.name())  # [5L, 9L]
t2.rownumbers()  # [5L, 9L]
```

The last statements show that the function returns the row numbers referring to the given table. Table t2 contains rows 2 and 4 in table t1, which are rows 5 and 9 in table '3C273XC1.MS'.

Note that when a table is opened using its name, that table can be a reference table. Thus in the example above the last 2 statements may give different results depending on the fact if 3C273XC1.MS is a reference table or not. The function should always be called with a table argument. The ability of omitting the argument is only present for backward compatibility. The function can be useful to get the correct values from the result of a getcol or getcolslice on the original table.

!!!NOTE INPUT PARAMETERS IGNORED!!!

**Arguments**

---

1682
<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>tab</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>nbytes</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Returns**

```python
intArray
```

**Example**

```python
#!NOTE INPUT PARAMETERS IGNORED!!!
tb.open("3C273XC1.MS")
tb.nrows()
#7669L
data = tb.getcolslice("DATA", [0,0], [0,0])
data.shape
#(1, 1, 7669)
selt = tb.query("ANTENNA1==1")
selt.nrows()
#544L
print len(selt.rownumbers())
#544L
```
table.setmaxcachesize.html

**table.setmaxcachesize - Function**

3.2.1 set maximum cache size for column in the table

**Description**

It can sometimes be useful to limit the size of the cache used by a column stored with the tiled storage manager. This function requires some more knowledge about the table system and is not meant for the casual user.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>Name of column</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>nbytes</td>
<td>Maximum cache size in bytes</td>
<td>int</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
tb.open("3C273XC1.MS")
tb.nrows()
# 7669L
tb.setmaxcachesize ("DATA", 4*1024*1024);
# True```

1684
**table.isscalarcol** - Function

3.2.1 is the specified column scalar?

**Description**

A column may contain either scalars or arrays in each cell. This tool function tests if the specified column has scalar contents.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of column</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>allowed: string</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
tb.open("tcal")
tb.isscalarcol("StartTime")
# True
tb.open("tcal")
tb.isscalarcol("Gain")
# False
```
table.isvarcol.html

**Table.isvarcol - Function**

3.2.1 tell if column contains variable shaped arrays

**Description**

This function tells if the column contains variable shaped arrays. If so, the function `getvarcol` should be used to get the entire column. Otherwise `getcol` can be used.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>Name of column</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool
table.coldatatype.html

**table.coldatatype - Function**

3.2.1 return the column data type

**Description**

A column may contain various data types. This tool function returns the type of the column as a string.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of column</th>
<th>allowed: string</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>Name of column</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

string

**Example**

```java
    tb.open("tcal")
    tb.coldatatype("StartTime")
    # double
    tb.open("tcal")
    tb.coldatatype("Gain")
    # complex
```
### Description

The possible column array types are defined as:

**FixedShape**  FixedShape means that the shape of the array must be the same in each cell of the column. If not given, the array shape may vary. Option Direct forces FixedShape.

**Direct**  Direct means that the data is directly stored in the table. Direct forces option FixedShape. If not given, the array is indirect, which implies that the data will be stored in a separate file.

### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>Name of column</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

### Returns

string

### Example

```python
tb.open("tcal")
tb.colarraytype("Gain")
# Direct,FixedShape
```
table.ncols.html

**table.ncols - Function**

3.2.1 return number of columns

**Arguments**

**Returns**

int

**Example**

```python
tb.open("3C273XC1.MS")
tb.ncols()
tb.ncols()
# 23L
```
table.nrows.html

**table.nrows - Function**

3.2.1 return number of rows

**Description**

Note that rows are numbered starting at 0.

**Arguments**

**Returns**

int

**Example**

```
tb.open("3C273XC1.MS")
tb.nrows()
# 7669L
```
**table.addrows** - Function

3.2.1 add a specified number of rows

**Description**

Rows can be added to the end of a table that was opened `nomodify=False`. The new rows are empty.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nrow</td>
<td>Number of rows to add</td>
</tr>
</tbody>
</table>

allowed: int

Default: 1

**Returns**

bool
table.removerows.html

**table.removerows - Function**

3.2.1 remove the specified rows

**Description**

Remove the row numbers specified in the vector from the table. It fails when the table does not support row removal.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>rownrs</td>
<td>Row numbers to remove</td>
<td>intArray</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool
### table.addcols - Function

3.2.1 !!!REQUIRES COLUMN DESCRIPTION FUNCTIONS THAT HAVE NOT BEEN IMPLEMENTED!!! add one or more columns

**Description**

Columns can be added to a table that was opened nomodify=False. The new columns will be filled with a default value (0 or blank).

!!!THESE COLUMN DESCRIPTION FUNCTIONS HAVE NOT BEEN IMPLEMENTED!!!

For each column to be added a column description has to be setup using function tablecreatescalarcoldesc or tablecreatearraycoldesc. When multiple columns are used, they have to be combined in a single record using tablecreatedesc.

It is possible to specify data manager info in order to define a data manager (storage manager or virtual column engine) for the columns to be added.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>desc</td>
<td>Description of one or more columns</td>
</tr>
<tr>
<td></td>
<td>allowed: record</td>
</tr>
<tr>
<td>dminfo</td>
<td>Optional description data manager to use</td>
</tr>
<tr>
<td></td>
<td>allowed: record</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

!!!REQUIRES COLUMN DESCRIPTION FUNCTIONS THAT HAVE NOT BEEN IMPLEMENTED!!!

```plaintext
tb.open("mytable", nomodify=False)
```
dc3 = tablecreate scalar coldesc('C3', 'a')
dc4 = tablecreate scalar coldesc('C4', as_float(0))
dc5 = tablecreate array coldesc('C5', as_double(0), 2, [10,20])
tb.addcols(dc3)
# True
tb.addcols(tablecreatedesc(dc4, dc5))
# True

A single column can be added as such, but multiple columns have to be combined.
Description

A column can be renamed in a table that was opened nomodify=False. However, renaming is not possible in a (reference) table resulting from a select or sort operation.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>name of column to be renamed</th>
</tr>
</thead>
<tbody>
<tr>
<td>oldname</td>
<td>name of column to be renamed</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>newname</td>
<td>new name of column</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool

Example

```python
tb.open("3C273XC1.MS", nomodify=False)
tb.renamecol ('DATA', 'DATA2')
# T
print tb.colnames()
tb.renamecol ('DATA2', 'DATA')
# T
print tb.colnames()
```

Column \texttt{DATA} is renamed to \texttt{DATA2} and then back to \texttt{DATA} again.
table.removecols.html

**table.removecols - Function**

3.2.1 remove one or more columns

### Description

Columns can be removed from a table that was opened nomodify=False. It may not always be possible to remove a column, because some data managers do not support column removal. However, if all columns of a data manager are removed, it will always succeed. It results in the removal of the entire data manager (and its possible files).

Note that function getdminfo can be used to find which columns are served by which data manager.

### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>names of columns to be removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnnames</td>
<td>allowed: stringArray</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

### Returns

bool

### Example

```python
    tb.open("mytable", nomodify=False)
    tb.removecols ("col1 col2")
    print tb.colnames()
    # T
    print tb.colnames()

Two columns are removed.
```
**Description**

A column containing variable shaped arrays can have an empty cell (if no array has been put into it). This function tests if a cell is defined (thus is not empty). Note that a scalar column and a fixed shape array column cannot have empty cells.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of column</th>
<th>allowed: string</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>Name of column</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rownr</td>
<td>Row number, starting at 0</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

**Returns**

bool
3.2.1 get a specific cell

Description

A cell is the value at one row in one column. It may be a scalar or an array.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of column</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>Name of column</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>rownr</td>
<td>Row number, starting at 0</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
</tbody>
</table>

Returns

anyvariant
3.2.1 get a slice from a specific cell

**Description**

A cell is the value at one row in one column. It must be an array. The slice must be specified as blc, trc with an optional stride. In blc and trc -1 can be used to indicate all values for a dimension (-1 in blc is equivalent to 0, so -1 is especially useful for trc).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of column</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>Name of column</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>rownr</td>
<td>Row number, starting at 0</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>blc</td>
<td>Bottom left corner (e.g. [0,0,0] is start of 3D array)</td>
<td>intArray</td>
<td></td>
</tr>
<tr>
<td>trc</td>
<td>Top right corner</td>
<td>intArray</td>
<td></td>
</tr>
<tr>
<td>incr</td>
<td>Stride (defaults to 1 for all axes)</td>
<td>intArray</td>
<td>1</td>
</tr>
</tbody>
</table>

**Returns**

anyvariant

**Example**

1700
tb.open("3C273XC1.MS")
data=tb.getcellslice("DATA", 0, [0,0], [1,0])
print data.shape
# [2 1]
table.getcol.html

**table.getcol - Function**

3.2.1 get a specific column

**Description**

The entire column (or part of it) is returned. Warning: it might be big! The functions can only be used if all arrays in the column have the same shape. That is guaranteed for columns containing scalars or fixed shaped arrays. For columns containing variable shaped arrays it only succeeds if all those arrays happen to have the same shape.

Note that function `getvarcol` can be used to get a column of arbitrary shaped arrays, which also handles empty cells correctly. Function `isvarcol` tells if a column contains variable shaped arrays. shaped

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>Name of column</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>startrow</td>
<td>First row to read (default 0)</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>nrow</td>
<td>Number of rows to read (default -1 means till the end)</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>rowincr</td>
<td>Increment in rows to read (default 1)</td>
<td>int</td>
<td>1</td>
</tr>
</tbody>
</table>

**Returns**

anyvariant

**Example**
tb.open("3C273XC1.MS")
# True
gain=tb.getcol("DATA")
print gain.shape
# (4, 1, 7669)
Function `getcol` can only use if values in the column cells to get have the same shape. Function `getvarcol` addresses this limitation by returning the values as a record instead of an array. Each field in the record contains the value for a column cell. If the value is undefined (i.e. the cell does not contain a value), the unset value is put in the record. Each field name is the letter r followed by the row number. The length of the record is the number of rows to get.

Note that the function `isvarcol` tells if a column contains variable shaped arrays.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of column</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>Name of column</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>startrow</td>
<td>First row to read (default 0)</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
<tr>
<td>nrow</td>
<td>Number of rows to read (default -1 means till the end)</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td>rowincr</td>
<td>Increment in rows to read (default 1)</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 1</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

1704
tb.open("3C273XC1.MS")
gain=tb.getvarcol("DATA")
print len(gain)
# 7669
table.getcolslice.html

**table.getcolslice - Function**

3.2.1 get a slice from a specific column array

**Description**

A slice from the entire column (or part of it) is returned. Warning: it might be big!
In blc and trc -1 can be used to indicate all values for a dimension (-1 in blc is equivalent to 1, so -1 is especially useful for trc). Note that blc and trc should not contain the row number, only the blc and trc of the arrays in the column.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of column</th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>Name of column</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>blc</td>
<td>Bottom left corner (e.g. [0,0,0] is start of 3D array)</td>
<td>intArray</td>
<td></td>
</tr>
<tr>
<td>trc</td>
<td>Top right corner</td>
<td>intArray</td>
<td></td>
</tr>
<tr>
<td>incr</td>
<td>Stride (defaults to 1 for all axes)</td>
<td>intArray</td>
<td></td>
</tr>
<tr>
<td>startrow</td>
<td>First row to read (default 0)</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>nrow</td>
<td>Number of rows to read (default -1 means till the end)</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>rowincr</td>
<td>Increment in rows to read (default 1)</td>
<td>int</td>
<td>1</td>
</tr>
</tbody>
</table>

**Returns**

anyvariant

1706
Example

tb.open("3C273XC1.MS")
data=tb.getcolslice("DATA", [0,0], [1,0])
data.shape
# (2 1 7669)
### table.putcell - Function

#### 3.2.1 put a specific cell

**Description**

A cell is the the value at one row in one column. It may be a scalar or an array.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>Name of column</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>rownr</td>
<td>Row number(s) (0-relative)</td>
<td>intArray</td>
<td></td>
</tr>
<tr>
<td>thevalue</td>
<td>Value</td>
<td>any</td>
<td>variant</td>
</tr>
</tbody>
</table>

**Returns**

bool
**table.putcellslicel - Function**

**3.2.1 put a slice into a specific cell**

**Description**

A cell is the value at one row in one column. It must be an array. The slice must be specified as blc, trc with an optional stride. In blc and trc -1 can be used to indicate all values for a dimension (-1 in blc is equivalent to 0, so -1 is especially useful for trc).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of column</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>Name of column</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>rownr</td>
<td>Row number, starting at 0</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>Value</td>
</tr>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
<tr>
<td>blc</td>
<td>Bottom left corner (e.g. [0,0,0] is start of 3D array)</td>
</tr>
<tr>
<td>allowed:</td>
<td>intArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>trc</td>
<td>Top right corner</td>
</tr>
<tr>
<td>allowed:</td>
<td>intArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>incr</td>
<td>Stride (defaults to 1 for all axes)</td>
</tr>
<tr>
<td>allowed:</td>
<td>intArray</td>
</tr>
<tr>
<td>Default:</td>
<td>1</td>
</tr>
</tbody>
</table>

**Returns**

bool
table.putcol.html

**table.putcol - Function**

3.2.1 put a specific column

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>Name of column</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>Array</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>startrow</td>
<td>First row to put (default 0)</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>nrow</td>
<td>Number of rows to put (default -1 means till the end)</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>rowincr</td>
<td>Increment in rows to put (default 1)</td>
<td>int</td>
<td>1</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
import tb

# open table
tb.open("3C273XC1.MS", nomodify=False)
data = tb.getcol("DATA")
# [could modify data here]
tb.putcol("DATA", data)
tb.flush()
```
3.2.1 put a specific column (for variable arrays)

Description

`putcol` can only used if values in the column cells to put have the same shape. `putvarcol` addresses this limitation by passing the values as a record instead of an array. Each field in the record contains the value for a column cell. So the length of the record has to match the number of rows to put. If a value is the unset value, no put is done for that row.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of column</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>Name of column</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>Record with values</td>
</tr>
<tr>
<td>allowed:</td>
<td>record</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>startrow</td>
<td>First row to put (default 0)</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td>nrow</td>
<td>Number of rows to put (default -1 means till the end)</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>-1</td>
</tr>
<tr>
<td>rowincr</td>
<td>Increment in rows to put (default 1)</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>1</td>
</tr>
</tbody>
</table>

Returns

bool

Example

1711
tb.open("3C273XC1.MS",nomodify=False)
gain=tb.getvarcol("DATA", 0, 10)
tb.putvarcol("Gain", gain, 10, 10)
tb.flush()

This example copies the values from row 0-9 to row 10-19.
3.2.1 put a slice into a specific column

Description

In blc and trc, -1 can be used to indicate all values for a dimension (-1 in blc is equivalent to 0, so -1 is especially useful for trc). Note that blc and trc should not contain the row number, only the blc and trc of the arrays in the column.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of column</th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>Name of column</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>Array</td>
<td>any</td>
<td>variant</td>
</tr>
<tr>
<td>blc</td>
<td>Bottom left corner (e.g. [0,0,0] is start of 3D array)</td>
<td>intArray</td>
<td></td>
</tr>
<tr>
<td>trc</td>
<td>Top right corner</td>
<td>intArray</td>
<td></td>
</tr>
<tr>
<td>incr</td>
<td>Stride (defaults to 1 for all axes)</td>
<td>intArray</td>
<td>1</td>
</tr>
<tr>
<td>startrow</td>
<td>First row to put (default 0)</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>nrow</td>
<td>Number of rows to put (default -1 means till the end)</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>rowincr</td>
<td>Increment in rows to put (default 1)</td>
<td>int</td>
<td>1</td>
</tr>
</tbody>
</table>

Returns

1713
Example

tb.open("3C273X1.MS", nomodify=False)
data_all = tb.getcolslice("DATA", [-1,-1], [-1,-1])
print data_all.shape
# (4, 1, 7669)
data = tb.getcolslice("DATA", [0,0],[3,0])
# can modify data here
tb.putcolslice("DATA", data, [0,0],[3,0])
tb.flush()
### table.getcolshapestring - Function

3.2.1 get shape of arrays in a specific column

#### Description

The shapes of the arrays in the entire column (or part of it) are returned as strings like [20,3]. When the column contains fixed shaped arrays, a single string is returned. Otherwise a vector of strings is returned.

#### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>Name of column</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>startrow</td>
<td>First row to read (default 0)</td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: 0</td>
</tr>
<tr>
<td>nrow</td>
<td>Number of rows to read (default -1 means till the end)</td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td>rowincr</td>
<td>Increment in rows to read (default 1)</td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: 1</td>
</tr>
</tbody>
</table>

#### Returns

stringArray

#### Example

```python
tb.open("3C273XC1.MS")
shapes=tb.getcolshapestring("DATA")
print len(shapes)
```
**table.getkeyword.html**

**table.getkeyword - Function**

3.2.1 get value of specific table keyword

**Description**

The value of the given table keyword is returned. The value can be of any type, including a record and a table. If a keyword is a table, its value is returned as a string containing the table name prefixed by 'Table: '. It is possible that the value of a keyword is a record itself (arbitrarily deeply nested). A field in such a subrecord can be read by separating the name with dots.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name or seqnr of keyword: string or int</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyword</td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
</tbody>
</table>

**Returns**

anyvariant

**Example**

```plaintext
tb.open('3C273XC1.MS')
tb.getkeywords()
tb.getkeyword('MS_VERSION')
# 2.0
    tb.close()
tb.open('tcal')
tb.getkeyword('rec.fld')  # get field from a record
# 3.14
```

1717
table.getkeywords - Function

3.2.1 get values of all table keywords

Description

The values of all table keywords are returned. The values can be of any type, including a record and a table. If a keyword is a table, its value is returned as a string containing the table name prefixed by 'Table: '.

Arguments

Returns

record

Example

```python
tb.open('3C273XC1.MS')
tb.getkeywords()
#{
    'ANTENNA': 'Table: /home/aips2mgr/testing/3C273XC1.MS/ANTENNA',
    'DATA_DESCRIPTION': 'Table: /home/aips2mgr/testing/3C273XC1.MS/DATA_DESCRIPTION',
    'FEED': 'Table: /home/aips2mgr/testing/3C273XC1.MS/FEED',
    'FIELD': 'Table: /home/aips2mgr/testing/3C273XC1.MS/FIELD',
    'FLAG_CMD': 'Table: /home/aips2mgr/testing/3C273XC1.MS/FLAG_CMD',
    'HISTORY': 'Table: /home/aips2mgr/testing/3C273XC1.MS/HISTORY',
    'MS_VERSION': 2.0,
    'OBSERVATION': 'Table: /home/aips2mgr/testing/3C273XC1.MS/OBSERVATION',
    'POINTING': 'Table: /home/aips2mgr/testing/3C273XC1.MS/POINTING',
    'POLARIZATION': 'Table: /home/aips2mgr/testing/3C273XC1.MS/POLARIZATION',
    'PROCESSOR': 'Table: /home/aips2mgr/testing/3C273XC1.MS/PROCESSOR',
    'SOURCE': 'Table: /home/aips2mgr/testing/3C273XC1.MS/SOURCE',
}
```
# 'SPECTRAL_WINDOW': 'Table: /home/aips2mgr/testing/3C273XC1.MS/SPECTRAL_WINDOW',
# 'STATE': 'Table: /home/aips2mgr/testing/3C273XC1.MS/STATE'}
Description

The value of the given column keyword is returned. The value can be of any type, including a record and a table. If a keyword is a table, its value is returned as a string containing the table name prefixed by 'Table:'. It is possible that the value of a keyword is a record itself (arbitrarily deeply nested). A field in such a subrecord can be read by separating the name with dots.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>Name of column</td>
<td>allowed: string</td>
<td>Default:</td>
</tr>
<tr>
<td>keyword</td>
<td>Name or seqnr of keyword: string or int</td>
<td>allowed: any</td>
<td>Default: variant</td>
</tr>
</tbody>
</table>

Returns

anyvariant

Example

```
tb.open("3C273XC1.MS")
tb.getcolkeyword("UVW", "QuantumUnits")
#array(["m", "m", "m"],
#      dtype='|S2')
```

1721
table.getcolkeywords.html

**table.getcolkeywords - Function**

*3.2.1* get values of all keywords for a column

**Description**

The values of all keywords for the given column are returned. The values can be of any type, including a record and a table. If a keyword is a table, its value is returned as a string containing the table name prefixed by 'Table: '.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of column</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>Name of column</td>
</tr>
</tbody>
</table>

allowed: string

Default: 

**Returns**

anyvariant

**Example**

```python
# tb.open("3C273XC1.MS")
b.getcolkeywords("UVW")
#{'MEASINFO': {'Ref': 'ITRF', 'type': 'uvw'},
 # 'QuantumUnits': array([m', 'm', 'm'],
 # dtype='|S2'})
```
**table.putkeyword.html**

**table.putkeyword - Function**

3.2.1 put a specific table keyword

**Description**

Put a table keyword. The value of the keyword can be a scalar or an array of any type or it can be a record. It is possible to define a keyword holding a subtable. In that case a special string containing the name of the subtable will be passed to the table client. It is possible that the value of a keyword is a record itself (arbitrarily deeply nested). A field in such a subrecord can be written by separating the name with dots. If a subrecord does not exist, an error is returned unless `makesubrecord=True` is given. In such a case intermediate records are created when needed.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name or seqnr of keyword: string or int</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyword</td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
<tr>
<td>value</td>
<td>Value of keyword</td>
</tr>
<tr>
<td></td>
<td>allowed: any</td>
</tr>
<tr>
<td></td>
<td>Default: variant</td>
</tr>
<tr>
<td>makesubrecord</td>
<td>Create intermediate records</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: false</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
 tb.open("3C273XC1.MS", nomodify=False)
```
tb.putkeyword("VERSION", "1.66")
# True
# define ANTENNA subtable
  tb.putkeyword("ANTENNA", 'Table: 3C273XC1.MS/ANTENNA')
  tb.flush()
# True
# write a field in a record and create subrecords when needed
  tb.putkeyword("REC.SUB.FLD", "val", True)
# True
# write a keyword with a record value
  tb.putkeyword("REC", {'SUB': {'FLD': 'val'}})
# True

Note that the last example does the same as the previous one (assuming
that \
texttt{REC} does not exist yet with other fields).
table.putkeywords.html

**table.putkeywords - Function**

3.2.1 !!!BROKEN!!! put multiple table keywords

**Description**

Put multiple table keywords. All fields in the given record are put as table keywords. The value of each field can be a scalar or an array of any type or it can be a record. It is also possible to define a keyword holding a subtable. This can be done by giving the keyword a string value consisting of the subtable name prefixed by 'Table: '.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Record of keyword=value pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>allowed: record</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
 tb.open('3C273XC1.MS', nomodify=False)
 kw=tb.getkeywords()
 print kw['MS_VERSION']
 # 2.0
 kw['MS_VERSION']=2.1
 tb.putkeywords(kw)
 # !!!BROKEN. Keywords containing float are not handled properly!!!
 tb.flush()
 # True
```

1725
table.putcolkeyword.html

**table.putcolkeyword - Function**

3.2.1 put a specific keyword for a column

**Description**

Put a keyword in the given column. The value of the keyword can be a scalar or an array of any type or it can be a record.

It is possible to define a keyword holding a subtable. In that case a special string containing the name of the subtable will be passed to the table client.

It is possible that the value of a keyword is a record itself (arbitrarily deeply nested). A field in such a subrecord can be written by separating the name with dots. If a subrecord does not exist, an error is returned unless `makesubrecord=True` is given. In such a case intermediate records are created when needed.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of column</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>Name of column</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>keyword</th>
<th>Name or sequ of keyword,string or int</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>value</th>
<th>Value of keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>any</td>
</tr>
<tr>
<td>Default:</td>
<td>variant</td>
</tr>
</tbody>
</table>

**Returns**

`bool`

**Example**

```
tb.open("3C273XC1.MS", nomodify=False)
```

1727
ckw=tb.getcolkeyword("UVW","QuantumUnits")
print ckw
# modify ckw as desired
tb.putcolkeyword("UVW","QuantumUnits",ckw)
# True
tb.flush()
# True
3.2.1 put multiple keywords for a column

Description

Put multiple keywords in the given column. All fields in the given record are put as column keywords. The value of each field can be a scalar or an array of any type or it can be a record.
It is also possible to define a keyword holding a subtable. This can be done by giving the keyword a string value consisting of the subtable name prefixed by 'Table: '.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
</tr>
<tr>
<td>Name of column</td>
</tr>
<tr>
<td>allowed: string</td>
</tr>
<tr>
<td>Default:</td>
</tr>
<tr>
<td>value</td>
</tr>
<tr>
<td>Record of keyword=value pairs</td>
</tr>
<tr>
<td>allowed: record</td>
</tr>
<tr>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

bool

Example

```
b.open("3C273XC1.MS", nomodify=False)
kws = tb.getcolkeywords("UVW")
kws
#{'MEASINFO': {'Ref': 'ITRF', 'type': 'uvw'},
# 'QuantumUnits': array([['m', 'm', 'm'],
# dtype='|S2'])
# kws['MEASINFO']['Ref']="B1950"
```

1729
tb.putcolkeywords(kws)
# True
table.removekeyword - Function

3.2.1 remove a specific table keyword

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyword</td>
<td>Name or seqnr of keyword: string or int</td>
<td>any</td>
<td>variant</td>
</tr>
</tbody>
</table>

Returns

bool

Example

```python
  tb.open("3C273XC1.MS", nomodify=False)
  tb.removekeyword("MS_VERSION")
  # True
  tb.flush()
  # True
```
table.removecolkeyword.html

**table.removecolkeyword - Function**

3.2.1 remove a specific keyword for a column

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>Name of column</td>
<td>string</td>
<td>string</td>
</tr>
<tr>
<td>keyword</td>
<td>Name or seqnr of keyword: string or int</td>
<td>any</td>
<td>variant</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
tb.open("3C273XC1.MS", nomodify=False)
tb.removecolkeyword("UVW", "QuantumUnits")
# True
tb.flush()
# True```

1732
**table.getdminfo - Function**

3.2.1 get the info about data managers

**Description**

This function returns the types and names of the data managers used. For each data manager it also returns the names of the columns served by it. The information is returned as a record containing a subrecord for each data manager. Each subrecord contains the fields TYPE, NAME and COLUMNS.

**Arguments**

**Returns**

record

**Example**

```python
tb.open('3C273XC1.MS')
rec = tb.getdminfo()
```

Print the output record shows that the table uses 9 storage managers.
**table.keywordnames - Function**

3.2.1 get the names of all table keywords

**Description**

This function returns a vector of strings containing the names of all table keywords.

**Arguments**

**Returns**

stringArray
**table.fieldnames - Function**

3.2.1 get the names of fields in a table keyword

**Description**

This function returns a vector of strings containing the names of all fields in
the given table keyword. It is only valid if the keyword value is a record.
If no keyword name is given, the names of all table keywords are returned.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>keyword name</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyword</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

stringArray
**table.colkeywordnames - Function**

3.2.1 get the names of all keywords in a column

**Description**

This function returns a vector of strings containing the names of all keywords in the column with the given name.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>column name</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>column name</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

stringArray

**Example**

```plaintext
tb.open('3C273XC1.MS')
tb.colkeywordnames("UVW")
```
**table.colfieldnames - Function**

3.2.1 get the names of fields in a keyword in a column

**Description**

This function returns a vector of strings containing the names of all fields in the given keyword in the given column. It is only valid if the keyword value is a record. If no keyword name is given, the names of all keywords in the column are returned.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>column name</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>column name</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>string</td>
</tr>
<tr>
<td>keyword</td>
<td>keyword name</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>string</td>
</tr>
</tbody>
</table>

**Returns**

stringArray
**table.getdesc - Function**

3.2.1 get the table description

**Description**

The table description is a casapy record that contains a complete description of the layout of the table (except for the number of rows). By default the actual table description is returned (thus telling the actual shapes and data managers used). It is also possible to get the table description used when creating the table.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>actual table description?</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>true</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
tb.open("3C273XC1.MS")
tb.getdesc()
```
3.2.1. Get the description of a specific column

Description

The column description is a casapy record that contains a complete description of the layout of a specified column (except for the number of rows). It can be used to construct a table description.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of column</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnname</td>
<td>Name of column</td>
</tr>
<tr>
<td>allowed: string</td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

record

Example

```python
tb.open("3C273XC1.MS")
tb.getcoldesc("DATA")
#{'comment': 'The data column',
# 'dataManagerGroup': 'TiledData',
# 'dataManagerType': 'TiledShapeStMan',
# 'maxlen': 0,
# 'ndim': 2,
# 'option': 0,
# 'valueType': 'complex'}
```
table.ok.html

**table.ok - Function**

3.2.1 Is the table tool ok?

**Description**

Perform a number of sanity checks and return T if ok. Failure (returning F) is a sign of a bug.

**Arguments**

**Returns**

bool
table.clearlocks - Function

3.2.1 Clears any table lock associated with the current process

Description

Occasionally a table will be inretrievably locked to another process no matter how much closing is done. So clearLocks will unlock all the files in the table cache that use AutoLocking.

Arguments

Returns

bool
table.listlocks.html

**table.listlocks - Function**

Lists any table lock associated with the current process

**Description**

Occasionally a table will be inretrievably locked to another process no matter how much closing is done. So listLocks will list the offending tables (and unoffending ones, too), so we can figure out where the problem might be.

**Arguments**

**Returns**

bool
table.statistics.html

**table.statistics - Function**

3.2.1 Get statistics on the selected table column

**Description**

This function computes descriptive statistics on the table column. It returns the statistical values as a dictionary. The given column name must be a numerical column. If it is a complex valued column, the parameter complex_value defines which derived real value is used for the statistics computation.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Column name</th>
<th>allowed: string</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>column</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>complex_value</td>
<td>Which derived value to use for complex columns (amp, amplitude, phase, imag, real, imaginary)</td>
<td>allowed: string</td>
<td>Default:</td>
</tr>
<tr>
<td>useflags</td>
<td>Use the data flags</td>
<td>allowed: bool</td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

record

**Example**

```python
    tb.open("ggtau.1mm.amp.gcal")
    s = tb.statistics(column="GAIN", complex_value="phase")
```

1743
table.showcache.html

**table.showcache - Function**

3.2.1 show the contents of the table cache

**Description**

Show the contents of the table cache.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>verbose</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: true</td>
</tr>
</tbody>
</table>

**Returns**

stringArray

**Example**

```plaintext
tb.showcache()
```

1745
table.testincrstman - Function

3.2.1 Checks consistency of an Incremental Store Manager bucket layout

Description

Checks consistency of an Incremental Store Manager bucket layout
In case of corruption it returns False and a SEVERE msg is posted containing information about the location of the corrupted bucket

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Column name</th>
</tr>
</thead>
<tbody>
<tr>
<td>column</td>
<td>string</td>
</tr>
</tbody>
</table>

Returns

bool

Example

```python
mytb = tbtool()
mytb.open('uid___A002_X841035_X203.ms.split')
mytb.testincrstman('FLAG_ROW')
```
3.2.2 tableplot - Tool

Plot data from tables via plotter

Requires:

Synopsis

Description

tableplot is a plotting tool for general CASA tables. Table columns can be plotted against each other, and can be combined using TaQL to create expressions for derived quantities, the result of which can then be plotted. Data from more than one table can be accessed and plotted at the same time. Expressions producing arrays result in overlay plots. The default plot style is single-panel, but if an iteration axis is specified, multi-panel plots are supported. Zooming and region-based flagging is possible on single and multi panel plots. A GUI, adapted to a particular kind of table (measurement set) can call the tableplot functions and customize the generated plots.

Methods

- open: Specify list of tables to operate on.
- setgui: Set the GUI on or off. Can be done only once !!
- savefig: Save the currently plotted image.
- selectdata: Perform a TaQL based subtable selection for subsequent plots.
- plotdata: Plot the result of a general TaQL expression.
- iterplotstart: Initialize plotting with an iteration axis
- replot: Replot all existing panels and layers.
- iterplotnext: Start/Continue plotting
- iterplotstop: Stop plot iterations.
- markregions: Mark a rectangular region to flag
- flagdata: Flag Data for selected flag regions
- unflagdata: Unset flags for all regions marked using tp.markregions().
- locatedata: Print info about data selected using tp.markregions().
- clearflags: Clear all flags in the table.
- saveflagversion: Save current flags with a version name.
- restoreflagversion: Restore flags from a saved flag_version.
- deleteflagversion: Delete a saved flag_version.
- getflagversionlist: Print out a list of saved flag_versions.
- clearplot: Clear a plot.
- done: Clean up the tableplot tool
tableplot.open.html

**tableplot.open - Function**

3.2.2 Specify list of tables to operate on.

**Description**

Specify a list of table names to open for plotting. All plots will operate on these tables until open is called again to change the list of tables. Returns true if tables are valid, false otherwise.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>List of strings identifying Table names</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabnames</td>
<td>allowed: stringArray</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```
# set the list of tables to plot from.
tp.open(tabnames=['3c273.ms','3c48.ms'])
```
Set the GUI on or off. Can be done only once!!

**Description**
Set the GUI on or off. Can be done only once!!

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>gui</th>
<th>gui=False to turn off gui</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: false</td>
</tr>
</tbody>
</table>

**Returns**
bool
tableplot.savefig.html

**tableplot.savefig - Function**

3.2.2 Save the currently plotted image.

**Description**

Store the contents of the plot window in a file. The file format (type) is based on the file name, i.e. the file extension given determines the format the file is saved as. The accepted formats are eps, ps, png, pdf, and svg. Internally, this function uses the matplotlib pl.savefig function. Note that if a full path is not given that the files will be saved in the current working directory.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
</table>
| filename | Name the plot image is to be saved to.  
allowed: string  
Default: |
| dpi | Number of dots per inch (resolution) to save the image at.  
allowed: int  
Default: -1 |
| orientation | Either landscape or portrait. Supported by the postscript format only.  
allowed: string  
Default: |
| papertype | Valid values are: letter, legal, exective, ledger, a0-a10 and b0-b10. This option is supported by the postscript format only.  
allowed: string  
Default: |
| facecolor | Color of space between the plot and the edge of the square. Valid values are the same as those accepted by the plotcolour option of tp.plotdata().  
allowed: string  
Default: |
| edgecolor | Color of the outer edge. Valid values are the same as those accepted by the plotcolour option of tp.plotdata().  
allowed: string  
Default: |
Returns
bool
tableplot.selectdata.html

**tableplot.selectdata - Function**

3.2.2 Perform a TaQL based subtable selection for subsequent plots.

**Description**

Specify a TaQL select string. A subtable will be generated, and passed to the plotter.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>TaQL string for selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>taqlstring</td>
<td>string</td>
</tr>
<tr>
<td>allowed:</td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
# set the list of tables to plot from.
tp.open(tabnames=['3c273.ms','3c48.ms'])
tp.selectdata(taqlstring='ANTENNA1==5')
```
tableplot.plotdata.html

**tableplot.plotdata - Function**

3.2.2 Plot the result of a general TaQL expression.

**Description**

This function evaluates the specified TaQL expressions for the X and Y axes of a two-dimensional plot, extracts the resulting columns, and plots them.

1. TaQL expressions resulting in scalars are plotted as a single X-Y plot. Expressions involving Array-Columns can result in arrays which are then plotted as overlays.

2. The default mode of operation is to plot one expression against another. In the case of Array-Columns (ex. DATA column of a measurement set with each row containing an array of shape NColumns x NPolarizations), a Cross-plot mode allows plots with the x-axis representing one axes of the Array-Column. (ex. X-axis is channel number or polarization number).

3. If multiple tables are specified in `tp.open()`, then the TaQL expressions are applied to all tables and overlay plots are generated.

4. Plots can be made to separate panels whose locations on the plot window are user specified as a 3-tuple `[nrows, ncolumns, panelnumber]`.

5. Multiple plots can be stacked upon each other on a panel (overplot mode).

6. Plotter Options can be specified to control appearance, plot-style, labels, etc.

Valid TaQL strings must satisfy the following conditions.

1. Each TaQL string must result in a Double scalar or array.
   - `'AMPLITUDE(DATA[1,1])'` results in a Double scalar (valid).
   - `'AMPLITUDE(DATA[1:2,1])'` results in a Double array (valid).
   - `'MEAN(AMPLITUDE(DATA[1:2,1]))'` results in a Double scalar (valid).
   - `'DATA[1,1]'` results in a Complex scalar (NOT valid).
   - `'AMPLITUDE(DATA[1,1])<10'` results in a Bool scalar (NOT valid).

2. All TaQL functions resulting in Double Scalars/Arrays are allowed, except for those involving an explicit collapse axis (means,sums,etc..). Note that these functions are different from mean,sum,etc.. which are supported.
3. TaQL strings must be provided as pairs of strings, with the X-TaQL first, followed by the Y-TaQL. There are 3 cases.
   X-TaQL − Scalar, Y-TaQL − Scalar (one-to-one single plot)
   X-TaQL − Scalar, Y-TaQL − Array (one-to-many overlay plot)
   X-TaQL − Array, Y-TaQL − Array (if the shapes are the same, then a one-to-one mapping is done, otherwise only the first X-TaQL result is used for a one-to-many mapping with the Y-TaQL Array.)

4. For cross plots (for example amplitude vs channel plots in an MS), the X-TaQL must be a string containing 'CROSS'. The Y-TaQL is used to read out the data from the table, and the x-values are the column indices (channel numbers) chosen by the Y-TaQL.

Plotting options ('poption' entries) are listed below. Default values are indicated within [ ] when present.

- nrows [ 1 ] : Number of rows of panels
- ncols [ 1 ] : Number of columns of panels
- panel [ 1 ] : Panel index. Must be in \([1,\text{nrows} \times \text{ncols}]\)
- plotcolour [ 1 ] : Plot colour. Codes for matplotlib are \([0:black, 1:red, 2:green, 3:blue, 4:cyan, 5:yellow]\). \([\text{magenta is reserved for plotting flagged values if showflags = 1 , and cannot be set by the user as a plot colour}]\)
- If the plotcolour field is left out from 'poption', colours are chosen automatically.
- multicolour [ False ] : True - Each channel, pol appears in a different colour. False - Data from all pols and channels appear in the same colour. Different colours appear for different layers (overplot). or data from different tables.
- timeplot [ False ] : True - Turn on date/time formatting for the x-axis.
- overplot [ False ] : True - Overlay on an existing plot. All layers will remain active for data editing via flagging. Labels will be those of the top-most layer. False - Replace an existing plot with a new one. In the case of an existing stack of plots, the top-most layer is replaced. For example, this can be used to modify the colour of the top-most layer without creating an additional layer.
- py_plotsymbol [ , ] : Plot markers. Options for matplotlib are \([,]:\text{pixel}, [,]:\text{point}, [o]:\text{circle}, [x]:\text{cross}, [+]:\text{plus}, [t]:\text{triangle up}, [v]:\text{triangle down}, [j]:\text{triangle left}, [z]:\text{triangle right}, [-]:\text{solid line}, [ ]:\text{dashed line}, [.-]:\text{dash-dot line}, [:]:\text{dotted line}, [s]:\text{square}, [d]:\text{diamond}, [d]:\text{thin diamond}, [1]:\text{tripod down}, [2]:\text{tripod up}, [3]:\text{tripod left}, [4]:\text{tripod right}, [h]:\text{hexagon}, [H]:\text{rotated hexagon}, [p]:\text{pentagon}, [—]:\text{vertical line symbol}, [\|]:\text{horizontal line symbol}].
- markersize [ 10.0 ] : The size (in pixels) of the markers being plotted.
- showflags [ 0 ] : True - Plot only unflagged data. False - Plot only flagged data.
- crossdirection [ False ] : Applies only with CROSS-plots on table ArrayColumns. False - use column number as the x-axis (ex. channel no.). True - use row number as the x-axis (ex. polarization no.).
- pointlabels [ ] : Data points can be annotated by supplying a list of labels. If N labels are supplied, the first N data points plotted will be
annotated. (Note that if data is edited via flagging, the points are relabeled to label the first N points.)

`windowsize [ 8.0 ]`: horizontal size of plot window (inches)

`aspectratio [ 0.8 ]`: aspect-ratio of the plot window (dx/dy)

`fontsize [ 12.0 ]`: Font size of title text. Font size of x,y labels are 80% of this.

Returns true if plotting is successful, false otherwise.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>poption</td>
<td>Record of plot options</td>
<td>record</td>
<td>1 1 6 0.8 1.0</td>
</tr>
<tr>
<td>labels</td>
<td>List of strings : Title,Xlabel,Ylabel</td>
<td>stringArray</td>
<td></td>
</tr>
<tr>
<td>datastr</td>
<td>List of TaQL strings : X,Y</td>
<td>stringArray</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

Plot data amplitude vs uv-distance for two Measurement set tables as a single panel plot. Operate on channel 1 for Stokes 1 and 2 using the DATA column.

```plaintext
tp.open(tabnames=['3c273.ms','3c48.ms'])
pop = {'nrows':1, 'ncols':1,'panel':1}
labels = ['Amplitude vs UVdist','uvdist','amplitude']
xystr = ['SQRT(SUMSQUARE(UVW[1:2]))','AMPLITUDE(DATA[1,1:2])']

tp.plotdata(poption=pop,labels=labels,datastr=xystr)
```

TaQL strings for the above example can also be written as follows.

```plaintext
xystr = ['SQRT(UVW[1]*UVW[1]+UVW[2]*UVW[2])','AMPLITUDE(DATA[1,1:2])']
```
Example

Multi-panel plotting

# uvdist for pol 1 and chan 1,2 on panel 211
pop = { 'nrows':2, 'ncols':1,'panel':1}
labels = ['Amplitude vs UVdist','uvdist','amplitude']
xystr = ['SQRT(SUMSQUARE(UVW[1:2]))','AMPLITUDE(DATA[1,1:2])']
tp.plotdata(poption=pop,labels=labels,datastr=xystr)

# uvdist for pol 2 and chan 1,2 (overplot=True) on panel 211
pop = { 'nrows':2, 'ncols':1,'panel':1, 'overplot':True}
labels = ['Amplitude vs UVdist','uvdist','amplitude']
xystr = ['SQRT(SUMSQUARE(UVW[1:2]))','AMPLITUDE(DATA[2,1:2])']
tp.plotdata(poption=pop,labels=labels,datastr=xystr)

# uv coverage on panel 223
pop = { 'nrows':2, 'ncols':2,'panel':3, 'plotcolour':4}
labels = ['UV Coverage','u','v']
xystr = ['UVW[1]','UVW[2]','-UVW[1]','-UVW[2]']
 tp.plotdata(poption=pop,labels=labels,datastr=xystr)

# amp(data[1:2,1:10]) vs channel number on panel 224
pop = { 'nrows':2, 'ncols':2,'panel':4, 'plotcolour':1}
labels = ['Amplitude vs Baseline number','baseline number','amplitude']
xystr = ['28*ANTENNA1+ANTENNA2-(ANTENNA1-1)*(ANTENNA1+2)/2',
'AMPLITUDE(DATA[1:2,1:10])']
 tp.plotdata(poption=pop,labels=labels,datastr=xystr)

Example

Plotting with time formatting

1756
# vistime for 10 chans (timeplot=1)
pop = { 'nrows':1, 'ncols':1,'panel':1,'timeplot':True}
labels = ['Timeplot','time','amplitude']
xystr = ['TIME','AMPLITUDE(DATA[1:2,1:10])']
tp.plotdata(poption=pop,labels=labels,datastr=xystr)

Example

Cross-plots - take in a single TaQL expression involving an ArrayColumn, and use the column numbers of each Array per row of the table as the x-axis. In a measurement set, the DATA ArrayColumn contains 2D Arrays, each with NCHAN columns and NPOL rows. Plotting with 'CROSS' as the X-TaQL, uses channel numbers as the x-axis. The option 'crossdirection=True' can be used to plot with polarization on the x-axis.

pop = { 'nrows':2, 'ncols':1,'panel':1, 'plotcolour':2}
labels = ['Amplitude vs Channel number','chan','amplitude']
xystr = ['CROSS','AMPLITUDE(DATA[1:2,1:10])']
tp.plotdata(poption=pop,labels=labels,datastr=xystr)


1757
Example

To plot with multiple colours for each channel and polarization of an MS.

```python
#(multicolour=1, plotcolour > 5)
pop = {'nrows':1, 'ncols':1, 'panel':1, 'plotcolour':6, 'showflags':0, 'multicolour':True}
labels = ['Amplitude vs UVdist', 'uvdist', 'amplitude']
xystr = ['SQRT(SUMSQUARE(UW[1:2]))', 'AMPLITUDE(DATA[1:2,1:2])']
tp.plotdata(poption=pop, labels=labels, datastr=xystr)
```
**tableplot.iterplotstart - Function**

**3.2.2 Initialize plotting with an iteration axis**

**Description**

Begin a series of plots using subtables constructed via an iteration axes. In addition to plotdata parameters, set a list of iteration axes (Table column names) and use iterplotnext() to step through. Only forward step through is allowed.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>poption</td>
<td>Record of plot options default is nxpanels=1, nypanels=1, windowsize=6, aspectratio=0.8, fontsize=1.0</td>
</tr>
<tr>
<td>labels</td>
<td>List of strings : Title,Xlabel,Ylabel</td>
</tr>
<tr>
<td>datastr</td>
<td>List of TaQL strings : X,Y</td>
</tr>
<tr>
<td>iteraxes</td>
<td>List of strings : Iteration axes</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```plaintext
# Open a list of MS tables to plot from,
```

1759
# and initialize a plot of Amplitude vs UV distance for
# channel 1 and stokes 1, iterating over Antenna1

```python
tp.open(tabnames=['3c273.ms','3c48.ms'])
pop = { 'nrows':3, 'ncols':1, 'panel':1, 'plotcolour':1,
        'aspectratio':1.6}
iteraxes = ['ANTENNA1']
labels = ['Amplitude vs UVdist','uvdist','amplitude']
xystr = ['SQRT(SUMSQUARE(UVW[1:2]))','AMPLITUDE(DATA[1,1:2])']
tp.iterplotstart(poption=pop, labels=labels, datastr=xystr,
                 iteraxes=iteraxes)
```

tp.iterplotnext()

Example

To iterate over baseline and plot Amplitude vs time, for stokes 1, channel 1.

```python
pop = { 'nrows':4, 'ncols':1 }
labels = ['Amplitude vs UVdist (iterating over Baseline)',
          'uvdist','amplitude']
xystr = ['SQRT(SUMSQUARE(UVW[1:2]))','AMPLITUDE(DATA[1,1])']
iteraxes = ['ANTENNA1','ANTENNA2']
tp.iterplotstart(poption=plotopts,labels=labels,
                 datastr=xystr,iteraxes=iteraxes)
```
tableplot.replot.html

**tableplot.replot - Function**

3.2.2 Replot all existing panels and layers.

**Description**

Replot all existing panels and layers. To be used after a change of flag version, to get all visible plots to reflect the changed flags.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
</table>

**Returns**

bool
Description

Start/Continue plotting by stepping through the iteration axes. Call after `tp.iterplotstart()`. Returns 1 if additional iteration steps remain, 0 if last iteration has completed.

Arguments

Returns

int

Example

```python
# iterate through the data

tp.open(tabnames=['3c273.ms','3c48.ms'])
plotopts = {'aspectratio': 1.2, 'ncols': 2, 'nrows': 1}
labels = ['Amplitude vs UVdist (iterating over Antenna1)',
          'uvdist','amplitude']
xystr = ['SQRT(SUMSQURE(UVW[1:2])),AMPLITUDE(DATA[1,1])]'
iteraxes = ['ANTENNA1']
tp.iterplotstart(poption=plotopts,labels=labels,
                 datastr=xystr,iteraxes=iteraxes)
ret = tp.iterplotnext()
ret = tp.iterplotnext()
ret = tp.iterplotnext()
...
```
**tableplot.iterplotstop - Function**

3.2.2 Stop plot iterations.

**Description**

To be called at the end of the plot iterations, or in between if desired. Okay if ignored.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rmplotter</td>
<td>Indicates whether the plot window should be removed (true) from the display or left (false) allowed: bool Default: false</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
# iterate through and stop after 5 iterations of 2 plots per page

tp.open(tabnames=['3c273.ms','3c48.ms'])
plotopts = {'ncols': 2, 'nrows':1 } 
labels = ['Amplitude vs UVdist (iterating over Antenna1)',
         'uvdist','amplitude']
xystr = ['SQRT(SUMSQUARE(UVW[1:2]))','AMPLITUDE(DATA[1,1])']
iteraxes = ['ANTENNA1']
ret = tp.iterplotstart(poption=plotopts,labels=labels,
datastr=xystr,iteraxes=iteraxes)
ret = tp.iterplotnext()
ret = tp.iterplotnext()
ret = tp.iterplotnext()
```

1764
ret = tp.iterplotnext()
ret = tp.iterplotnext()
tp.iterplotstop()
### tableplot.markregions - Function

#### 3.2.2 Mark a rectangular region to flag

**Description**

Mark or specify a rectangular region to flag. Call without arguments to enable mouse based interactive region marking. Marked regions can be discarded via the ‘Alt’ key. Command-line region marking can be done by setting panel and region parameters. After marking flag regions, call `tp.flagdata()` or `tp.unflagdata()`.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nrows</td>
<td>Number of rows of panels</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>ncols</td>
<td>Number of columns of panels</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>panel</td>
<td>Panel number</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>region</td>
<td>[xmin,ymin,xmax,ymax] bounding box</td>
<td>doubleArray</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
tp.markregions(nrows=2, ncols=1, panel=1, region=[300.0, 400.0, 0.090, 0.095])
```
tableplot.flagdata.html

**tableplot.flagdata - Function**

3.2.2 Flag Data for selected flag regions

**Description**

Set flags for all regions marked using `tp.markregions()`. The plot is automatically redrawn after applying flags. If reduction TaQL functions such as `sum,mean` are used, flags corresponding to all accessed values will be modified. For example, with a measurement set table, flagging on the mean amplitude of stokes 1 and channels 1 to 5, given by `MEAN(AMPLITUDE(DATA[1,1:5]))` results in flags being set for all 5 accessed channels.

For a measurement set, by default, flags are set only for accessed channels and stokes when the DATA column is used. However all channels/stokes can be flagged for the marked flag regions by setting the corresponding row flag.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
</table>

**Returns**

`bool`
**tableplot.unflagdata - Function**

3.2.2 Unset flags for all regions marked using `tp.markregions()`.

**Description**

Unset flags for all regions marked using `tp.markregions()`. This is similar to the `tp.flagdata()` function in all other respects.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
</table>

**Returns**

`bool`

**Example**

```python
# mark 2 flag regions on a multi-panel plot, one in panel 1 and one
# in panel 2. Then apply the flags and write to disk.
tp.markregions(panel=1)
tp.markregions(panel=2)
tp.unflagdata()
```
tableplot.locatedata.html

**tableplot.locatedata - Function**

3.2.2 Print info about data selected using tp.markregions().

**Description**

Print info about data selected using tp.markregions().

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnlist</td>
<td>List of strings : Column names (or TaQL expressions !)</td>
</tr>
<tr>
<td></td>
<td>allowed: stringArray</td>
</tr>
<tr>
<td></td>
<td>Default: bool</td>
</tr>
</tbody>
</table>

**Returns**

bool
tableplot.clearflags.html

**tableplot.clearflags - Function**

3.2.2 Clear all flags in the table.

**Description**

Clear all flags from the table. This may eventually be modified to allow for selective un-flagging of previously flagged regions (specified by indexing into a stored history of marked flag-regions).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>roottable</td>
<td>false : clear flags for the current sub-selection; true : clear flags for root table</td>
</tr>
<tr>
<td>allowed</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
# clear all flags from two measurement set tables

tp.open(tabnames=['3c273.ms','3c48.ms'])
tp.clearflags()
tp.done()
```

1771
tableplot.saveflagversion - Function

3.2.2 Save current flags with a version name.

Description

Save current flags with a version name. This applies to the last opened Tables.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Version name</th>
</tr>
</thead>
<tbody>
<tr>
<td>versionname</td>
<td>Version name</td>
</tr>
<tr>
<td>allowed: string</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Comment for this flag table</th>
</tr>
</thead>
<tbody>
<tr>
<td>comment</td>
<td>Comment for this flag table</td>
</tr>
<tr>
<td>allowed: string</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inputs</th>
<th>merge type</th>
</tr>
</thead>
<tbody>
<tr>
<td>merge</td>
<td>merge type</td>
</tr>
<tr>
<td>allowed: string</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool

![Image of a tableplot.saveflagversion function with version name, comment, and merge details.](image-url)
tableplot.restoreflagversion.html

**tableplot.restoreflagversion - Function**

3.2.2 Restore flags from a saved flag_version.

**Description**

Restore flags from a saved flag_version. This applies to the last opened Tables.

- **versionname**: name of flag version to restore to main table
- **merge**: Type of operation to perform during restoration.
  - merge = replace: replaces the main table flags
  - merge = and: logical AND with main table flags
  - merge = or: logical OR with main table flags
  - Default: replace.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>versionname</td>
<td>Version name</td>
</tr>
<tr>
<td>allowed: stringArray</td>
<td></td>
</tr>
<tr>
<td>merge</td>
<td>merge type</td>
</tr>
<tr>
<td>allowed: string</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool
**tableplot.deleteflagversion.html**

**tableplot.deleteflagversion - Function**

Delete a saved flag version.

**Description**

Delete a saved flag version. This applies to the last opened Tables.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Version name</th>
</tr>
</thead>
<tbody>
<tr>
<td>versionname</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>stringArray</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool
表plot.getflagversionlist.html

**tableplot.getflagversionlist - Function**

3.2.2 Print out a list of saved flag versions.

**Description**

Print out a list of saved flag versions. This applies to the last opened Tables.

**Arguments**

Inputs

**Returns**

bool
tableplot.clearplot.html

**tableplot.clearplot - Function**

3.2.2 Clear a plot.

**Description**

Clear a plot. An empty argument list (i.e., clearplot()) or clearplot(0) clears all plots currently visible whereas clearplot(nrows, ncols, panel) clears a plot on a specified panel.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nrows</td>
<td>Number of rows of panels</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>ncols</td>
<td>Number of columns of panels</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>panel</td>
<td>Panel number (index)</td>
<td>int</td>
<td>0</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```python
# clear all flags from two measurement set tables

tp.open(tabnames=['3c273.ms', '3c48.ms'])
tp.clearflags()
```

1776
tableplot.done.html

**tableplot.done - Function**

3.2.2 Clean up the tableplot tool

**Description**

Clean up the tableplot tool, and make it ready for tp.open() again.

**Arguments**

**Returns**

bool
Chapter 4

Package ThirdParty

The ThirdParty package contains modules that interface to 3rd party software.

4.1 atmosphere - Module

Module for interfacing to Juan R. Padro’s ATM library

Description

This is a thin-interface to Juan R. Padro’s proprietary ATM library atmlib. The atm library can be used to calculate atmospheric opacity and phase, and to perform radiative transfer calculations. In CASA, it is used for calibration, phase correction and for simulations. See Juan Padro’s web page for further information about the library structure and for information about other uses.
4.1.1 atmosphere - Tool

Atmosphere model

Requires:

Synopsis

Methods

- **atmosphere**: Construct an atmosphere tool
- **getAtmVersion**: Returns the version of ATM library.
- **listAtmosphereTypes**: Returns a list of atmospheric types used by ATM.
- **initAtmProfile**: Set initial atmospheric profile for atmosphere tool
- **updateAtmProfile**: Update basic atmospheric parameters of atmosphere tool
- **getBasicAtmParms**: Gets the current basic atmospheric parameters of the model.
- **getNumLayers**: Returns the number of layers in the atmospheric profile.
- **getGroundWH2O**: get the zenith column of water vapor
- **getAtmosphericProfile**: get atmospheric profile
- **initSpectralWindow**: initialize spectral window
- **addSpectralWindow**: add a new spectral window
- **getNumSpectralWindows**: Get number of spectral windows
- **getNumChan**: return the number of channels of ith band
- **getRelChan**: Get the reference channel of a given spectral window
- **getRelFreq**: Get the reference frequency of given spectral window
- **getChanSep**: Get the channel separation for regularly spaced grid for spectral window
- **getChanFreq**: Get the channel frequency for a given grid point for the specified spectral window
- **getSpectralWindow**: Get the spectral grid for the specified spectral window
- **getBandwidth**: Get the grid position for a given frequency in the specified spectral window
- **getMinFreq**: Get lowest frequency channel for the specified spectral window
- **getMaxFreq**: Get highest frequency channel for the specified spectral window
- **getDryOpacity**: get the integrated Dry Opacity along the atmospheric path for channel nc in spectral window swpId
- **getDryContOpacity**: get the integrated Dry Continuum Opacity along the atmospheric path for channel nc in spectral window swpId
- **getO2LinesOpacity**: get the integrated O2 Lines Opacity along the atmospheric path for channel nc in spectral window swpId
- **getO3LinesOpacity**: get the integrated O3 Lines Opacity along the atmospheric path for channel nc in spectral window swpId
- **getCOLinesOpacity**: get the integrated CO Lines Opacity along the atmospheric path for channel nc in spectral window swpId
- **getN2OLinesOpacity**: get the integrated N2O Lines Opacity along the atmospheric path for channel nc in spectral window swpId
- **getWetOpacity**: get the integrated Wet Opacity along the atmospheric path for channel nc in spectral window swpId
- **getH2OLinesOpacity**: get the integrated H2O Lines Opacity along the atmospheric path for channel nc in spectral window swpId
- **getH2OContOpacity**: get the integrated H2O Continuum Opacity along the atmospheric path for channel nc in spectral window swpId
- **getDryOpacitySpec**: get the integrated Dry optical depth along the atmospheric path on each channel of a band
- **getWetOpacitySpec**: get the integrated Wet optical depth along the atmospheric path on each channel of a band
- **getDispersivePhaseDelay**: get the integrated zenith H2O Atmospheric Phase Delay
- **getDispersiveWetPhaseDelay**: get the integrated dispersive wet Atmospheric Phase Delay
getNonDispersiveWetPhaseDelay get the integrated nondispersive wet Atmospheric Phase Delay
getNonDispersiveDryPhaseDelay get the integrated nondispersive dry Atmospheric Phase Delay
getNonDispersivePhaseDelay get the integrated zenith H2O Atmospheric Phase Delay (Non-Dispersive part)
getDispersivePathLength get the integrated Atmospheric Dispersive Path
getDispersiveWetPathLength get the integrated wet Atmospheric Dispersive Path
getNonDispersiveWetPathLength get the integrated dry Atmospheric NonDispersive Path
getNonDispersiveWetPhaseLength get the integrated wet Atmospheric NonDispersive Path
getO2LinesPathLength get the integrated O2 lines Path
getO3LinesPathLength get the integrated O3 lines Path
getCOLinesPathLength get the integrated CO lines Path
getN2OLinesPathLength get the integrated N2O lines Path
getNonDispersivePathLength get the integrated Atmospheric Non-Dispersive Path
getNonDispersiveWetPathLength get the integrated wet Atmospheric Non-Dispersive Path
getNonDispersiveDryPathLength get the integrated dry Atmospheric NonDispersive Path
getAbsH2OLines Get H2O lines Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid
getAbsH2OCont Get H2O continuum Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid
getAbsO2Lines Get O2 lines Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid
getAbsDryCont Get Dry Continuum Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid
getAbsO3Lines Get O3 lines Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid
getAbsCOLines Get CO lines Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid
getAbsN2OLines Get N2O lines Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid
getTotalDry get Total Dry Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid
getTotalWet get total wet absorption coefficient at layer nl and frequency channel nf in spectral window spwid
setUserWH2O set the user zenith water vapor column
getUserWH2O get the user zenith water vapor column
setAirMass Set the air mass
getAirMass Get the air mass
setSkyBackgroundTemperature Set the sky background temperature
getSkyBackgroundTemperature Set the sky background temperature
getAverageTebbSky Returns average equiv. BB Temp
getTebbSky Returns equiv. BB Temp
getAverageTrjSky Returns equiv. BB Temp on each channel of a band
getTrjSky Returns the average Rayleigh-Jeans Temperature
getTrjSkySpec Returns the Rayleigh-Jeans Temperatures on each channel of a band
Description

This is used to construct an atmosphere tool.

Arguments

Inputs

Returns
atmosphere

Example

A default atmosphere tool is created automatically during casapy startup and defined as 'at'. Manual tool construction is done this way:

at = casac.atmosphere()
atmosphere.getAtmVersion.html

atmosphere.getAtmVersion - Function

4.1.1 Returns the version of ATM library.

Description

Returns the version of ATM library implemented to this tool.

Arguments

Returns

string

Example

at.getAtmVersion()
# 'ATM-0_5_0'
atmosphere.listAtmosphereTypes.html

**atmosphere.listAtmosphereTypes** - Function

4.1.1 Returns a list of atmospheric types used by ATM.

### Description

Returns a list of index numbers and corresponding atmosphere types used by the ATM library.

### Arguments

### Returns

`stringArray`

### Example

```javascript
at.listAtmosphereTypes()
# ['1 - TROPICAL', '2 - MIDLATSUMMER', '3 - MIDLATWINTER',
#  '4 - SUBARTSUMMER', '5 - SUBARTWINTER']
```
atmosphere.initAtmProfile.html

**atmosphere.initAtmProfile - Function**

4.1.1 Set initial atmospheric profile for atmosphere tool

**Description**

An atmospheric profile is composed of 4 quantities as a function of altitude $z$:
* the layer thickness
* the pressure $P$
* the temperature $T$
* the gas densities for H2O, O3, CO and N2O.

This method is needed for computing the absorption and phase coefficients, as well as for performing radiative transfer calculations (only layer thickness/T are needed).

This method builds an atmospheric profile that can be used to calculate absorption and phase coefficients, as well as to perform forward and/or retrieval radiative transfer calculations. It is composed of a set of parameters needed to build a layer thickness/P/T/gas densities profile from simple parameters currently available at observatories (from weather stations for example) using functions from the ATM library. The set of input parameters consists of the pressure $P$, the temperature $T$ and the relative humidity at the ground, the altitude of the site, the tropospheric temperature lapse rate,... The profile is built as: thickness of the considered atmospheric layers above the site, and mean $P,T,H2O,O3,CO,N2O$ in them. The total number of atmospheric layers in the particular profile is also available (a negative value indicates an error). The zenith column of water vapor can be calculated by simply integrating the H2O profile.

**Arguments**
### Inputs

<table>
<thead>
<tr>
<th>Variable</th>
<th>Unit</th>
<th>Description</th>
<th>Allowed Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>altitude</td>
<td>m</td>
<td>Site altitude - Quantity with units of altitude, meter</td>
<td>doublem</td>
<td>5000.</td>
</tr>
<tr>
<td>temperature</td>
<td>K</td>
<td>Ambient Temperature - Quantity with units of temperature, K</td>
<td>doubleK</td>
<td>270.0</td>
</tr>
<tr>
<td>pressure</td>
<td>mbar</td>
<td>Ambient pressure - Quantity with units of pressure, mbar</td>
<td>doublembar</td>
<td>560.0</td>
</tr>
<tr>
<td>maxAltitude</td>
<td>km</td>
<td>altitude of the top of the modelled atmosphere - Quantity with dimension of length, and units of kilometer</td>
<td>doublekm</td>
<td>48.0</td>
</tr>
<tr>
<td>humidity</td>
<td></td>
<td>used to guess water (0-100)</td>
<td>double</td>
<td>20.0</td>
</tr>
<tr>
<td>dTem, dh</td>
<td>K/km</td>
<td>the derivative of temperature with respect to height - Quantity with units of K/km</td>
<td>doubleK/km</td>
<td>-5.6</td>
</tr>
<tr>
<td>dP</td>
<td>mbar</td>
<td>initial pressure step - Quantity with the units of pressure, mbar</td>
<td>doublembar</td>
<td>10.0</td>
</tr>
<tr>
<td>dPm</td>
<td></td>
<td>pressure multiplicative factor for steps</td>
<td>double</td>
<td>1.2</td>
</tr>
<tr>
<td>h0</td>
<td>km</td>
<td>scale height for water( exp distribution ) - Quantity with the dimension of length, and units of kilometer</td>
<td>doublekm</td>
<td>2.0</td>
</tr>
<tr>
<td>atmType</td>
<td></td>
<td>atmospheric type 1(tropical),2(mid latitude summer),3(mid latitude winter), 4(subarctic summer),5(subarctic winter), dimensionless</td>
<td>int</td>
<td>1</td>
</tr>
</tbody>
</table>

### Returns

string 1785
Example

tmp = qa.quantity(270.0, 'K')
pre = qa.quantity(560.0, 'mbar')
hum = 20.0
alt = qa.quantity(5000, 'm')
h0 = qa.quantity(2.0, 'km')
wvl = qa.quantity(-5.6, 'K/km')
mxA = qa.quantity(48, 'km')
dpr = qa.quantity(10.0, 'mbar')
dpm = 1.2
att = 1
myatm = at.initAtmProfile(alt, tmp, pre, mxA, hum, wvl, dpr, dpm, h0, att)
print myatm

# BASIC ATMOSPHERIC PARAMETERS TO GENERATE REFERENCE ATMOSPHERIC PROFILE
#
# Ground temperature T: 270 K
# Ground pressure P: 560 mb
# Relative humidity rh: 20 %
# Scale height h0: 2 km
# Pressure step dp: 10 mb
# Altitude alti: 5000 m
# Attitude top atm profile: 48 km
# Pressure step factor: 1.2
# Tropospheric lapse rate: -5.6 K/km
# Atmospheric type: TROPICAL
#
# Built atmospheric profile with 20 layers.
atmosphere.updateAtmProfile.html

**atmosphere.updateAtmProfile - Function**

4.1.1 Update basic atmospheric parameters of atmosphere tool

**Description**

This is used to update the atmosphere tool when basic atmospheric parameters change.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Type</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>altitude</td>
<td>m</td>
<td>Site altitude - Quantity with units of altitude, meter</td>
<td>double m</td>
<td>5000.0</td>
</tr>
<tr>
<td>temperature</td>
<td>K</td>
<td>Ambient ground temperature - Quantity with units of temperature, K</td>
<td>double K</td>
<td>270.0</td>
</tr>
<tr>
<td>pressure</td>
<td>mbar</td>
<td>Ambient ground pressure - Quantity with units of pressure, mbar</td>
<td>double mbar</td>
<td>560.0</td>
</tr>
<tr>
<td>humidity</td>
<td></td>
<td>Relative humidity used to guess water (0-100)</td>
<td>double</td>
<td>20.0</td>
</tr>
<tr>
<td>dTem_dh</td>
<td>K/km</td>
<td>Tropospheric Lapse Rate - the derivative of temperature with respect to height - Quantity with units of K/km</td>
<td>double K/km</td>
<td>-5.6</td>
</tr>
<tr>
<td>h0</td>
<td>km</td>
<td>Scale height for water (exp distribution) - Quantity with the dimension of length, and units of kilometer</td>
<td>double km</td>
<td>2.0</td>
</tr>
</tbody>
</table>

**Returns**

string
Example

```python
new_tmp = qa.quantity(275.0, 'K')
print at.updateAtmProfile(alt, new_tmp, pre, hum, wvl, h0)
# UPDATED BASIC ATMOSPHERIC PARAMETERS TO GENERATE REFERENCE ATMOSPHERIC PROFILE
#
# Ground temperature T: 275 K
# Ground pressure P: 560 mb
# Relative humidity rh: 20 %
# Scale height h0: 2 km
# Altitude alti: 5000 m
# Tropospheric lapse rate: -5.6 K/km
```
atmosphere.getBasicAtmParms.html

**atmosphere.getBasicAtmParms - Function**

4.1.1 Gets the current basic atmospheric parameters of the model.

**Arguments**
<table>
<thead>
<tr>
<th>Outputs</th>
<th>Description</th>
<th>Units</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>altitude</td>
<td>Site altitude - Quantity with units of altitude, meter</td>
<td>m</td>
<td>doublem</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>temperature</td>
<td>Ambient ground temperature - Quantity with units of temperature, K</td>
<td>K</td>
<td>doubleK</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pressure</td>
<td>Ambient pressure - Quantity with units of pressure, mbar</td>
<td>mbar</td>
<td>doublembar</td>
<td></td>
</tr>
<tr>
<td>maxAltitude</td>
<td>altitude of the top pf the modelled atmosphere - Quantity with dimension of length, and units of kilometer</td>
<td>km</td>
<td>doublekm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>humidity</td>
<td>Ground relative humidity used to guess water (0-100)</td>
<td></td>
<td>double</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dTem_dh</td>
<td>Current Tropospheric Lapse Rate (the derivative of temperature with respect to height) - Quantity with units of K/km</td>
<td>K/km</td>
<td>doubleK/km</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dP</td>
<td>initial pressure step - Quantity with the units of pressure, mb</td>
<td>mbar</td>
<td>doublembar</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dPm</td>
<td>pressure multiplicative factor for steps</td>
<td></td>
<td>double</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h0</td>
<td>Water vapor scale height ( exp distribution ) - Quantity with the dimension of length, and units of kilometer</td>
<td>km</td>
<td>doublekm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>atmType</td>
<td>atmospheric type used to describe the behaviour above the tropopause. 1(tropical),2(mid latitude summer),3(mid latitude winter), 4(subarctic summer),5(subarctic winter), dimensionless</td>
<td></td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

string

1790
Example

```python
p = at.getBasicAtmParms()
# returns a tuple of
# 0 - string listing of parameters, 1 - altitude, 2 - temperature,
# 3 - pressure, 4 - maxAltitude, 5 - humidity, 6 - dTem_dh,
# 7 - dP, 8 - dPm. 9 - h0, and 10 - atmType
print "Atmospheric type: ", p[10]
# Atmospheric type: TROPICAL
print "Ground temperature: ", p[2]['value'][0], p[2]['unit']
# Ground temperature: 288.16 K
print p[0]  # a "pretty" listing of all the parameters
# CURRENT ATMOSPHERIC PARAMETERS OF REFERENCE ATMOSPHERIC PROFILE
#
# Ground temperature T: 275 K
# Ground pressure P: 560 mbar
# Relative humidity rh: 20 %
# Scale height h0: 2 km
# Pressure step dp: 10 mbar
# Altitude alti: 5000 m
# Attitude top atm profile 48 km
# Pressure step factor 1.2
# Tropospheric lapse rate -5.6 K/km
# Atmospheric type: TROPICAL
#
# Atmospheric profile has 20 layers.
```
atmosphere.getNumLayers() - Function

4.1.1 Returns the number of layers in the atmospheric profile.

Arguments

Returns
int

Example

p = at.getProfile()
for i in range(at.getNumLayers()):
    # Print atmospheric profile returned by at.getProfile():
    # Layer thickness (idx=1), Temperature (idx=2),
    # Number density of water vapor (idx=4), and Pressure (idx=5)
    print p[1]['value'][i], p[2]['value'][i], p[4]['value'][i], p[5]['value'][i]
atmosphere.getGroundWH2O.html

**atmosphere.getGroundWH2O - Function**

4.1.1 get the zenith column of water vapor

**Description**

Method to get the zenith column of water vapor. It is computed by simply integrating the H2O profile:

**Arguments**

**Returns**

Quantity

**Example**

```python
w = at.getGroundWH2O()
print "Guessed water content: ", w["value"][0], w["unit"]
# Guessed water content:  2.6529103462750112 mm
```
atmosphere.getProfile.html

atmosphere.getProfile - Function

4.1.1 get atmospheric profile

Description

Get the atmospheric profile.

Arguments
<table>
<thead>
<tr>
<th>Outputs</th>
<th>unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>thickness m</td>
<td>m</td>
<td>thickness of every atmospheric layer - Quantum with a vector value and unit of length, m</td>
</tr>
<tr>
<td></td>
<td></td>
<td>allowed: double</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>temperature K</td>
<td>K</td>
<td>temperature of every atmospheric layer - Quantum with a vector value and unit of temperature, K</td>
</tr>
<tr>
<td></td>
<td></td>
<td>allowed: doubleK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>watermassdensity kg.m-3</td>
<td>kg.m-3</td>
<td>water vapor mass density content of every atmospheric layer - Quantum with a vector value and unit of kg.m-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>allowed: doublekg.m-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>water m-3</td>
<td>m-3</td>
<td>water vapor content of every atmospheric layer - Quantum with a vector value and unit of m-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>allowed: doublem-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>pressure Pa</td>
<td>Pa</td>
<td>pressure of every atmospheric layer - Quantum with a vector value and unit of Pascal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>allowed: doublePa</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>O3 m-3</td>
<td>m-3</td>
<td>O3 of every atmospheric layer - Quantum with a vector value and unit of m-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>allowed: doublem-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>CO m-3</td>
<td>m-3</td>
<td>CO of every atmospheric layer - Quantum with a vector value and unit of m-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>allowed: doublem-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>N2O m-3</td>
<td>m-3</td>
<td>N2O of every atmospheric layer - Quantum with a vector value and unit of m-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>allowed: doublem-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

string

**Example**

```python
p = at.getProfile()
```
# returns a tuple of
# 0 - string listing of layer values, and arrays of layer, 1 - thickness,
# 2 - temperature, 3 - watermassdensity, 4 - water (number density),
# 5 - pressure, 6 - O3 (number density), 7 - CO, 8 - N2O
for i in range(at.getNumLayers()):
    # Print atmospheric profile returned by at.getProfile():
    # Layer thickness (idx=1), Temperature (idx=2),
    # Number density of water vapor (idx=4), and Pressure (idx=5)
    print p[1]['value'][i], p[2]['value'][i], p[4]['value'][i],
    p[5]['value'][i]

print p[0]  # "pretty" listing of all layer parameters
**atmosphere.initSpectralWindow - Function**

### 4.1.1 initialize spectral window

**Description**

function that defines a spectral window, computes absorption and emission coefficients for this window, using the above atmospheric parameters.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nbands</td>
<td>number of spectral windows/bands. The value must be &gt; 0.</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>fCenter</td>
<td>center frequencies - Quantum with a vector value and unit of frequency, GHz</td>
<td>doubleGHz</td>
<td>90</td>
</tr>
<tr>
<td>fWidth</td>
<td>frequency width of band - Quantum with a vector value and unit of frequency, GHz</td>
<td>doubleGHz</td>
<td>0.64</td>
</tr>
<tr>
<td>fRes</td>
<td>resolution inside band - Quantum with a vector value and unit frequency, GHz. Default is for a single frequency.</td>
<td>doubleGHz</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Returns**

int

**Example**

1797
nb = 1
fC = qa.quantity(88., 'GHz')
fW = qa.quantity(0.5, 'GHz')
fR = qa.quantity(0.5, 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)

nb = 2
fC = qa.quantity([88., 90.], 'GHz')
fW = qa.quantity([0.5, 0.5], 'GHz')
fR = qa.quantity([0.125, 0.125], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
atmosphere.addSpectralWindow.html

**atmosphere.addSpectralWindow** - Function

4.1.1 add a new spectral window

**Description**

Add a new spectral window, uniformly sampled, this spectral window having no sideband.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>fCenter GHz</th>
<th>frequencies - Quantum with a double value and unit of frequency, GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fWidth GHz</td>
<td>frequency width of band - Quantum with a double value and unit of frequency, GHz</td>
</tr>
<tr>
<td></td>
<td>fRes GHz</td>
<td>resolution inside band - Quantum with a double value and unit frequency, GHz</td>
</tr>
</tbody>
</table>

| allowed: | doubleGHz | Default: | 350 |
| allowed: | doubleGHz | Default: | 0.008 |
| allowed: | doubleGHz | Default: | 0.002 |

**Returns**

int

**Example**

```python
fC2 = qa.quantity(350.0, 'GHz')
fW2 = qa.quantity(0.008, 'GHz')
fR2 = qa.quantity(0.002, 'GHz')
cmp = at.addSpectralWindow(fC2, fW2, fR2)
print "New spectral window has ", ncf, " channels"
```

1799
atmosphere.getNumSpectralWindows.html

**atmosphere.getNumSpectralWindows - Function**

4.1.1 Get number of spectral windows

**Arguments**

**Returns**

int

**Example**

```python
numSpw = at.getNumSpectralWindows()
print "There are ", numSpw, " spectral windows"
```
4.1.1 return the number of channels of ith band

Description

Return the number of channels of ith band (passes in as parameter).

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spwid</td>
<td>Int standing for identifier of bands (0-based).</td>
</tr>
<tr>
<td></td>
<td>The value must be ( \geq 0 ).</td>
</tr>
<tr>
<td></td>
<td>allowed:  int</td>
</tr>
<tr>
<td></td>
<td>Default:  0</td>
</tr>
</tbody>
</table>

Returns

int

Example

for spwid in range(at.getNumSpectralWindows()):
    numCh = at.getNumChan(spwd)
    print "Spectral window ", spwid, " has ", numCh, " frequency channels"
**atmosphere.getRefChan.html**

### atmosphere.getRefChan - Function

4.1.1 Get the reference channel of a given spectral window

**Description**

Return the reference channel of the given spectral window

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

**Returns**

int

**Example**

```python
rc = at.getRefChan()
print "Reference channel retrieved: ", rc
```

---

1803
atmosphere.getRefFreq.html

**atmosphere.getRefFreq - Function**

4.1.1 Get the reference frequency of given spectral window

**Description**

Return the reference frequency of the given spectral window

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be ( \geq 0 ).</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

**Returns**

Quantity

**Example**

```python
rf = at.getRefFreq()
print "Reference frequency retrieved: ", rf[\'value\'][0], rf[\'unit\']
```

---

1804
4.1.1 Get the channel separation for regularly spaced grid for spectral window

Description

Return the channel separation of the given spectral window

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be ( \geq 0 ).</td>
<td>int</td>
<td>0</td>
</tr>
</tbody>
</table>

Returns

Quantity

Example

```python
cs = at.getChanSep()
print "Channel separation retrieved: ", cs[\'value\'][0], cs[\'unit\']
```
atmosphere.getChanFreq.html

atmosphere.getChanFreq - Function

4.1.1 Get the channel frequency for a given grid point for the specified spectral window.

Description

Return the channel frequency for a given grid point for the specified spectral window.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chanNum</td>
<td>Int standing for channel number (0-based). The value must be &gt;= 0. allowed: int Default: 0</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0. allowed: int Default: 0</td>
</tr>
</tbody>
</table>

Returns

Quantity

Example

```python
for spwid in range(at.getNumSpectralWindows()):
    numCh = at.getNumChan(spwid)
    print "Spectral window ", spwid, " has ", numCh, " frequency channels"
    for n in range(numCh):
        freq = at.getChanFreq(n, spwid)
        print "Channel ", n, " Frequency:", freq['value'][0], freq['unit']
```

1806
atmosphere.getSpectralWindow.html

**atmosphere.getSpectralWindow - Function**

*4.1.1* Get the spectral grid for the specified spectral window.

**Description**

Return the spectral grid for the specified spectral window.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

**Returns**

Quantity

**Example**

```python
print at.getSpectralWindow()['value'],at.getSpectralWindow()['unit']

sg = at.getSpectralWindow()
for i in range(len(sg['value'])):
    print sg['value'][i], sg['unit']
```

1808
atmosphere.getChanNum.html

**atmosphere.getChanNum - Function**

4.1.1 Get the grid position for a given frequency in the specified spectral window.

**Description**

Return the channel number for given frequency in the specified spectral window relative to the reference channel number.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>freq GHz</td>
<td>Frequency</td>
</tr>
<tr>
<td>allowed: doubleGHz</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
</tr>
<tr>
<td>allowed: int</td>
<td></td>
</tr>
<tr>
<td>Default: 0</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

double

**Example**

```python
# List current spectral window setting of SPW0
at.getRefFreq()['value'][0], at.getRefFreq()['unit']
# (90.0, 'GHz')
print at.getChanSep()['value'][0], at.getChanSep()['unit']
# 10.0 MHz
at.getRefChan()
# 32

# Get grid positions
```
at.getChanNum(qa.quantity(90., 'GHz'))
# 0.0

at.getChanNum(qa.quantity(90., 'GHz'), 0)
# 0.0

at.getChanNum(qa.quantity(90.08, 'GHz'), 0)
# 8.0

at.getChanNum(qa.quantity(89.985, 'GHz'), 0)
# -1.5

at.getChanNum(qa.quantity(89.98, 'GHz'), 0)
# -2.0
**atmosphere.getBandwidth** - Function

4.1.1 Get the frequency range encompassing the list of frequency grid points for the specified spectral window.

**Description**

Get the frequency range encompassing the list of frequency grid points for the specified spectral window.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based).</td>
<td>int</td>
<td>0</td>
</tr>
</tbody>
</table>

**Returns**

Quantity

**Example**

```python
print "Total bandwidth retrieved: ", at.getBandwidth()[:0], at.getBandwidth()[:1]
```
atmosphere.getMinFreq.html

atmosphere.getMinFreq - Function

4.1.1 Get lowest frequency channel for the specified spectral window.

Description
Get lowest frequency channel for the specified spectral window.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

Returns
Quantity

Example

```
print "Frequency range: from ", at.getMinFreq()[’value’][0], " to ",
not getMaxFreq()[’value’][0], at.getMinFreq()[’unit’]
```
atmosphere.getMaxFreq.html

**atmosphere.getMaxFreq - Function**

4.1.1 Get highest frequency channel for the specified spectral window.

**Description**

Get highest frequency channel for the specified spectral window.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
<td>int</td>
<td>0</td>
</tr>
</tbody>
</table>

**Returns**

Quantity

**Example**

```python
print "Frequency range: from ", at.getMinFreq() ['value'][0], " to ", \
      at.getMaxFreq() ['value'][0], at.getMaxFreq() ['unit']
```

1813
**atmosphere.getDryOpacity - Function**

4.1.1 get the integrated Dry Opacity along the atmospheric path for channel nc in spectral window swpId

**Description**

Get the integrated Dry Opacity for one channel in a band.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Channel number (0-based; defaults to reference channel)</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

**Returns**

double

**Example**

```python
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
fR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total Dry Opacity at ", fC['value'][0], fC['unit'],
     " for 1.0 air mass: ", at.getDryOpacity()
```

1814
atmosphere.getDryContOpacity.html

atmosphere.getDryContOpacity - Function

4.1.1 get the integrated Dry Continuum Opacity along the atmospheric path for channel nc in spectral window spwid

Description

Get the integrated Dry Continuum Opacity for one channel in a band.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Channel number (0-based; defaults to reference channel)</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

Returns

double

Example

```python
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
fR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total Dry Cont Opacity at ", fC['value'][0], fC['unit'], 
" for 1.0 air mass: ", at.getDryContOpacity()
```

1815
atmosphere.getO2LinesOpacity.html

atmosphere.getO2LinesOpacity - Function

4.1.1 get the integrated O2 Lines Opacity along the atmospheric path for channel nc in spectral window spwid

Description

Get the integrated O2 Lines Opacity for one channel in a band.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Channel number (0-based; defaults to reference channel)</td>
<td>allowed: int</td>
<td>Default: -1</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
<td>allowed: int</td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

Returns
double

Example

```python
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
fR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total O2 Lines Opacity at ", fC['value'][0], fC['unit'], " for 1.0 air mass: ", at.getO2LinesOpacity()
```

1816
4.1.1 get the integrated O3 Lines Opacity along the atmospheric path for channel nc in spectral window spwid

Description
Get the integrated O3 Lines Opacity for one channel in a band.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Channel number (0-based; defaults to reference channel)</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be ( \geq 0 ).</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

Returns
double

Example

```python
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
fR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total O3 Lines Opacity at ", fC['value'][0], fC['unit'], ", " for 1.0 air mass: ", at.getO3LinesOpacity()
```
atmosphere.getCOLinesOpacity.html

**atmosphere.getCOLinesOpacity** - Function

4.1.1 get the integrated CO Lines Opacity along the atmospheric path for channel nc in spectral window spwid

**Description**

Get the integrated CO Lines Opacity for one channel in a band.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Channel number (0-based; defaults to reference channel)</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
<td>int</td>
<td>0</td>
</tr>
</tbody>
</table>

**Returns**

double

**Example**

```python
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
fR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total CO Lines Opacity at ", fC['value'][0], fC['unit'], " for 1.0 air mass: ", at.getCOLinesOpacity()
```

1818
atmosphere.getN2OLinesOpacity.html

atmosphere.getN2OLinesOpacity - Function

get the integrated N2O Lines Opacity along the atmospheric path for channel nc in spectral window spwid

Description

Get the integrated N2O Lines Opacity for one channel in a band.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Channel number (0-based; defaults to reference channel)</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be \geq 0.</td>
</tr>
</tbody>
</table>

Returns

double

Example

```python
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
fR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total N2O Lines Opacity at ", fC['value'][0], fC['unit'],
      " for 1.0 air mass: ", at.getN2OLinesOpacity()
```

1819
atmosphere.getWetOpacity.html

**atmosphere.getWetOpacity - Function**

4.1.1 get the integrated Wet Opacity along the atmospheric path for channel nc in spectral window spwid

**Description**

Get the integrated Wet Opacity for one channel in a band.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Channel number (0-based; defaults to reference channel)</td>
<td>allowed: int</td>
<td>Default: -1</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
<td>allowed: int</td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

**Returns**

Quantity

**Example**

```python
for i in range(at.getNumSpectralWindows()):
    for j in range(at.getNumChan(i)):
        print "Frequency: ", at.getChanFreq(j, i)['value'][0], at.getChanFreq(j, i)['unit']
        print "Wet opacity:", at.getWetOpacity(j, i)['value'][0], at.getWetOpacity(j, i)['unit'], " for ", at.getUserWH2O()['value'][0], at.getUserWH2O()['unit'], " H2O
```

1820
atmosphere.getH2OLinesOpacity - Function

get the integrated H2O Lines Opacity along the atmospheric path for channel nc in spectral window spwid

Description

Get the integrated H2O Lines Opacity for one channel in a band.

Arguments

Inputs

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Channel number (0-based; defaults to reference channel)</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
<td>int</td>
<td>0</td>
</tr>
</tbody>
</table>

Returns
double

Example

```python
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
fR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total H2O Lines Opacity at ", fC['value'][0], fC['unit'], 
" for 1.0 air mass: ", at.getH2OLinesOpacity()
```
atmosphere.getH2OContOpacity.html

**atmosphere.getH2OContOpacity - Function**

4.1.1 get the integrated H2O Continuum Opacity along the atmospheric path for channel nc in spectral window spwid

**Description**

Get the integrated H2O Continuum Opacity for one channel in a band.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Channel number (0-based; defaults to reference channel)</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
</tr>
</tbody>
</table>

**Returns**

double

**Example**

```python
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
fR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total H2O Cont Opacity at ", fC['value'][0], fC['unit'],
" for 1.0 air mass: ", at.getH2OContOpacity()
```

1822
atmosphere.getDryOpacitySpec.html

**atmosphere.getDryOpacitySpec - Function**

4.1.1 get the integrated Dry optical depth along the atmospheric path on each channel of a band

**Description**

Get the integrated Dry optical depth along the atmospheric path on each channel in a band.

**Arguments**

<table>
<thead>
<tr>
<th>Outputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>dryOpacity</td>
<td>dry opacity for each channel</td>
<td>allowed: doubleArray</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

**Returns**

int

**Example**

```python
at.getDryOpacitySpec()
# (8,
#  array([0.12113794420465548, 0.11890122206854335,
#         0.11713584932434795, 0.11572780449702716,
#         0.11459567027114714, 0.11368004975916192,
#         0.1129367842232195, 0.11233248854020933]))
```

1823
atmosphere.getWetOpacitySpec.html

atmosphere.getWetOpacitySpec - Function

4.1.1 get the integrated Wet optical depth along the atmospheric path on each channel of a band

Description

Get the integrated optical Wet depth along the atmospheric path on each channel in a band.

Arguments

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Inputs</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>wetOpacity mm-1</td>
<td>spwid Int</td>
<td>int</td>
</tr>
<tr>
<td>wet opacity for each channel in band - Quantum with a vector value and unit of mm-1</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
<td></td>
</tr>
<tr>
<td>allowed: double mm-1</td>
<td>allowed: int</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>Default: 0</td>
<td></td>
</tr>
</tbody>
</table>

Example

```python
sw=at.getWetOpacitySpec()
# returns a tuple of
# 0 - The number of channels and
# 1 - an quantity array of wet opacity for each channel in band
sw[1][‘value’]
# array([1.7225454913767393, 1.7204246078103735,
#        1.7188614166349163, 1.7179243635081174,
#        1.7188614166349163, 1.7179243635081174,
```
Another example:
for s in range(at.getNumSpectralWindows()):
    print "band", s
    for i in range(at.getNumChan(0)):
        print " - dryOpacity ", at.getDryOpacitySpec(spwid=s)[1][i], " wet Opacity/mm ", at.getWetOpacitySpec(spwid=s)[1][\'value\'][i]
atmosphere.getDispersivePhaseDelay - Function

4.1.1 get the integrated zenith H2O Atmospheric Phase Delay

Description

Get the integrated zenith H2O Atmospheric Phase Delay (Dispersive part) for the current conditions, for channel number nc of spectral window spwid.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Channel number (0-based; defaults to reference channel)</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be ( \geq 0 ).</td>
</tr>
</tbody>
</table>

Returns

Quantity

Example

```python
w = at.getUserWH2O()
numSpw = at.getNumSpectralWindows()
for spwid in range(numSpw):
    numCh = at.getNumChan(spwid)
    print "Spectral window ", spwid, " has ", numCh, " frequency channels"
    for n in range(numCh):
        freq = at.getChanFreq(n, spwid)
        print "Total Dispersive Phase Delay at ",freq['value'][0], freq['unit'], " if
        (at.getDispersivePhaseDelay(n, spwid)['value'][0]/(w['value'][0]))," dere
        ((100*at.getDispersivePhaseDelay(n, spwid)['value'][0]/(w['value'][0]))/1826
```

1826
"% of the Non-dispersive one )"
atmosphere.getDispersiveWetPhaseDelay.html

**atmosphere.getDispersiveWetPhaseDelay - Function**

4.1.1 get the integrated dispersive wet Atmospheric Phase Delay

**Description**

Function to retrieve the integrated wet Atmospheric Phase Delay (Dispersive part) along the atmospheric path corresponding to the 1st guess water column.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>nc</strong></td>
<td>Channel number (0-based; defaults to reference channel)</td>
</tr>
<tr>
<td>allowed: <strong>int</strong></td>
<td>Default: -1</td>
</tr>
<tr>
<td><strong>spwid</strong></td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
</tr>
<tr>
<td>allowed: <strong>int</strong></td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

**Returns**

Quantity

**Example**

```python
w = at.getUserWH2O()
numSpw = at.getNumSpectralWindows()
for spwid in range(numSpw):
    numCh = at.getNumChan(spwid)
    print "Spectral window ", spwid, " has ", numCh, " frequency channels"
    for n in range(numCh):
        freq = at.getChanFreq(n, spwid)
        print "Total Dispersive Wet Phase Delay at ", freq[‘value’][0], freq[‘unit’]
        print at.getDispersiveWetPhaseDelay(n, spwid)[‘value’][0]/(w[‘value’][0]),
        print "degrees per mm of water vapor (",
        print ((100*at.getDispersiveWetPhaseDelay(n, spwid)[‘value’][0])/(w[‘value’][0]))/(at.getNonDispersiveWetPhaseDelay(n, spwid)[‘value’][0]/w[‘value’][0]),
```

1828
"% of the Non-dispersive one )"
atmosphere.getNonDispersiveWetPhaseDelay.html

atmosphere.getNonDispersiveWetPhaseDelay - Function

4.1.1 get the integrated nondispersive wet Atmospheric Phase Delay

Description

Function to retrieve the integrated wet Atmospheric Phase Delay (NonDispersive part) along the atmospheric path corresponding to the 1st guess water column.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Channel number (0-based; defaults to reference channel)</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>int</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
</tr>
</tbody>
</table>

Returns

Quantity

Example

```python
w = at.getUserWH2O()
numSpw = at.getNumSpectralWindows()
for spwid in range(numSpw):
    numCh = at.getNumChan(spwid)
    print "Spectral window ", spwid, " has ", numCh, " frequency channels"
    for n in range(numCh):
        freq = at.getChanFreq(n, spwid)
        print "Total Dispersive Wet Phase Delay at ", freq['value'][0], freq['unit']
        (at.getDispersiveWetPhaseDelay(n, spwid)]['value'][0])/(w['value'][0]),
```

1830
\[
\frac{(100 \times \text{at.getDispersiveWetPhaseDelay}(n, \text{spwid})[\text{value}][0])}{w[\text{value}][0]} \times \% \text{ of the Non-dispersive one }
\]
atmosphere.getNonDispersiveDryPhaseDelay.html

atmosphere.getNonDispersiveDryPhaseDelay - Function

4.1.1 get the integrated nondispersive dry Atmospheric Phase Delay

Description

Function to retrieve the integrated dry Atmospheric Phase Delay (NonDispersive part) along the atmospheric path corresponding to the 1st guess water column.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Channel number (0-based; defaults to reference channel)</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>-1</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
</tbody>
</table>

Returns

Quantity

Example

```python
w = at.getUserWH2O()
numSpw = at.getNumSpectralWindows()
for spwid in range(numSpw):
    numCh = at.getNumChan(spwd)
    print "Spectral window ", spwid, " has ", numCh, " frequency channels"
    for n in range(numCh):
        freq = at.getChanFreq(n, spwid)
        print "Total Dispersive Dry Phase Delay at ", freq['value'][0], freq['unit']
        (at.getDispersiveDryPhaseDelay(n,spwid)['value'][0]/(w['value'][0]),'
```

1832
\[
((100 \times \text{at.getDispersiveDryPhaseDelay}(n, \text{spwid})['value'][0]) / (w['value'][0]))/
((\text{at.getNonDispersiveDryPhaseDelay}(n, \text{spwid})['value'][0]) / (w['value'][0]))
\]

"% of the Non-dispersive one \)"
atmosphere.getNonDispersivePhaseDelay.html

**atmosphere.getNonDispersivePhaseDelay - Function**

4.1.1 get the integrated zenith H2O Atmospheric Phase Delay (Non-Dispersive part)

**Description**

Get the integrated zenith H2O Atmospheric Phase Delay (Non-Dispersive part) for the current conditions, for channel number nc of spectral window spwid.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Channel number (0-based; defaults to reference channel)</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

**Returns**

Quantity

**Example**

```python
w = at.getUserWH2O()
umSpw = at.getNumSpectralWindows()
for spwid in range(numSpw):
    numCh = at.getNumChan(spwd)
    print "Spectral window ", spwid, " has ", numCh, " frequency channels"
    for n in range(numCh):
        freq = at.getChanFreq(n, spwid)
        print "Total Dispersive Phase Delay at ", freq[‘value’][0], freq[‘unit’], "
        print 1834
```

1834
(at.getDispersivePhaseDelay(n, spwid)['value'][0])/(\(w['value'][0]\))," degrees per mm of water vapor (\(\frac{(100\times at.getDispersivePhaseDelay(n, spwid)['value'][0])}{(w['value'][0])}\)/\(\frac{at.getNonDispersivePhaseDelay(n, spwid)['value'][0]}{w['value'][0]}\)) \("% of the Non-dispersive one \)"
atmosphere.getDispersivePathLength.html

**atmosphere.getDispersivePathLength - Function**

4.1.1 get the integrated Atmospheric Dispersive Path

**Description**

Retrieve the integrated Atmospheric Path length (Dispersive part) along the atmospheric path corresponding to the user water column for channel nc in spectral window spwid.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Channel number (0-based; defaults to reference channel)</td>
<td>int</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
<td>int</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

Quantity

**Example**

```python
w = at.getUserWH20()
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
fR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total Dispersive Delay at ", fC['value'][0], fC['unit'], " for 1.0 air mass: ",
       at.getDispersivePathLength()['value'][0] / w['value'][0], " meters per mm of water vapor",
print ",",100*(at.getDispersivePathLength()['value'][0] / w['value'][0])/(at.getNonDispersivePathLength() ['value'][0])
```

1836
"% of the Non-dispersive one)"
atmosphere.getDispersiveWetPathLength - Function

4.1.1 get the integrated wet Atmospheric Dispersive Path

Description

Retrieve the integrated wet Atmospheric Path length (Dispersive part) along the atmospheric path corresponding to the 1st guess water column for channel nc in spectral window spwid.

Arguments

Inputs

<table>
<thead>
<tr>
<th>nc</th>
<th>Channel number (0-based; defaults to reference channel)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

Returns

Quantity

Example

```python
w = at.getGroundWH2O()
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
fR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total Dispersive Delay at ", fC['value'][0], fC['unit'], " for 1.0 air mass: ",
     at.getDispersiveWetPathLength()['value'][0] / w['value'][0], " meters per mm of water"
print "",100*(at.getDispersiveWetPathLength()['value'][0] / w['value'][0])/(at.getNonDispersiveWetPathLength()['value'][0])
```

1838
"\% of the Non-dispersive one )"
atmosphere.getNonDispersiveWetPathLength.html

**atmosphere.getNonDispersiveWetPathLength** - Function

4.1.1 get the integrated wet Atmospheric NonDispersive Path

**Description**

Retrieve the integrated wet Atmospheric Path length (NonDispersive part) along the atmospheric path corresponding to the 1st guess water column for channel `nc` in spectral window `spwid`.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Channel number (0-based; defaults to reference channel)</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

**Returns**

Quantity

**Example**

```python
w = at.getGroundWH2O()
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
fR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total Dispersive Delay at " + fC['value'][0], fC['unit'], " for 1.0 air mass: ",
        at.getDispersiveWetPathLength()['value'][0] / w['value'][0], " meters per mm of water;
print "(" + 100*(at.getDispersiveWetPathLength()['value'][0] / w['value'][0])/(at.getNonDisperse
```

1840
"% of the Non-dispersive one )"
atmosphere.getNonDispersiveDryPathLength.html

atmosphere.getNonDispersiveDryPathLength - Function

4.1.1 get the integrated dry Atmospheric NonDispersive Path

**Description**

Retrieve the integrated dry Atmospheric Path length (NonDispersive part) along the atmospheric path corresponding to the 1st guess water column for channel nc in spectral window spwid.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Channel number (0-based; defaults to reference channel)</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be $\geq 0$.</td>
<td>int</td>
<td>0</td>
</tr>
</tbody>
</table>

**Returns**

Quantity

**Example**

```python
w = at.getGroundWH2O()
nc = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
fR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nc, fC, fW, fR)
print "Total Dispersive Delay at ", fC['value'][0], fC['unit'], " for 1.0 air mass: ",
    at.getDispersiveDryPathLength()['value'][0] / w['value'][0], " meters per mm of water"
print "(",100*(at.getDispersiveDryPathLength()['value'][0] / w['value'][0])/(at.getNonDispersiveDryPathLength()['value'][0]),
```

1842
"/% of the Non-dispersive one )"
atmosphere.getO2LinesPathLength.html

**atmosphere.getO2LinesPathLength - Function**

### 4.1.1 get the integrated O2 lines Path

**Description**

Retrieve the integrated Atmospheric Phase Delay (due to O2 Lines) along the atmospheric path corresponding to the 1st guess water column for channel nc in spectral window spwid.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Channel number (0-based; defaults to reference channel)</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be $\geq 0$.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

**Returns**

Quantity

**Example**

1844
atmosphere.getO3LinesPathLength.html

**atmosphere.getO3LinesPathLength - Function**

**4.1.1** get the integrated O3 lines Path

**Description**

Retrieve the integrated Atmospheric Phase Delay (due to O3 Lines) along the atmospheric path corresponding to the 1st guess water column for channel nc in spectral window spwid.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Channel number (0-based; defaults to reference channel)</td>
<td>int</td>
<td>-1</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
<td>int</td>
<td>0</td>
</tr>
</tbody>
</table>

**Returns**

Quantity

**Example**

1845
atmosphere.getCOLinesPathLength.html

atmosphere.getCOLinesPathLength - Function

4.1.1 get the integrated CO lines Path

Description

Retrieve the integrated Atmospheric Path length (due to CO Lines) along the atmospheric path corresponding to the 1st guess water column for channel nc in spectral window spwid.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Channel number (0-based; defaults to reference channel)</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

Returns

Quantity

Example

1846
atmosphere.getN2OLinesPathLength.html

**atmosphere.getN2OLinesPathLength - Function**

4.1.1 get the integrated N2O lines Path

**Description**

Retrieve the integrated Atmospheric Path length (due to N2O Lines) along the atmospheric path corresponding to the 1st guess water column for channel nc in spectral window spwid.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Channel number (0-based; defaults to reference channel)</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

**Returns**

Quantity

**Example**

```
1847
```
atmosphere.getNonDispersivePathLength.html

**atmosphere.getNonDispersivePathLength - Function**

4.1.1 get the integrated Atmospheric Non-Dispersive Path

**Description**

Get the integrated zenith H2O Atmospheric Path length (Non-Dispersive part) for the current conditions, for channel nc in spectral window spwid.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Channel number (0-based; defaults to reference channel)</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td>spwid</td>
<td>Int standing for spectral window id (0-based). The value must be &gt;= 0.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

**Returns**

Quantity

**Example**

```python
w = at.getUserWH2O()
nb = 1
fC = qa.quantity([850.0], 'GHz')
fW = qa.quantity([0.5], 'GHz')
fR = qa.quantity([0.5], 'GHz')
at.initSpectralWindow(nb, fC, fW, fR)
print "Total Dispersive Delay at ", fC['value'][0], fC['unit'], " for 1.0 air mass: ",\          
     at.getDispersivePathLength()['value'][0] / w['value'][0], " meters per mm of water vapor",
print ", (",100*(at.getDispersivePathLength()['value'][0] / w['value'][0])/(at.getNonDispersivePathLength()['value'][0])", "% of the Non-dispersive one )"
```

1848
atmosphere.getAbsH2OLines.html

**atmosphere.getAbsH2OLines - Function**

4.1.1 Get H2O lines Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid

**Description**

Accessor to get H2O lines Absorption Coefficient at layer nl, spectral window spwid and channel nf.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>nl</td>
<td>atmospheric layer number. The value must be &gt;= 0.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: int</td>
</tr>
<tr>
<td>nf</td>
<td>frequency channel number. The value must be &gt;= 0.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
<tr>
<td>spwid</td>
<td>spectral window id. The value must be &gt;= 0.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

**Returns**

Quantity

**Example**

```python
ac = at.getAbsH2OLines(0, 0, 0)
print "H2O lines absorption coefficient for layer 0, channel 0 is ", ac['value'][0], ac['unit']
```

---

1850
atmosphere.getAbsH2OCont - Function

4.1.1 Get H2O continuum Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid

Description

Get H2O continuum Absorption Coefficient at layer nl, spectral window spwid and frequency channel nf

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nl</td>
<td>atmospheric layer number. The value must be &gt;= 0.</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nf</td>
<td>frequency channel number. The value must be &gt;= 0.</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>spwid</td>
<td>spectral window id. The value must be &gt;= 0.</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Returns

Quantity

Example

```python
ac = at.getAbsH2OCont(0, 0, 0)
print "H2OCont absorption coefficient for layer 0, channel 0 is ", ac[‘value’][0], ac[‘unit’]
```
atmosphere.getAbsO2Lines - Function

4.1.1 Get O2 lines Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid

Description

Get O2 lines Absorption Coefficient at layer nl, spectral window spwid and frequency channel nf

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nl</td>
<td>atmospheric layer number. The value must be ( \geq 0 ).</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nf</td>
<td>frequency channel number. The value must be ( \geq 0 ).</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>spwid</td>
<td>spectral window id. The value must be ( \geq 0 ).</td>
<td>int</td>
<td>0</td>
</tr>
</tbody>
</table>

Returns

Quantity

Example

```python
ac = at.getAbsO2Lines(0, 0, 0)
print "O2 lines absorption coefficient for layer 0, channel 0 is ", ac['value'][0], ac['unit']
```

1852
atmosphere.getAbsDryCont.html

**atmosphere.getAbsDryCont - Function**

4.1.1 Get Dry Continuum Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid

**Description**

Get Dry Continuum Absorption Coefficient at layer nl, spectral window spwid and frequency channel nf

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nl</td>
<td>atmospheric layer number. The value must be ( \geq 0 ).</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>nf</td>
<td>frequency channel number. The value must be ( \geq 0 ).</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>spwid</td>
<td>spectral window id. The value must be ( \geq 0 ).</td>
<td>int</td>
<td>0</td>
</tr>
</tbody>
</table>

**Returns**

Quantity

**Example**

```python
ac = at.getAbsDryCont(0, 0, 0)
print "Dry Continuum absorption coefficient for layer 0, channel 0 is ", ac['value'][0], ac['unit']
```
atmosphere.getAbsO3Lines.html

**atmosphere.getAbsO3Lines - Function**

4.1.1 Get O3 lines Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid

**Description**

Get O3 lines Absorption Coefficient at layer nl, spectral window spwid and frequency channel nf

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Value Range</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nl</td>
<td>atmospheric layer number. The value must be ( \geq 0 ).</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>nf</td>
<td>frequency channel number. The value must be ( \geq 0 ).</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>spwid</td>
<td>spectral window id. The value must be ( \geq 0 ).</td>
<td>int</td>
<td>0</td>
</tr>
</tbody>
</table>

**Returns**

Quantity

**Example**

```python
ac = at.getAbsO3Lines(0, 0, 0)
print "O3 lines absorption coefficient for layer 0, channel 0 is ", ac['value'][0], ac['unit']
```
atmosphere.getAbsCOLines.html

**atmosphere.getAbsCOLines - Function**

4.1.1 Get CO lines Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid

**Description**

Get CO lines Absorption Coefficient at layer nl, spectral window spwid and frequency channel nf

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nl</td>
<td>atmospheric layer number. The value must be $\geq 0$.</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nf</td>
<td>frequency channel number. The value must be $\geq 0$.</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>spwid</td>
<td>spectral window id. The value must be $\geq 0$.</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

Quantity

**Example**

```python
ac = at.getAbsCOLines(0, 0, 0)
print "CO lines absorption coefficient for layer 0, channel 0 is ", ac['value'][0], ac['unit']
```

1855
atmosphere.getAbsN2OLines - Function

4.1.1 Get N2O lines Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid

Description

Get N2O lines Absorption Coefficient at layer nl, spectral window spwid and frequency channel nf

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nl</td>
<td>atmospheric layer number. The value must be &gt;= 0.</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nf</td>
<td>frequency channel number. The value must be &gt;= 0.</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>spwid</td>
<td>spectral window id. The value must be &gt;= 0.</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Returns

Quantity

Example

ac = at.getAbsN2OLines(0, 0, 0)
print "N2O lines absorption coefficient for layer 0, channel 0 is ", ac['value'][0], ac['unit']
atmosphere.getAbsTotalDry.html

atmosphere.getAbsTotalDry - Function

4.1.1 Get Total Dry Absorption Coefficient at layer nl and frequency channel nf in spectral window spwid

Description

Get total dry Absorption Coefficient at layer nl, spectral window spwid and frequency channel nf

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nl</td>
<td>atmospheric layer number. The value must be &gt;= 0.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>nf</td>
<td>frequency channel number. The value must be &gt;= 0.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
<tr>
<td>spwid</td>
<td>spectral window id. The value must be &gt;= 0.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
</tbody>
</table>

Returns

Quantity

Example

ac = at.getAbsTotalDry(0, 0, 0)
print "Total dry absorption coefficient for layer 0, channel 0 is ", ac['value'][0], ac['unit']
atmosphere.getAbsTotalWet.html

**atmosphere.getAbsTotalWet - Function**

4.1.1 Get total wet absorption coefficient at layer nl and frequency channel nf in spectral window spwid

**Description**

Get total wet absorption coefficient at layer nl, spectral window spwid and frequency channel nf

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nl</td>
<td>atmospheric layer number. The value must be $\geq 0$.</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>nf</td>
<td>frequency channel number. The value must be $\geq 0$.</td>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>spwid</td>
<td>spectral window id. The value must be $\geq 0$.</td>
<td>int</td>
<td>0</td>
</tr>
</tbody>
</table>

**Returns**

Quantity

**Example**

```python
ac = at.getAbsTotalWet(0, 0, 0)
print "Total wet absorption coefficient for layer 0, channel 0 is ", ac[\'value\'][0], ac[\'unit\']
```
atmosphere.setUserWH2O - Function

4.1.1 set the user zenith water vapor column

Description

Set user zenith water vapor column for forward radiative transfer calculations.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>wh2o</td>
<td>mm</td>
<td>User water vapor column</td>
</tr>
<tr>
<td></td>
<td>allowed: double mm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: 0.0</td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool

Example

\[
\text{wh2o=} \text{qa.quantity(0.8, }"\text{mm}")
\text{at.setUserWH2O(wh2o)}
\]
get the user zenith water vapor column

Description
Get user zenith water vapor column for forward radiative transfer calculations.

Arguments

Returns
Quantity

Example

print "water vapor column: ", at.getUserWH2O()['value'][0], at.getUserWH2O()['unit']
atmosphere.setAirMass.html

atmosphere.setAirMass - Function

4.1.1 Set the air mass

Description

Setter for air mass in SkyStatus without performing water vapor retrieval.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>airmass</td>
<td>Air Mass</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

Returns

bool

Example

at.setAirMass(1.51)
atmosphere.getAirMass.html

atmosphere.getAirMass - Function

4.1.1 Get the air mass

Description
Accessor to get airmass.

Arguments

Returns
double

Example

at.setAirMass(2.0)
print "(INPUT CHANGE) Air mass: ", at.getAirMass()
atmosphere.setSkyBackgroundTemperature.html

**atmosphere.setSkyBackgroundTemperature - Function**

4.1.1 Set the sky background temperature

**Description**

Set sky background temperature in SkyStatus without performing water vapor retrieval

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>tbgr K</td>
<td>sky background temperature</td>
<td>doubleK</td>
<td>2.73</td>
</tr>
</tbody>
</table>

**Returns**

bool

**Example**

```java
at.setSkyBackgroundTemperature(qa.quantity(2.73,'K'))
```

---

1863
atmosphere.getSkyBackgroundTemperature.html

**atmosphere.getSkyBackgroundTemperature** - Function

Get the sky background temperature

**Description**

Get the sky background temperature

**Arguments**

**Returns**

Quantity

**Example**

t = at.getSkyBackgroundTemperature()
print t['value'][0], t['unit']
# 2.73 K

1864
atmosphere.getAverageTebbSky.html

**atmosphere.getAverageTebbSky - Function**

4.1.1 Returns average equiv. BB Temp

**Description**

Returns the average Equivalent Blackbody Temperature in spectral window spwid, for the current conditions and a perfect sky coupling.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>spwid</strong></td>
<td>Spectral window (0-based). The value must be &gt;= 0.</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td><strong>wh2o</strong></td>
<td>User specified water column length in mm. Default is not to use wh2o.</td>
</tr>
<tr>
<td>allowed:</td>
<td>double</td>
</tr>
<tr>
<td>Default:</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Returns**

Quantity

**Example**

wh2o = qa.quantity(0.4,'mm')
print "(INPUT CHANGE) water vapor column:" , wh2o['value'], wh2o['unit']
print "(NEW OUTPUT) T_EBB =", at.getAverageTebbSky(0,wh2o)['value'][0], at.getAverageTebbSky
atmosphere.getTebbSky.html

**atmosphere.getTebbSky - Function**

4.1.1 Returns equiv. BB Temp

**Description**

Gets the Equivalent Blackbody Temperature in spectral window spwid and channel nc, for Water Vapor Column wh2o, the current Air Mass, and perfect Sky Coupling to the sky.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Channel number (0-based) - defaults to reference channel</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td>spwid</td>
<td>Spectral window (0-based). The value must be &gt;= 0.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
<tr>
<td>wh2o</td>
<td>User specified water column length in mm. Default is not to use wh2o.</td>
</tr>
<tr>
<td></td>
<td>allowed: doublemm</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
</tbody>
</table>

**Returns**

Quantity

**Example**

```python
for s in range(at.getNumSpectralWindows()):
    for i in range(at.getNumChan(s)):
        print "Band", s, " channel ", i, "TebbSky = ", at.getTebbSky(i,s)[value][0]
```
atmosphere.getTebbSkySpec - Function

4.1.1 Returns equiv. BB Temp on each channel of a band

Description

Gets the Equivalent Blackbody Temperatures in a spectral window spwid for Water Vapor Column wh2o, the current Air Mass, and perfect Sky Coupling to the sky.

Arguments

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Inputs</th>
<th>Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>tebbSky K</td>
<td>spwid</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>wh2o</td>
<td></td>
</tr>
</tbody>
</table>

Outputs:
- **tebbSky K**: the Equivalent Blackbody Temperatures in a band - Quantum with a vector value and unit of K
  - allowed: doubleK
  - Default:

Inputs:
- **spwid**: Spectral window (0-based). The value must be >= 0.
  - allowed: int
  - Default: 0
- **wh2o**: User specified water column length in mm. Default is not to use wh2o.
  - allowed: double
  - Default: -1

Returns:
- int

Example

```python
sw=at.getWetOpacitySpec()
# returns a tuple of
# 0 - The number of channels, and
# 1 - the Equivalent Blackbody Temperatures in a band
sw[1]["value"]
```
Another example:
for s in range(at.getNumSpectralWindows()):
    print "band", s
    tebbspec = at.getTebbSkySpec(swid=s)
    for i in range(at.getNumChan(s)):
        print " - TebbSky %f [%s] " % (tebbspec[1]['value'][i], tebbspec[1]['unit'])
atmosphere.getAverageTrjSky.html

atmosphere.getAverageTrjSky - Function

4.1.1 Returns the average Rayleigh-Jeans Temperature

Description

Returns the average Rayleigh-Jeans Temperature in spectral window spwid, for the current conditions and a perfect sky coupling.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spwid</td>
<td>Spectral window (0-based). The value must be $\geq 0$.</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
<tr>
<td>wh2o</td>
<td>User specified water column length in mm. Default is not to use wh2o.</td>
</tr>
<tr>
<td>allowed:</td>
<td>doublemm</td>
</tr>
<tr>
<td>Default:</td>
<td>-1</td>
</tr>
</tbody>
</table>

Returns

Quantity

Example

```python
wh2o = qa.quantity(0.4, 'mm')
print "(INPUT CHANGE) water vapor column:", wh2o['value'], wh2o['unit']
print "(NEW OUTPUT) T_RJ =", at.getAverageTrjSky(0, wh2o)['value'][0], at.getAverageTrjSky(0, wh2o)['unit']
```

1870
atmosphere.getTrjSky.html

atmosphere.getTrjSky - Function

Returns the Rayleigh-Jeans Temperature

Description

Gets the Rayleigh-Jeans Temperature in spectral window spwid and channel
nc, for Water Vapor Column wh2o, the current Air Mass, and perfect Sky
Coupling to the sky.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Channel number (0-based) - defaults to reference channel</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
<tr>
<td>spwid</td>
<td>Spectral window (0-based). The value must be &gt;= 0.</td>
</tr>
<tr>
<td></td>
<td>allowed: int</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
<tr>
<td>wh2o mm</td>
<td>User specified water column length in mm. Default is not to use wh2o.</td>
</tr>
<tr>
<td></td>
<td>allowed: double mm</td>
</tr>
<tr>
<td></td>
<td>Default: -1</td>
</tr>
</tbody>
</table>

Returns

Quantity

Example

```python
for s in range(at.getNumSpectralWindows()):
    for i in range(at.getNumChan(s)):
        print "Band", s, " channel ", i, "TrjSky = ", at.getTrjSky(i,s)['value'][0],
```

1871
atmosphere.getTrjSkySpec.html

atmosphere.getTrjSkySpec - Function

4.1.1 Returns the Rayleigh-Jeans Temperatures on each channel of a band

Description

Gets the Rayleigh-Jeans Temperatures in a spectral window spwid for Water Vapor Column wh2o, the current Air Mass, and perfect Sky Coupling to the sky.

Arguments

<table>
<thead>
<tr>
<th>Outputs</th>
<th>K</th>
<th>the Rayleigh-Jeans Temperatures in a band - Quantum with a vector value and unit of K</th>
</tr>
</thead>
<tbody>
<tr>
<td>trjSky</td>
<td></td>
<td>allowed: doubleK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inputs</th>
<th>spwid</th>
<th>Spectral window (0-based). The value must be &gt;= 0.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>int</td>
<td>allowed:</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Default:</td>
</tr>
</tbody>
</table>

| wh2o    | mm    | User specified water column length in mm. Default is not to use wh2o.                   |
|         |       | allowed: doublemm                                                                      |
|         |       | Default: -1                                                                           |

Returns

int

Example

```python
sw=at.getWetOpacitySpec()
# returns a tuple of
# 0 - The number of channels, and
# 1 - the Equivalent Blackbody Temperatures in a band
sw[1]['value']
```

1873
Another example:
for s in range(at.getNumSpectralWindows()):
    print "band", s
    trjspec = at.getTrjSkySpec(swid=s)
    for i in range(at.getNumChan(s)):
        print "  - TrjSky %f [%s] " % (trjspec[1]['value'][i],trjspec[1]['unit'])
Chapter 5

Package SingleDish

Single-dish data analysis package

5.1 sd - Module

Single-dish data analysis package

Description The sd module provides functions and tools to reduce and analyse single-dish data. The module can be categorised as

- sd - Top level tool that contains the following sub tools and general functions for single-dish data reduction and analysis such as calibration, data averaging, etc.
- sd.scantable - Representation of single-dish data format. The tool also contains getter/setter functions as well as functions that manipulates data contents such as baseline fitting, flagging, etc.
- sd.selector - Data selector that operates on scantable.
- sd.fitter - This tool is an engine to perform both baseline and spectral line fitting.
- sd.linefinder - This tool performs automatic line finding according to the user specified thresholds.
- sd.linecatalog - Line catalog that contains name of species, line frequency with error, and line intensity. It accepts both pre-defined line catalog (JPL) for this tool and an user provided catalog in ASCII or specific table format.
- sd.plotter - Plotter tool dedicated for single-dish data
• `sd.coordinate` - Tool for conversion from pixel to velocity or frequency.
• `sd.opacity_model` - MIRIAD like opacity model.
• `sd.asapgrid` - Tool to convolve map data onto regularly spaced grid.
• `sd.asaplog` - Wrapper object to allow for both casapy and asap logging.
5.1.1 sd - Tool

Single-dish data analysis package
Requires: Synopsis

Description

The sd tool provides data reduction functions for single-dish (auto-correlation) data. It is actually a standalone software that is called ASAP (ATNF Spectral Analysis Package). For more information about ASAP, visit [here](#).

The sd tool is a top level object and it contains various sub tools as well as a number of functions.

Important note

In the latest release, single dish package is loaded at the start-up of CASA. Therefore, it is not necessary to load ASAP explicitly anymore.

Data format

There are various types of data format for single-dish data. Currently, sd tool supports the following data formats for reading and writing.

- reading: Scantable, Measurement Set, SDFITS (GBT, ATNF), NRO
- writing: Scantable, Measurement Set, SDFITS (ATNF), image FITS, CLASS, ASCII

An internal data format for sd tool is Scantable. The Scantable is defined as CASA table and is specially designed for single-dish data. The data is converted to Scantable during the processing whatever its original data format is. The result of the processing can be stored in other data format than Scantable. The Scantable is implemented as `sd.scantable` object in sd tool. See `sd.scantable` for details.

Selecting data

The Scantable can contain data with various types of frequency setting, multiple polarization components, and so on. In case you want to select data by some conditions, `sd.selector` is available for that. The `sd.selector` provides various functions to select and/or to sort data as well as it provides an interface to select data by TaQL. See `sd.selector` for details.

Fitting

The `sd.fitter` is an object to perform fitting data. Both baseline and line fitting for spectral data can be done by this object if appropriate fitting function and masked region are set. Currently, Gaussian and Lorentzian are supported for line fitting, while polynomials with arbitrary order are available for baseline fitting.

1877
fitting. Masking should be applied in the form of Bool array. See sd.fitter for details.

**Line finding**
The sd tool provides automatic line finding functionality that is called sd.linefinder. The sd.linefinder has a few control parameter for line finding. The user is able to customize a behavior of line finding process by changing these parameters. Simplified line finding tool sd.simplelinefinder is also available. See sd.linefinder and sd.simplelinefinder for details.

**Line catalog**
The sd.linecatalog object is an interface for line catalog. The input data should be provided from the user. Supported format is ASCII table or specific format for sd.linecatalog. CASA package contains a default line catalog that is a part of JPL line catalog and is specific table format for the sd tool (ASAP). Benefit of this object is that it provides an interface to the catalog such as selecting lines by frequency and/or line intensity. It also enables an interaction between spectral data on the sd.plotter such as overlaying line catalog on the plotted spectra. See sd.linecatalog for details.

**Plotter**
The sd.plotter is a plotter object that is exclusively designed for sd tool. That provides plotting functionalities for spectral data, time variation of azimuth and elevation, and pointing information. For plotting spectral data, it supports multi-panel and multi-page plot. See sd.plotter for details.

**Coordinate system**
The sd.coordinate is a representation of spectral coordinate of the data (frequency axis). It provides functions for conversion between pixel (channel), frequency, and velocity. See sd.coordinate for details.

**Opacity model**
In the sd tool, MIRIAD like atmospheric opacity model is implemented as sd.opacity. It calculates opacities from given atmospheric conditions (temperature, pressure, and humidity) and elevation. See sd.opacity for details.

**Gridding**
The tool to convolve spectral data onto regularly spaced grid, sd.asapgrid is available. Currently, three convolution kernels, box, prolate-spheroidal, and gaussian, can be used for convolution. See sd.asapgrid for details.

**Logging**
Although logging system is integrated into CASA logger, sd tool has own logging functions so that you can use it instead of CASA logging functions. Log messages will be displayed in the CASA logger either you use logging functions for CASA or sd tool. See sd.asaplog for details.

Methods

-almacal- Calibration function specific for ALMA data
apexcal Calibration function specific for APEX data
average_time Averaging data in time
calls Calibration function for frequency switched data
calibrate High level function for calibration
calnod Calibration function for nodding data
calps Calibration function for position switched data
calnod Calibration function for frequency switched data
calps
commands Show a list of commands and their short descriptions
dosigref Equivalent function with dosigref in GBTIDL
dototalpower Equivalent function with dototalpower in GBTIDL
fitter Create a fitter object
get_revision Get revision of the source code for the tool
is_asap cli Return True if the tool is launched as standalone software
is_casapy Return True if the tool is loaded as part of CASA
linecatalog Create a linecatalog object
linefinder
list_files Return a list of files readable by sd tool
list_rcparameters Print a list of rc parameters
list_scans Return a list of scantables created by the user
mask_and Logical operation function on array
mask_or Logical operation function on array
mask_not Logical operation function on array
merge Merge a list of scantables into one
opacity_model Create an opacity_model object
quotient Take a quotient of a signal and reference scan
rc Set the current rc parameters
scantable Create scantable object
selector Create selector object
skydip Determine the opacity from a set of 'skydip' observations
splitant Split Measurement Set data by antenna and save them as scantable
unique Return the unique values in a list
welcome Return a welcome message
sd.almacal - Function

5.1.1 Calibration function specific for ALMA data

Description

This method is properly defined for calibration of ALMA data. Input data must be a scantable object. The calibration scheme is,

\[ T_a^* = T_{sy} \cdot \frac{ON - OFF}{OFF}, \]

where \( T_a^* \) is an antenna temperature, \( T_{sy} \) is a system temperature, \( ON \) and \( OFF \) are raw (uncalibrated) spectral data that correspond to on-source and off-source position, respectively. The \( OFF \) scan is linearly interpolated in time if it exists in the vicinity of target \( ON \) scan. The calmode argument specifies the calibration mode. Supported calibration modes are 'ps' (position switch) and 'fs' (frequency switch). The 'ps' includes calibration for nutator switching as well as classical position switching. The 'ps' mode also supports calibration of OTF position raster scanned data that consists of on-source scans with simple scan pattern and explicit off-source scans.

Arguments

Inputs

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>scantab</td>
<td>input data as a scantable</td>
<td>scantable</td>
<td></td>
</tr>
<tr>
<td>scannos</td>
<td>a list of scan numbers to be calibrated</td>
<td>(list of) integer, string</td>
<td>[] (all data)</td>
</tr>
<tr>
<td>calmode</td>
<td>Calibration mode</td>
<td>string ('ps', 'fs', 'none')</td>
<td>'none'</td>
</tr>
<tr>
<td>verify</td>
<td>Verify calibration if True</td>
<td>bool</td>
<td>False</td>
</tr>
</tbody>
</table>

Returns

scantable

Example

1880
s=sd.scantable('alma-scans.asap',average=False,getpt=True)
scal=sd.almacal(s,calmode='ps')
sd.apexcal.html

**sd.apexcal - Function**

5.1.1 Calibration function specific for APEX data

**Description**

This method is properly defined for calibration of APEX data. Input data must be a scantable object. The calibration scheme is essentially same as ALMA data,

\[
T_a^* = T_{\text{sys}} \frac{ON - OFF}{OFF},
\]

where \(T_a^*\) is an antenna temperature, \(T_{\text{sys}}\) is a system temperature, \(ON\) and \(OFF\) are raw (uncalibrated) spectral data that correspond to on-source and off-source position, respectively. Only difference with ALMA calibration is that \(T_{\text{sys}}\) doesn’t provided. Instead, it is computed from a calibration temperature and two calibration scans: a blank sky and a load with known temperature. The \(OFF\) scan is linearly interpolated in time if it exists in the vicinity of target \(ON\) scan. The calmode argument specifies calibration mode. Supported calibration modes are ‘ps’ (position switch) and ‘fs’ (frequency switch). The ‘ps’ includes calibration for nutator switching as well as classical position switching.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>scantab</td>
<td>input data as a scantable</td>
<td>scantable</td>
<td></td>
</tr>
<tr>
<td>scannos</td>
<td>a list of scan numbers to be calibrated</td>
<td>(list of) integer, string</td>
<td></td>
</tr>
<tr>
<td>calmode</td>
<td>Calibration mode</td>
<td>string ('ps', 'fs', 'none')</td>
<td></td>
</tr>
<tr>
<td>verify</td>
<td>Verify calibration if True</td>
<td>bool</td>
<td>False</td>
</tr>
</tbody>
</table>

**Returns**

scantable
Example

```python
s = sd.scantable('alma-scans.asap', average=False, getpt=True)
scal = sd.apexcal(s, calmode='ps')
```
sd.average_time.html

sd.average_time - Function

5.1.1 Averaging data in Time

Description

The function computes a weighted time average of a scantable or a list of scantables. The averaging is done in channel only. Supported weighting schemes are as follows:

- none — no weighting
- var — 1/var(spectrum) weighted
- tsys — 1/Tsys**2 weighted
- tint — integration time weighted
- tintsys — Tint/Tsys**2 weighted
- median — median averaging

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input data</td>
<td>Input data as a scantable</td>
</tr>
<tr>
<td>allowed:</td>
<td>scantable or a list of scantables</td>
</tr>
<tr>
<td>Default:</td>
<td>scanable or a list of scantables</td>
</tr>
<tr>
<td>mask</td>
<td>An optional mask (only used for ‘var’ and ‘tsys’ weighting)</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool list</td>
</tr>
<tr>
<td>Default:</td>
<td>none</td>
</tr>
<tr>
<td>scanav</td>
<td>True averages each scan separately, False averages all scans together</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>False</td>
</tr>
<tr>
<td>weight</td>
<td>Weighting scheme (see description)</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>‘tint’</td>
</tr>
<tr>
<td>align</td>
<td>Align the spectra in velocity before averaging. It takes the time of the</td>
</tr>
<tr>
<td></td>
<td>first spectrum in the first scantable as reference time.</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>False</td>
</tr>
</tbody>
</table>

1884
Returns
scantable

Example

```python
scana = sd.scantable('scana.asap')
scnb = sd.scantable('scnb.asap')
# return a time averaged scan from scana and scnb
# without using a mask
scanav = sd.average_time(scana, scnb)
# or equivalent
scanav = sd.average_time([scana, scnb])
# return the (time) averaged scan, i.e. the average of
# all correlator cycles
scanav = sd.average_time(scan, scanav=True)
```
### sd.calfs - Function

#### 5.1.1 Calibration function for frequency switched data

**Description**

Calibrate GBT frequency switched data. Adopted from GBTIDL getfs. Currently calfs identify the scans as frequency switched data if source type enum is fson and fsoff. The data must contain 'CAL' signal on/off in each integration. To identify 'CAL' on state, the source type enum of foncal and foffcal need to be present.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>scantab</td>
<td>Input data as a scantable</td>
<td>scantable</td>
<td></td>
</tr>
<tr>
<td>scannos</td>
<td>A list of scan numbers to be calibrated</td>
<td>(list of) integer</td>
<td>[] (all data)</td>
</tr>
<tr>
<td>smooth</td>
<td>Optional box smoothing order for the reference</td>
<td>int</td>
<td>1 (no smoothing)</td>
</tr>
<tr>
<td>tsysval</td>
<td>Optional user specified Tsys</td>
<td>float</td>
<td>0.0 (use Tsys in the data)</td>
</tr>
<tr>
<td>tauval</td>
<td>Optional user specified optical depth</td>
<td>float</td>
<td>0.0</td>
</tr>
<tr>
<td>verify</td>
<td>Verify calibration if True</td>
<td>bool</td>
<td>False</td>
</tr>
</tbody>
</table>

**Returns**

scantable

**Example**

```plaintext
1886
```
s=sd.scantable('FLS3a_HI.asap')  # load in the saved ASAP dataset with FLS3a
scanns = s.getscannos()        # get a list of the scan numbers in the scantable
sn=list(scanns)               # Do a frequency switched calibration on
res=sd.calfs(s,sn)
sd.calibrate.html

**sd.calibrate - Function**

5.1.1 High level function for calibration that calls appropriate calibration function depending on the data

**Description**

This method is a high level function for calibration. It calls appropriate calibration function depending on the origin of the data. It checks an calmode argument first, and then looks antenna name in the data. Calibration functions that can be called from the method are:

- **calmod** if calmode is ’nod’
- **auto.quotient** if calmode is ’quotient’
- **calps** if calmode is ’ps’ and antenna name is ’GBT’
- **calfs** if calmode is ’fs’ and antenna name is ’GBT’
- **apexcal** if calmode is ’ps’ or ’fs’ and antenna name contains ’APEX’
- **almacal** if calmode is ’ps’, ’fs’, or ’otf’ and antenna name contains ’ALMA’ or ’OSF’

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scantab</td>
<td>input data as a scantable</td>
</tr>
<tr>
<td></td>
<td>allowed: scantable</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>scannos</td>
<td>a list of scan numbers to be calibrated</td>
</tr>
<tr>
<td></td>
<td>allowed: (list of) integer, string</td>
</tr>
<tr>
<td></td>
<td>Default: [ ] (all data)</td>
</tr>
<tr>
<td>calmode</td>
<td>Calibration mode</td>
</tr>
<tr>
<td></td>
<td>allowed: string (’ps’, ’fs’, ’none’)</td>
</tr>
<tr>
<td></td>
<td>Default: ’none’</td>
</tr>
<tr>
<td>verify</td>
<td>Verify calibration if True</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: False</td>
</tr>
</tbody>
</table>

**Returns**

scantable
Example

```python
s = sd.scantable('alma-scans.asap', average=False, getpt=True)
scal = sd.calibrate(s, calmode='ps')
```
sd.calnod.html

\textbf{sd.calnod - Function}

5.1.1 Calibration function for nodding data

\textbf{Description}

This method performs full (but a pair of scans at time) processing of GBT Nod data calibration. Adopted from GBTIDL’s getnod

\textbf{Arguments}

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scantab</td>
<td>Input data as a scantable</td>
</tr>
<tr>
<td>allowed:</td>
<td>scantable</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>scannos</td>
<td>A pair of scan numbers, or the first scan number of the pair</td>
</tr>
<tr>
<td>allowed:</td>
<td>(list of) integer</td>
</tr>
<tr>
<td>Default:</td>
<td>[] (all data)</td>
</tr>
<tr>
<td>smooth</td>
<td>Box car smoothing order</td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>1 (no smoothing)</td>
</tr>
<tr>
<td>tsysval</td>
<td>Optional user specified Tsys</td>
</tr>
<tr>
<td>allowed:</td>
<td>float</td>
</tr>
<tr>
<td>Default:</td>
<td>0.0 (use Tsys in the data)</td>
</tr>
<tr>
<td>tauval</td>
<td>Optional user specified optical depth (not implemented yet)</td>
</tr>
<tr>
<td>allowed:</td>
<td>float</td>
</tr>
<tr>
<td>Default:</td>
<td>0.0</td>
</tr>
<tr>
<td>tcalval</td>
<td>Optional user specified Tcal</td>
</tr>
<tr>
<td>allowed:</td>
<td>float</td>
</tr>
<tr>
<td>Default:</td>
<td>0.0</td>
</tr>
<tr>
<td>verify</td>
<td>Verify calibration if True</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>False</td>
</tr>
</tbody>
</table>

\textbf{Returns}

scantable

\textbf{Example}

1890
s=sd.scantable('IRC+10216_rawACSmod',False)#load the data without averaging
scal=sd.calnod(s,[229,230])  # Calibrate CS scans
sd.calps - Function

5.1.1 Calibration function for position switched data

Description

The method calibrates GBT position switched data. Adopted from GBTIDL
getps Currently calps identify the scans as position switched data if source
type enum is pson or psoff. The data must contain 'CAL' signal on/off in
each integration. To identify 'CAL' on state, the source type enum of poncal
and poffcal need to be present.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>scantab</td>
<td>Input data as a scantable</td>
<td>allowed: scantable</td>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>scannos</td>
<td>A list of scan numbers to be calibrated</td>
<td>allowed: (list of) integer</td>
<td>Default: [] (all data)</td>
<td></td>
</tr>
<tr>
<td>smooth</td>
<td>Optional box smoothing order for the reference</td>
<td>allowed: int</td>
<td>Default: 1 (no smoothing)</td>
<td></td>
</tr>
<tr>
<td>tsysval</td>
<td>Optional user specified Tsys</td>
<td>allowed: float</td>
<td>Default: 0.0 (use Tsys in the data)</td>
<td></td>
</tr>
<tr>
<td>tauval</td>
<td>Optional user specified optical depth</td>
<td>allowed: float</td>
<td>Default: 0.0</td>
<td></td>
</tr>
<tr>
<td>tcalval</td>
<td>Optional user specified Tcal</td>
<td>allowed: float</td>
<td>Default: 0.0 (use Tcal in the data)</td>
<td></td>
</tr>
<tr>
<td>verify</td>
<td>Verify calibration if True</td>
<td>allowed: bool</td>
<td>Default: False</td>
<td></td>
</tr>
</tbody>
</table>

Returns

scantable

Example

1892
s=sd.scantable('OrionS_rawACSmod',False)#load the data without averaging
scal=sd.calps(s,[24,25,26,27]) # Calibrate SiO scans
sd.commands.html

**sd.commands - Function**

### 5.1.1 Show a list of commands and their short descriptions

**Description**

The method prints a list of commands and their short descriptions. The output is sorted by their intents and/or subtools that the commands associate.

**Arguments**

**Returns**

**Example**

```python
ds.commands()
# Output will be as follows
[The scan container]
  scantable - a container for integrations/scans
              (can open asap/rpfits/sdfits and ms files)
  copy - returns a copy of a scan
  get_scan - gets a specific scan out of a scantable
              (by name or number)
  drop_scan - drops a specific scan out of a scantable
              (by number)
  set_selection - set a new subselection of the data
  get_selection - get the current selection object
  summary - print info about the scantable contents
  stats - get specified statistic of the spectra in
          the scantable
  stddev - get the standard deviation of the spectra
           in the scantable
  get_tsys - get the TSys
  get_time - get the timestamps of the integrations
  get_inttime - get the integration time
  get_sourcename - get the source names of the scans
  get_azimuth - get the azimuth of the scans
```

1894
get_elevation  - get the elevation of the scans
get_parangle  - get the parallactic angle of the scans
get_coordinate - get the spectral coordinate for the given row,
                which can be used for coordinate conversions
get_weather   - get the weather condition parameters
get_unit      - get the current unit
set_unit      - set the abcissa unit to be used from this
                point on

...
sd.dosigref.html

**sd.dosigref - Function**

5.1.1 Equivalent function with dosigref in GBTIDL

**Description**

The method calculates a quotient \((\text{sig-ref/ref} \times \text{Tsys})\). Adopted from GBTIDL dosigref.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>sig</td>
<td>On-source data as a scantable</td>
<td>scantable</td>
<td></td>
</tr>
<tr>
<td>ref</td>
<td>Reference data as a scantable</td>
<td>scantable</td>
<td></td>
</tr>
<tr>
<td>smooth</td>
<td>Width of box car smoothing for reference</td>
<td>int</td>
<td>1 (no smoothing)</td>
</tr>
<tr>
<td>tsysval</td>
<td>User specified Tsys</td>
<td>float</td>
<td>0.0 (use Tsys in the data)</td>
</tr>
<tr>
<td>tauval</td>
<td>User specified optical depth (required if tsysval is set)</td>
<td>float</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Returns**

scantable

**Example**

```python
s = sd.scantable('OrionS_rawACSmod',average=False)
sel = sd.selector()
# calibration scans
sel.set_types([srctype.poncal,srctype.poffcal])
s.set_selection(sel)
ssubon=s.copy()
s.set_selection()
```

1896
sel.reset()
# off-calibration scans
sel.set_types([srctype.pson,srctype.psoff])
s.set_selection(sel)
ssuboff=s.copy()
s.set_selection()
sel.reset()
# calibration
cals = sd.dototalpower(ssubon,ssuboff)
# ON scan
sel.set_types(srctype.pson)
cals.set_selection(sel)
sig = cals.copy()
cals.set_selection()
sel.reset()
# OFF scan
sel.set_types(srctype.psoff)
cals.set_selection(sel)
ref = cals.copy()
cals.set_selection()
sel.reset()
# get calibrated data
ress = sd.dosigref(sig,ref,smooth=1)
**sd.dototalpower - Function**

5.1.1 Equivalent function with dototalpower in GBTIDL

**Description**

The method performs calibration for CAL on,off signals. Adopted from GBTIDL dototalpower.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calon</td>
<td>The 'cal on' subintegration as a scantable</td>
</tr>
<tr>
<td>caloff</td>
<td>The 'cal off' subintegration as a scantable</td>
</tr>
<tr>
<td>tcalval</td>
<td>User supplied Tsys</td>
</tr>
</tbody>
</table>

- **calon**: allowed: scantable, Default: 
- **caloff**: allowed: scantable, Default: 
- **tcalval**: allowed: float, Default: 0.0 (use Tcal in the data)

**Returns**

scantable

**Example**

```python
s = sd.scantable('OrionS_rawACSmod',average=False)
sel = sd.selector() # calibration scans
sel.set_types([srctype.poncal,srctype.poffcal])
s.set_selection(sel)
ssubon=s.copy()
s.set_selection()
sel.reset() # off-calibration scans
sel.set_types([srctype.pson,srctype.psoff])
s.set_selection(sel)
ssuboff=s.copy()
s.set_selection()
```

1898
sel.reset()
# calibration
cals = sd.dototalpower(ssubon,ssuboff)
# ON scan
sel.set_types(srctype.pson)
cals.set_selection(sel)
sig = cals.copy()
cals.set_selection()
    sel.reset()
# OFF scan
sel.set_types(srctype.psoff)
cals.set_selection(sel)
ref = cals.copy()
cals.set_selection()
    sel.reset()
# get calibrated data
ress = sd.dosigref(sig,ref,smooth=1)
sd.get_revision.html

**sd.get_revision - Function**

5.1.1 Get revision of the source code for the tool

**Description**

Get the revision of the software. Actually it returns a revision number of the source code that is managed by subversion.

**Arguments**

**Returns**

string

**Example**

```python
rev=sd.get_revision()
print rev
'13018'
```
sd.is_asap_cli.html

sd.is_asap_cli - Function

5.1.1 Check if the tool is loaded as part of CASA or is launched as standalone software

Description

The method checks if sd tool (ASAP) is running standalone. This always returns False if you use CASA.

Arguments

Returns

bool (False)

Example

isasap=sd.is_asap_cli()
print isasap
False
sd.is_casapy.html

**sd.is_casapy - Function**

Check if the tool is loaded as part of CASA or is launched as standalone software

**Description**

The method checks if sd tool is running on CASA. This always returns True if you use CASA.

**Arguments**

**Returns**

bool (True)

**Example**

```python
iscasa=sd.is_casapy()
print iscasa
True
```
**sd.list_files**

**sd.list_files** - **Function**

5.1.1 Return list of files readable by sd tool

**Description**

Return a list files readable by asap, such as MS, rpf, sdfits, mbf, asap. The method looks a directory that is indicated by path argument, and searches files with extension specified by suffix. Allowed extensions are:

- rpf — RPFITS (default)
- rpf.1 — RPFITS
- rpf.2 — RPFITS
- sdf — SDFITS
- sdfits — SDFITS
- mbf — MBFITS
- asap — Scantable
- ms — Measurement Set

Note that the method just checks the extension of the file name. Thus, the returned list doesn’t contains files that don’t have the above extensions even if they are readable.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The directory to list</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: './' (current directory)</td>
</tr>
<tr>
<td>suffix</td>
<td>The file extension</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: 'rpf'</td>
</tr>
</tbody>
</table>

**Returns**

string array
Example

```python
files = sd.list_files("data/", "sdfits")
print files
['data/2001-09-01_0332_P363.sdfits',
'data/2003-04-04_131152_t0002.sdfits',
'data/Sgr_86p262_best_SPC.sdfits']
```
sd.list_rcreadparams.html

**sd.list_rcreadparams - Function**

5.1.1 Print a list of rc parameters

**Description**

Print a list of rc parameters and its default values that determine basic behavior of the tool. The user can be accessed rc parameters directory since the rc parameters are available as a Python dictionary sd.rcParams. There is also a method to set rc parameters called [rc](#). Contents of the rc parameters are described below.

- **verbose**
  - It will disable exceptions and just print the messages (only valid in standalone mode)

- **useplotter**
  - Preload a default plotter

- **insitu**
  - Apply operations on the input scantable or return new one

- **plotter.gui**
  - Do we want a GUI or plot to a file

- **plotter.stacking**
  - Default mode for color stacking

- **plotter.panelling**
  - Default mode for panelling

- **plotter.ganged**
  - Push panels together, to share axis labels

- **plotter.decimate**
  - Decimate the number of points plotted by a factor of nchan/1024

- **plotter.colours**
  - Default color

- **plotter.linestyles**
  - Default linestyles

- **plotter.histogram**
  - Enable/disable histogram plotting

- **plotter.papertype**
  - Postscript paper type

1905
• plotter.axesformatting
  The formatting style of the x-axis

• scantable.storage
  Default storage of scantable ('memory'/disk')

• scantable.history
  Write history of each call to scantable

• scantable.save
  Default output format when saving

• scantable.autoaverage
  Auto averaging on read

• scantable.freqframe
  Default frequency frame to set when function `set_freqframe` is called

• scantable.verbosesummary
  Control the level of information printed by `summary`

• scantable.reference
  Control the identification of reference (off) scans (has to be regular expression)

• scantable.parallactify
  Indicate whether the data was parallactified (total phase offset is 0.0)

Arguments

Returns
string

Example

sd.list_rcparameters()
# output will be as follows
# general
# only valid in asap standard mode not in scripts or casapy
# It will disable exceptions and just print the messages
verbose : True

# preload a default plotter
useplotter : True

1906
# apply operations on the input scantable or return new one
insitu : True

# plotting

# do we want a GUI or plot to a file
plotter.gui : True

# default mode for colour stacking
plotter.stacking : Pol

# default mode for panelling
plotter.panelling : scan

# push panels together, to share axis labels
plotter.ganged : True

...
5.1.1 Return a list of scantables created by the user

Description

The method prints and returns a list of scantables that the user created.

Arguments

Returns

Example

```python
# no scantable is created yet
scanlist=sd.list_scans()
print scanlist
[]
# create scantable
s=sd.scantable('OrionS_rawACSmod',average=False)
# run list_scans() again
scanlist=sd.list_scans()
print scanlist
['s']
```
sd.mask_and.html

(sd.mask_and - Function)

5.1.1 Logical operation function on array

Description

This is an utility function that performs logical 'and' operation on specified two boolean arrays in element-by-element manner. Input arrays should have same length.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Input boolean array</td>
<td>allowed: bool array</td>
</tr>
<tr>
<td>b</td>
<td>Input boolean array</td>
<td>allowed: bool array</td>
</tr>
</tbody>
</table>

Returns

bool array

Example

```python
a=[True,False,False]
b=[True,True,False]
sd.mask_and(a,b)
[True,False,False]
```
sd.mask_or.html

sd.mask_or - Function

5.1.1 Logical operation function on array

Description

This is an utility function that performs logical 'or' operation on specified two boolean arrays in element-by-element manner. Input arrays should have same length.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Input boolean array</td>
</tr>
<tr>
<td></td>
<td>allowed: bool array</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>b</td>
<td>Input boolean array</td>
</tr>
<tr>
<td></td>
<td>allowed: bool array</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

bool array

Example

a=[True,False,False]
b=[True,True,False]
sd.mask_or(a,b)
[True,True,False]
**sd.mask_not.html**

**sd.mask_not - Function**

**5.1.1** Logical operation function on array

**Description**

This is an utility function that performs logical 'not' operation on the input boolean array in element-by-element manner.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Input boolean array</td>
<td>bool array</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool array

**Example**

```python
a=[True,False,False]
sd.mask_not(a)
[False,True,True]
```
sd.merge - Function

5.1.1 Merge a list of scantables into one

Description

Merge a list of scantables, or comma-sperated scantables into one scantable.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Input scantables</td>
<td>list of scantables or comma-separated scantables</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

Returns

scantable

Example

scan1 = sd.scantable('scan1.asap')
scan2 = sd.scantable('scan2.asap')
myscans = [scan1, scan2]
allscans = sd.merge(myscans)
# or equivalent
sameallscans = sd.merge(scan1, scan2)
sd.quotient.html

**sd.quotient - Function**

5.1.1 Take a quotient of a signal and reference scan

**Description**

Return the quotient of a 'source' (signal) scan and a 'reference' scan. The reference can have just one scan, even if the signal has many. Otherwise they must have the same number of scans. The cursor of the output scan is set to 0. The preserve argument controls if continuum is preserved or not. The equation used in the method depends on its value. If preserve is True, the equation is,

\[ T_a^* = T_{\text{sys}}^{\text{OFF}} \frac{ON}{OFF} - T_{\text{sys}}^{\text{OFF}}, \]

while if preserve is False,

\[ T_a^* = T_{\text{sys}}^{\text{OFF}} \frac{ON}{OFF} - T_{\text{sys}}^{\text{ON}}, \]

where \( T_a^* \) is antenna temperature, \( ON \) and \( OFF \) are raw (uncalibrated) spectral data that correspond to on-source and off-source position, \( T_{\text{sys}}^{ON} \) and \( T_{\text{sys}}^{OFF} \) are system temperatures of \( ON \) and \( OFF \) scans, respectively.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>source</th>
<th>The on-source scan as a scantable</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>scantable</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reference</td>
<td>The reference scan as a scantable</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>scantable</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>preserve</td>
<td>Preserve the continuum or remove it</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>True</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

scantable

**Example**

1913
s = sd.scantable('OrionS_rawACSmod', average=False)
sel = sd.selector()
# ON scan
sel.set_types(srctype.pson)
s.set_selection(sel)
sig = s.copy()
s.set_selection()
 sel.reset()
# OFF scan
sel.set_types(srctype.psoff)
s.set_selection(sel)
ref = s.copy()
s.set_selection()
 sel.reset()
# get quotient data
cals = sd.quotient(source=sig, reference=ref, preserve=True)
sd.rc.html

sd.rc - Function

5.1.1 Set the current rc parameters

Description

Set the current rc parameters (sd.rcParams, see `list_rcparameters`). The group
is the grouping for the rc, eg for scantable.save the group is 'scantable', for
plotter.stacking, the group is 'plotter', and so on. kwargs is a list of attribute
name/value pairs, eg

```
sd.rc('scantable', save='SDFITS')
```

sets the current rc params and is equivalent to

```
sd.rcParams['scantable.save'] = 'SDFITS'
```

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Grouping for the rc</th>
</tr>
</thead>
<tbody>
<tr>
<td>group</td>
<td>String (&quot;','scantable','plotter&quot;)</td>
</tr>
<tr>
<td>allowed</td>
<td>A list of attribute name/value pairs</td>
</tr>
<tr>
<td>Default</td>
<td>Comma separated list of name/value pairs</td>
</tr>
</tbody>
</table>

Returns

Example

```
# set scantable.save as 'SDFITS' and scantable.storage as 'disk'
sd.rc('scantable',save='SDFITS', storage='disk')
# set insitu as False
sd.rc('',insitu=False)
```
sd.skydip.html

sd.skydip - Function

5.1.1 Determine the opacity from a set of 'skydip' observations

Description

Determine the opacity from a set of 'skydip' observations. This can be any set of observations over a range of elevations, but will usually be a dedicated (set of) scan(s). Return a list of 'n' opacities for 'n' IFs. In case of averagepol being 'False' a list of 'n*m' elements where 'm' is the number of polarisations, e.g. nIF = 3, nPol = 2 ⇒ [if0pol0, if0pol1, if1pol0, if1pol1, if2pol0, if2pol1]

The opacity is determined by fitting a first order polynomial to:

\[ T_{sys}(\text{airmass}) = p_0 + \text{airmass} \times p_1, \]

where

\[ \text{airmass} = \frac{1}{\sin(\text{elevation})} \]

\[ \tau = \frac{p_1}{T_{sky}} \]

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>File name or scantable or list of them</td>
<td>string, string array, scantable, list of scantables</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>averagepol</td>
<td>Return opacity values per polarization or average of opacities for the polarizations</td>
<td>bool</td>
<td>True</td>
</tr>
<tr>
<td>tsky</td>
<td>The sky temperature</td>
<td>float</td>
<td>300.0</td>
</tr>
<tr>
<td>plot</td>
<td>Plot each fit (airmass versus Tsys)</td>
<td>bool</td>
<td>False</td>
</tr>
</tbody>
</table>

Returns

float array
Example

```python
s = sd.scantable('skydip.asap', average=False)
sd.skydip(data=s, averagepol=True, plot=True)
```
sd.splitant.html

sd.splitant - Function

5.1.1 Split Measurement Set data by antenna and save them as scantable

Description

Split Measurement Set by antenna name, save data as a scantables, and return a list of filename. Notice this method can only be available from CASA. The outprefix argument specifies a prefix of output scantable name. The names of output scantables will be 'outprefix.antenna1.asap', 'outprefix.antenna2.asap', ... where antenna1 and antenna2 is antenna name that are stored in ANTENNA subtable of input Measurement Set.

Arguments

Inputs

<table>
<thead>
<tr>
<th>Arg</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>string</td>
<td>The name of Measurement Set to be read</td>
</tr>
<tr>
<td>outprefix</td>
<td>string</td>
<td>The prefix of output scantable name</td>
</tr>
<tr>
<td>overwrite</td>
<td>bool</td>
<td>If the file should be overwritten if it exists</td>
</tr>
</tbody>
</table>

Returns

string array

Example

```python
# assume that input MS contains data from 'DV01' and 'PM03'
outfiles=sd.splitant('osfdata.ms',outprefix='test')
print outfiles
[‘test.DV01.asap’,’test.PM03.asap’]
```
### sd.unique - Function

#### 5.1.1 Return the unique values in a list

**Description**

This is an utility function that returns the unique values in a list

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>The list to reduce</td>
<td>array</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

any array

**Example**

```python
x=[1,2,3,3,4]
y=sd.unique(x)
print y
[1,2,3,4]
```
sd.welcome.html

sd.welcome - Function

5.1.1 Return a welcome message

Description

Return a welcome message.
Note that the message assumes the sd tool (ASAP) is used as a standalone software so that some informations are not fit with CASA. For example, you should also report bugs on sd tool to CASA helpdesk, and you have to run 'sd.commands()' instead of 'commands()' if you want to get a list of commands from CASA console.

Arguments

Returns

string

Example

msg=sd.welcome()
print msg
# output will be as follows
Welcome to ASAP vtrunk (2010-08-13) - the ATNF Spectral Analysis Package

Please report any bugs via:
http://svn.atnf.csiro.au/trac/asap/simpleticket

[IMPORTANT: ASAP is 0-based]
Type commands() to get a list of all available ASAP commands.
5.1.2 sd.scantable - Tool

Representation of single-dish data format

Description

The scantable is a representation of the Scantable, which is a data format designed properly for single-dish data. The scantable is implemented as CASA table so that it is possible to access data via table tool. On the other hand, the scantable has its own functions to get/set some data. In addition to the simple setter/getter and utility functions, the scantable has some data reduction function such as calibration, baseline fitting, etc. The 'insitu' option often appears in the functions. This parameter controls if a certain operation is applied to this scantable (true) or return another scantable that is a result of the operation (false). In former case, the original scantable loaded will be lost although the data on disk will be kept. The default value of the option depends on sd.rcParams dictionary. See sd.list_rcparameters for details about sd.rcParams.

The constructor takes several arguments that handles some of optional behavior of it. The getpt and antenna arguments are only effective if file specified by filename argument is Measurement Set format.

Definition

A documentation for detailed definition of the Scantable will be prepared by ATNF.

Arguments
Inputs

filename
Name of an input file, or a reference to an existing scantable (advanced)
allowed: string, scantable
Default:

average
Average all integrations within a scan on read
allowed: bool
Default: None (taken from scantable.autoaverage attribute of rc parameters)

unit
Brightness unit. It must be consistent with K or Jy. It overrides the default value or replaces the value in existing scantables
allowed: string
Default: None

getpt
Measurement Set input data only. If True, all pointing data are filled.
allowed: bool
Default: False

antenna
Measurement Set input data only. Antenna selection by ID or name
allowed: string, integer
Default: " (first antenna in the ANTENNA table)

parallactify
Indicate that the data had been parallactified.
allowed: bool
Default: None (taken from scantable.parallactify attribute of rc parameters)

Example

# create scantable from the data from the second antenna in Measurement Set
s=sd.scantable('sddata.ms',average=False,getpt=True,antenna=1)

Methods

add
Return a scan where all spectra have the offset added

auto_c spline_baseline
Perform automatic line finding and baseline subtraction using cubic spline function

auto_poly_baseline
Perform automatic line finding and baseline subtraction using polynomial function

auto_sinusoid_baseline
Perform automatic line finding and baseline subtraction using sinusoidal function

auto_quotient
Automatic quotient for coordinated scans

average_beam
Average beams together for multi-beam observation

average_pol
Average polarizations together

average_time
Return time average of a scan

bin
Perform binning of spectra
getifnos  Get a list of IF numbers in the scantable
getmolnos  Get a list of molecule ids in the scantable
getpol    Get polarization number of the given row
getpolnos  Get a list of polarization numbers in the scantable
getscan   Get scan number of the given row
getscannos Get a list of scan numbers in the scantable
history   Print a history
inverted   Invert the phase of the complex polarization
lagflag   Perform Fourier filtering on the spectra
mx_quotient Form a quotient using "off" beams when observing in "MX" mode
nbeam    Return a number of beams
nchan    Return a number of channels
npeel    Return a number of cycles
nopoly    Return a number of polarizations
nrow    Return a number of rows
ncycle    Return a number of cycles
nif    Return a number of IFs
npoly    Return a number of polarizations
nscan    Return a number of scans
opacity   Apply an opacity correction
parallactify Set a flag to indicate the data should be treated as "parallactified"
poltype   Get a polarization type
polybaseline Perform a baseline subtraction using polynomial function
recalez Perform a baseline subtraction using sinusoidal function
resample  Recalculate azimuth and elevation for each sky position
resampleazel Perform a binning
rotateImpolphase Rotate a phase of the complex polarization
rotatexyphase Rotate a phase of the XY correlation
save   Store the scantable on disk
scale    Scale spectra by the given factor
setdirframe Set the frame type of the direction on the sky
setdoppler Set definition of the Doppler correction
setfeedtype Set the feed type
setfluxunit Set flux unit
setfreqframe Set the frame type of the spectral axis
setinstrument Set antenna name
setrestfreqs Set rest frequency
setselection Select a subset of the data
setsourcetype Set the types of source to be source or reference scan
setspectrum Set spectrum for specified row
setunit    Set unit for spectral axis
shiftrefpix Shift the reference pixel of the spectral coordinate
smooth    Smooth the spectra
stats     Compute specified statistics of the spectra
stddev    Compute standard deviation of the spectra
summary   Print a summary of the contents of the scantable
swaplinars Swap the linear polarizations XX and YY
sd.scantable.add.html

**sd.scantable.add - Function**

5.1.2 Return a scan where all spectra have the offset added

**Description**

Return a scan where all spectra have the offset added. If insitu is True, or insitu is None and sd.rcParams['insitu'] is True, the method will not return the result, but apply operation on this scantable.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset</td>
<td>The offset</td>
</tr>
<tr>
<td>allowed</td>
<td>float</td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>insitu</td>
<td>If False a new scantable is returned</td>
</tr>
<tr>
<td>allowed</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>None (use default value)</td>
</tr>
</tbody>
</table>

**Returns**

scantable

**Example**
5.1.2 Perform automatic line finding and baseline subtraction using cubic spline function

**Description**

Return a scan which has been baselined (all rows) by cubic spline function (piecewise cubic polynomial). Fit will be done with 'sigma-clipping'. Spectral lines are detected first using linefinder and masked out to avoid them affecting the baseline solution. The edge argument is an optional number of channel to drop at the edge of spectrum. If only one value is specified, the same number will be dropped from both sides of the spectrum. Default is to keep all channels. Nested tuples represent individual edge selection for different IFs (a number of spectral channels can be different). The threshold and chan_avg_limit arguments are linefinder options. The former is the threshold used by line finder. It is better to keep it large as only strong lines affect the baseline solution. The later is a maximum number of consecutive spectral channels to average during the search of weak and broad lines. The default is no averaging (and no search for weak lines). If such lines can affect the fitted baseline (e.g. a high order polynomial is fitted), increase this parameter (usually values up to 8 are reasonable). Most users of this method should find the default value sufficient. See linefinder for more details on these options. Note: The best-fit parameter values output in logger and/or blfile are now based on specunit of 'channel'.

**Arguments**
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Allowed Types</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>insitu</td>
<td>If False a new scantable is returned</td>
<td>bool</td>
<td>None (use default value)</td>
</tr>
<tr>
<td>mask</td>
<td>An optional mask retrieved from scantable</td>
<td>bool</td>
<td>None (no mask)</td>
</tr>
<tr>
<td>npiece</td>
<td>Number of pieces</td>
<td>integer</td>
<td>2</td>
</tr>
<tr>
<td>clipthresh</td>
<td>Clipping threshold in unit of sigma</td>
<td>float</td>
<td>3.0</td>
</tr>
<tr>
<td>clipniter</td>
<td>Maximum number of iteration of clipping</td>
<td>integer</td>
<td>0</td>
</tr>
<tr>
<td>edge</td>
<td>An optional number of channel to drop at the edge of spectrum</td>
<td>integer, integer array</td>
<td>(0,0)</td>
</tr>
<tr>
<td>threshold</td>
<td>The threshold used by line finder</td>
<td>float</td>
<td>3</td>
</tr>
<tr>
<td>chan_avg_limit</td>
<td>A maximum number of consecutive spectral channels to average during the search</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>plot</td>
<td>Plot the fit and the residual (currently unavailable)</td>
<td>boolean</td>
<td>False</td>
</tr>
<tr>
<td>getresidual</td>
<td>If False, return best-fit value instead of residual</td>
<td>boolean</td>
<td>True</td>
</tr>
<tr>
<td>showprogress</td>
<td>Show progress status for large data</td>
<td>boolean</td>
<td>True</td>
</tr>
<tr>
<td>minnrow</td>
<td>Minimum number of spectra to show progress status</td>
<td>integer</td>
<td>1000</td>
</tr>
<tr>
<td>outlog</td>
<td>Output the coefficients of the best-fit function to logger</td>
<td>boolean</td>
<td>False</td>
</tr>
<tr>
<td>blfile</td>
<td>Name of text file in which the best-fit parameter values to be written</td>
<td>string</td>
<td>&quot;</td>
</tr>
</tbody>
</table>
Returns
scantable

Example

```python
scan = sd.scantable('OrionS_rawACSmod_cal', average=False)
bscan = scan.auto_cspline_baseline(npiece=3, insitu=False)
```
sd.scantable.auto_poly_baseline.html

**sd.scantable.auto_poly_baseline** - Function

5.1.2 Perform automatic line finding and baseline subtraction using polynomial function

**Description**

Return a scan which has been baselined (all rows) by a polynomial. Spectral lines are detected first using linefinder and masked out to avoid them affecting the baseline solution.

The edge argument is an optional number of channel to drop at the edge of spectrum. If only one value is specified, the same number will be dropped from both sides of the spectrum. Default is to keep all channels. Nested tuples represent individual edge selection for different IFs (a number of spectral channels can be different).

The threshold and chan_avg_limit arguments are linefinder options. The former is the threshold used by line finder. It is better to keep it large as only strong lines affect the baseline solution. The later is a maximum number of consecutive spectral channels to average during the search of weak and broad lines. The default is no averaging (and no search for weak lines). If such lines can affect the fitted baseline (e.g. a high order polynomial is fitted), increase this parameter (usually values up to 8 are reasonable). Most users of this method should find the default value sufficient. See linefinder for more details on these options.

You can verify and decide whether you apply the fit result or not, if plot argument is True. In that case, you have to answer 'y' or 'n' for each spectra so that setting True is not recommended for large dataset.

Note: The best-fit parameter values output in logger and/or blfile are now based on specunit of 'channel'.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>An optional mask retrieved from scantable</td>
<td>bool array</td>
<td>None (no mask)</td>
</tr>
<tr>
<td>order</td>
<td>The order of the polynomial</td>
<td>integer</td>
<td>0</td>
</tr>
<tr>
<td>edge</td>
<td>An optional number of channel to drop at the edge of spectrum</td>
<td>integer, integer array</td>
<td>(0,0)</td>
</tr>
<tr>
<td>threshold</td>
<td>The threshold used by line finder</td>
<td>float</td>
<td>3</td>
</tr>
<tr>
<td>chan_avg_limit</td>
<td>A maximum number of consecutive spectral channels to average during the search</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>plot</td>
<td>Plot the fit and the residual.</td>
<td>bool</td>
<td>False</td>
</tr>
<tr>
<td>insitu</td>
<td>If False a new scantable is returned</td>
<td>bool</td>
<td>False</td>
</tr>
<tr>
<td>getresidual</td>
<td>If False, return best-fit value instead of residual</td>
<td>bool</td>
<td>True</td>
</tr>
<tr>
<td>showprogress</td>
<td>Show progress status for large data</td>
<td>bool</td>
<td>True</td>
</tr>
<tr>
<td>minnrow</td>
<td>Minimum number of spectra to show progress status</td>
<td>integer</td>
<td>1000</td>
</tr>
<tr>
<td>outlog</td>
<td>Output the coefficients of the best-fit function to logger</td>
<td>bool</td>
<td>False</td>
</tr>
<tr>
<td>blfile</td>
<td>Name of text file in which the best-fit parameter values to be written</td>
<td>string</td>
<td>&quot;</td>
</tr>
</tbody>
</table>

**Returns**

scantable
Example

```python
scan = sd.scantable('OrionS_rawACSmod_cal', average=False)
scan2 = scan.auto_poly_baseline(order=7, insitu=False)
```
sd.scantable.auto_quotient.html

**sd.scantable.auto_quotient - Function**

### 5.1.2 Automatic quotient for coordinated scans

**Description**

This function allows to build quotients automatically. It assumes the observation to have the same number of "ons" and "offs". The formula to get result depends on the preserve parameter. If it is True, the continuum will be preserved while if it is False, the continuum will be removed. The equation used are

\[
T^*_a = T^\text{OFF}_{\text{sys}} \frac{O}{O} - T^\text{OFF}_{\text{sys}},
\]

if preserve is True, while

\[
T^*_a = T^\text{OFF}_{\text{sys}} \frac{O}{O} - T^\text{ON}_{\text{sys}},
\]

if preserve is False.

The mode argument controls the on/off decision mode. If mode is 'paired' (default), it identifies 'off' scans by the trailing 'R' (Mopra/Parkes) or 'e'/ 'w' (Tid) and matches on/off pairs from the observing pattern. On the other hand, 'time' finds the closest off in time.

Note that the verify argument is not yet implemented.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>preserve</td>
<td>You can preserve the continuum or remove it</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>mode</td>
<td>The on/off decision mode</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>'paired'</td>
<td></td>
</tr>
<tr>
<td>verify</td>
<td>Verify result (not yet implemented)</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>False</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

scantable

**Example**

1932
5.1.2 Perform automatic line finding and baseline subtraction using sinusoidal function

**Description**

Return a scan which has been baselined (all rows) with sinusoidal functions. Fit will be done with ‘sigma-clipping’.

Spectral lines are detected first using linefinder and masked out to avoid them affecting the baseline solution.

If applyfft is set to True, the function performs Fourier analysis to select wave numbers for sinusoidal fitting. Currently, 'fft' is only available to be used for the analysis. You can specify threshold for selection of wave number using fftthresh parameter. Both float and string is acceptable. Given a float value, the unit is set to sigma. For string values, allowed formats include:

- any decimal number plus 'sigma' (e.g. '3sigma')
- 'top' plus any decimal number (e.g. 'top10')

In addition, you can add or reject specific wave numbers from the fit using addwn and rejwn, respectively. You can specify wave numbers as an integer, string, or list of them. For string specification, syntax for those parameters are as follows:

- 'a-b' (= a, a+1, a+2, ..., b-1, b)
- '<a' (= 0, 1, ..., a-2, a-1)
- '>a' (= a+1, a+2, ... up to maximum wave number corresponding to the Nyquist frequency)

You can append '=' after inequality sign. When both addwn and rejwn are set, rejwn will take priority of addwn.

The edge argument is an optional number of channel to drop at the edge of spectrum. If only one value is specified, the same number will be dropped from both sides of the spectrum. Default is to keep all channels. Nested tuples represent individual edge selection for different IFs (a number of spectral channels can be different).

The threshold and chan_avg_limit arguments are linefinder options. The former is the threshold used by line finder. It is better to keep it large as only strong lines affect the baseline solution. The later is a maximum number of consecutive spectral channels to average during the search of weak and broad lines. The default is no averaging (and no search for weak lines). If such lines
can affect the fitted baseline (e.g. a high order polynomial is fitted), increase
this parameter (usually values up to 8 are reasonable). Most users of this
method should find the default value sufficient. See linefinder for more details
on these options.
Note: The best-fit parameter values output in logger and/or blfile are now
based on specunit of 'channel'.

Arguments
**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>insitu</td>
<td>If False a new scantable is returned</td>
<td>bool</td>
<td>None (use default value)</td>
</tr>
<tr>
<td>mask</td>
<td>An optional mask retrieved from scantable</td>
<td>bool array</td>
<td>None (no mask)</td>
</tr>
<tr>
<td>applyfft</td>
<td>Perform Fourier analysis to find appropriate sinusoidal component</td>
<td>bool</td>
<td>True</td>
</tr>
<tr>
<td>fftmethod</td>
<td>Method to find sinusoidal component (currently only 'fft' is available)</td>
<td>string</td>
<td>'fft'</td>
</tr>
<tr>
<td>fftthresh</td>
<td>Threshold to select wave number in Fourier analysis</td>
<td>float, string</td>
<td>3.0</td>
</tr>
<tr>
<td>addwn</td>
<td>Additional wave numbers to be used for fitting</td>
<td>integer, string, any array</td>
<td>[]</td>
</tr>
<tr>
<td>rejwn</td>
<td>Wave numbers not to be used for fitting</td>
<td>integer, string, any array</td>
<td>[]</td>
</tr>
<tr>
<td>clipthresh</td>
<td>Clipping threshold in unit of sigma</td>
<td>float</td>
<td>3.0</td>
</tr>
<tr>
<td>clipniter</td>
<td>Maximum number of iteration of clipping</td>
<td>integer</td>
<td>0</td>
</tr>
<tr>
<td>edge</td>
<td>An optional number of channel to drop at the edge of spectrum</td>
<td>integer, integer array</td>
<td>(0,0)</td>
</tr>
<tr>
<td>threshold</td>
<td>The threshold used by line finder</td>
<td>float</td>
<td>3</td>
</tr>
<tr>
<td>chan_avg_limit</td>
<td>A maximum number of consecutive spectral channels to average during the search</td>
<td>int</td>
<td>1</td>
</tr>
<tr>
<td>plot</td>
<td>Plot the fit and the residual (currently unavailable)</td>
<td>bool</td>
<td>False</td>
</tr>
<tr>
<td>getresidual</td>
<td>If False, return best-fit value instead of residual</td>
<td>bool</td>
<td>True</td>
</tr>
<tr>
<td>showprogress</td>
<td>Show progress status for large data</td>
<td>bool</td>
<td>True</td>
</tr>
<tr>
<td>minnrow</td>
<td>Minimum number of spectra to show progress status</td>
<td>integer</td>
<td>1000</td>
</tr>
<tr>
<td>outlog</td>
<td>Output the coefficients of the best-fit function to logger</td>
<td>bool</td>
<td>False</td>
</tr>
</tbody>
</table>
Returns
scantable

Example

```python
scan = sd.scantable('OrionS_rawACSmod_cal', average=False)
scan2 = scan.auto_sinusoild_baseline(addwn='<=10', insitu=False)
```
sd.scantable.average_beam.html

sd.scantable.average_beam - Function

5.1.2 Average beams together for multi-beam observation

Description

Average the Beams together.
The mask argument is an optional mask defining the region, where the averaging will be applied. The output will have all specified points masked. The weight argument specifies weighting scheme. Valid options are:

- 'none': no weight (default)
- 'var': 1/var(spec) weighted
- 'tsys': 1/Tsys**2 weighted

Arguments

Inputs

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>An optional mask defining the region</td>
<td>bool array</td>
<td>None</td>
</tr>
<tr>
<td>weight</td>
<td>Weighting scheme</td>
<td>string</td>
<td>'none'</td>
</tr>
</tbody>
</table>

Returns

scantable

Example

1937
5.1.2 Average polarizations together

**Description**

Average the Polarisations together.

The mask argument is an optional mask defining the region, where the averaging will be applied. The output will have all specified points masked. The weight argument specifies weighting scheme. Valid options are:

- `'none'`: no weight (default)
- `'var'`: $1/var(spec)$ weighted
- `'tsys'`: $1/Tsys^2$ weighted

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>bool array</td>
<td>None</td>
</tr>
<tr>
<td>weight</td>
<td>string</td>
<td>'none'</td>
</tr>
</tbody>
</table>

**Returns**

scantable

**Example**
sd.scantable.average_time.html

sd.scantable.average_time - Function

5.1.2 Return time average of a scan

Description

Return the (time) weighted average of a scan.
The weight argument specifies weighting scheme. Valid options are:

- 'none': no weight
- 'var': 1/var(spec) weighted
- 'tsys': 1/Tsys**2 weighted
- 'tint': integration time weighted (default)
- 'tintsys': Tint/Tsys**2 weighted
- 'median': median averaging

The align argument is effective only for channel. If it is True, align the spectra in velocity before averaging. It takes the time of the first spectrum as reference time.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>An optional mask (only used for 'var' and 'tsys' weighting)</td>
<td>bool array</td>
<td>None</td>
</tr>
<tr>
<td>scanav</td>
<td>True averages each scan separately, False averages all scans together</td>
<td>bool</td>
<td>False</td>
</tr>
<tr>
<td>weight</td>
<td>Weighting scheme</td>
<td>string</td>
<td>'none'</td>
</tr>
<tr>
<td>align</td>
<td>Align the spectral in velocity before averaging</td>
<td>bool</td>
<td>False</td>
</tr>
<tr>
<td>compel</td>
<td>True forces to average overwrapped IFs</td>
<td>bool</td>
<td>False</td>
</tr>
</tbody>
</table>

Returns

1939
Example

```python
scan = sd.scantable('OrionS_rawACSmod_cal', average=False)
# time average the scantable without using a mask
newscan = scan.average_time()
```
sd.scantable.bin.html

sd.scantable.bin - Function

5.1.2 Perform binning of spectra

Description

Return a scan where all spectra have been binned up. If insitu is True, or
insitu is None and sd.rcParams['insitu'] is True, the method will not return the
result, but apply operation on this scantable.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>The bin width in pixels</td>
</tr>
<tr>
<td>allowed:</td>
<td>integer</td>
</tr>
<tr>
<td>Default:</td>
<td>5</td>
</tr>
<tr>
<td>insitu</td>
<td>If False a new scantable is returned</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>None (use default value)</td>
</tr>
</tbody>
</table>

Returns

scantable

Example

1941
sd.scantable.chan2data.html

**sd.scantable.chan2data - Function**

5.1.2 Return channel/frequency/velocity and spectral value at an arbitrary row and channel

**Description**

Returns channel/frequency/velocity and spectral value at an arbitrary row and channel in the scantable. The returned value is a tuple with length of 2. The first element is a dictionary that contains an unit and a value for the abcissa, while the second one is also a dictionary that contains an unit and a value for the ordinate.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowno</td>
</tr>
<tr>
<td>chan</td>
</tr>
</tbody>
</table>

**Allowed**

<table>
<thead>
<tr>
<th>rowno</th>
<th>integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>chan</td>
<td>integer</td>
</tr>
</tbody>
</table>

**Default**

<table>
<thead>
<tr>
<th>rowno</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>chan</td>
<td>0</td>
</tr>
</tbody>
</table>

**Returns**

dictionary array

**Example**

```python
s = sd.scantable('OrionS_rawACSmod_cal', average=False)
s.chan2data(rowno=0, chan=0)
({'unit': 'channel', 'value': 0.0}, {'unit': 'K', 'value': 1.7028001546859741})
```
sd.scantable.clip.html

**sd.scantable.clip - Function**

5.1.2 Flag data by its spectral value

**Description**

Flag the selected data outside a specified range (in channel-base). The method requires to set upper and lower threshold for spectral value. If clipoutside is True, the data outside the range will be flagged/unflagged. On the other hand, if it is False, the data inside the range will be flagged/unflagged. The operation if flag or unflag is controlled by the unflag argument.

Note that the operation will always be applied to this scantable regardless of the value of sd.rcParams['insitu'].

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uthres</td>
<td>Upper threshold</td>
</tr>
<tr>
<td>allowed:</td>
<td>float</td>
</tr>
<tr>
<td>Default:</td>
<td>None</td>
</tr>
<tr>
<td>dthres</td>
<td>Lower threshold</td>
</tr>
<tr>
<td>allowed:</td>
<td>float</td>
</tr>
<tr>
<td>Default:</td>
<td>None</td>
</tr>
<tr>
<td>clipoutside</td>
<td>True for flagging data outside the range, False for flagging data inside the range</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>True</td>
</tr>
<tr>
<td>unflag</td>
<td>If True, unflag the data</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>False</td>
</tr>
</tbody>
</table>

**Returns**

**Example**

1943
sd.scantable.convert_flux.html

**sd.scantable.convert_flux - Function**

5.1.2 Return a scan where all spectra are converted to Jy or K

**Description**

Return a scan where all spectra are converted to either Jansky or Kelvin depending upon the flux units of the scan table. By default the function tries to look the values up internally. If it can’t find them (or if you want to over-ride), you must specify EITHER jyperk OR eta (and D which it will try to look up also if you don’t set it). jyperk takes precedence if you set both. If insitu is True, or insitu is None and sd.rcParams['insitu'] is True, the method will not return the result, but apply operation on this scantable.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jyperk</td>
<td>The Jy/K conversion factor</td>
</tr>
<tr>
<td>allowed: float</td>
<td>Default: None</td>
</tr>
<tr>
<td>eta</td>
<td>The aperture efficiency</td>
</tr>
<tr>
<td>allowed: float</td>
<td>Default: None</td>
</tr>
<tr>
<td>d</td>
<td>The geometric diameter (metres)</td>
</tr>
<tr>
<td>allowed: float</td>
<td>Default: None</td>
</tr>
<tr>
<td>insitu</td>
<td>If False a new scantable is returned</td>
</tr>
<tr>
<td>allowed: bool</td>
<td>Default: None</td>
</tr>
</tbody>
</table>

**Returns**

c scantable

**Example**
sd.scantable.convert_pol.html

**sd.scantable.convert_pol - Function**

5.1.2 Convert data to a different polarization type

**Description**

Convert the data to a different polarisation type. Note that you will need cross-polarisation terms for most conversions. The poltype argument specifies the new polarization type. Valid types are: 'linear', 'circular', 'stokes', and 'linpol'.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>poltype</td>
<td>The new polarization type</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td>None</td>
</tr>
</tbody>
</table>

**Returns**

scantable

**Example**
sd.scantable.copy.html

sd.scantable.copy - Function

5.1.2 Return a copy of this scantable

Description

Return a copy of this scantable.
Note that this makes a full (deep) copy. scan2 = scan1 makes a reference.

Arguments

Returns

scantable

Example

```python
s = sd.scantable('OrionS_rawACSmod_cal', average=True)
# deep copy
copiedscan = s.copy()
# this makes a reference
s2 = s
```
sd.scantable.create_mask.html

**sd.scantable.create_mask - Function**

5.1.2 Return a bool array based on [min,max] windows

**Description**

Compute and return a mask based on [min, max] windows. The specified windows are to be INCLUDED, when the mask is applied. The mask window should be given as a pairs of start/end points, e.g. [min, max], [min2, max2], ... If the invert argument specified as True, return an inverted mask, i.e. the regions specified are EXCLUDED. The mask is created using the specified row for unit conversions. This is only necessary if frequency varies over rows.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Pairs of start/end points</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>list or sequence of lists</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>invert</th>
<th>Determine the operation is inclusive or exclusive</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>False (inclusive)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>row</th>
<th>create the mask using the specified row for unit conversions</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>integer</td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
</tr>
</tbody>
</table>

**Returns**

bool array

**Example**

```python
scan = sd.scantable('OrionS_rawACSmod_cal',average=True)
scan.set_unit('channel')
# a)
msk = scan.create_mask([[400, 500], [800, 900]])
# masks everything outside 400 and 500
# and 800 and 900 in the unit 'channel'
```
# b)
msk = scan.create_mask([400, 500], [800, 900], invert=True)
# masks the regions between 400 and 500
# and 800 and 900 in the unit 'channel'

# c)
#mask only channel 400
msk = scan.create_mask([400])
Perform a baseline subtraction using cubic spline function

Description

Return a scan which has been baselined (all rows) by cubic spline function (piecewise cubic polynomial). Fit will be done with 'sigma-clipping'. Note: The best-fit parameter values output in logger and/or blfile are now based on specunit of 'channel'.

Arguments
### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>insitu</td>
<td>If False a new scantable is returned</td>
<td>bool</td>
<td>None (use default value)</td>
</tr>
<tr>
<td>mask</td>
<td>An optional mask retrieved from scantable</td>
<td>bool</td>
<td>None (no mask)</td>
</tr>
<tr>
<td>npiece</td>
<td>Number of pieces</td>
<td>integer</td>
<td>2</td>
</tr>
<tr>
<td>clipthresh</td>
<td>Clipping threshold in unit of sigma</td>
<td>float</td>
<td>3.0</td>
</tr>
<tr>
<td>clipniter</td>
<td>Maximum number of iteration of clipping</td>
<td>integer</td>
<td>0</td>
</tr>
<tr>
<td>plot</td>
<td>Plot the fit and the residual (currently unavailable)</td>
<td>bool</td>
<td>False</td>
</tr>
<tr>
<td>getresidual</td>
<td>If False, return best-fit value instead of residual</td>
<td>bool</td>
<td>True</td>
</tr>
<tr>
<td>showprogress</td>
<td>Show progress status for large data</td>
<td>bool</td>
<td>True</td>
</tr>
<tr>
<td>minnrow</td>
<td>Minimum number of spectra to show progress status</td>
<td>integer</td>
<td>1000</td>
</tr>
<tr>
<td>outlog</td>
<td>Output the coefficients of the best-fit function to logger</td>
<td>bool</td>
<td>False</td>
</tr>
<tr>
<td>blfile</td>
<td>Name of text file in which the best-fit parameter values to be written</td>
<td>string</td>
<td>&quot;</td>
</tr>
</tbody>
</table>

### Returns

- scantable

### Example

1950
scan = sd.scantable('OrionS_rawACSmed_cal', average=False)
# return a scan baselined by a cubic spline consisting of 2 pieces (i.e., 1 internal
# also with 3-sigma clipping, iteration up to 4 times
bscan = scan.cspline_baseline(npiece=2, clipthresh=3.0, clipniter=4)
sd.scantable.drop_scan.html

sd.scantable.drop_scan - Function

5.1.2 Return a new scantable where the specified scan number(s) are dropped

Description

Return a new scantable where the specified scan number(s) has(have) been dropped.
It always returns a new scantable regardless of sd.rcParams['insitu'].

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scanid</td>
<td>A (list of) scan number(s)</td>
</tr>
<tr>
<td>allowed:</td>
<td>integer, integer array</td>
</tr>
<tr>
<td>Default:</td>
<td>None</td>
</tr>
</tbody>
</table>

Returns

scantable

Example
sd.scantable.fft.html

**sd.scantable.fft - Function**

5.1.2 Apply FFT to the spectra

**Description**

Apply FFT to the spectra. Flagged data in the scantable get interpolated over the region. An optional channel mask specified by mask parameter is applied to all specified rows if it is given as one dimensional array. It returns a list of dictionaries containing the results for each spectrum. Each dictionary contains two values, the real and the imaginary parts when getrealimag = True, or the amplitude (absolute value) and the phase (argument) when getrealimag = False. The key for these values are 'real' and 'imag', or 'ampl' and 'phase', respectively.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowno</td>
<td>Row number(s) to be processed.</td>
<td>integer, integer list or tuple</td>
<td>[] (apply to whole data)</td>
</tr>
<tr>
<td>mask</td>
<td>An optional channel mask</td>
<td>bool array</td>
<td>[]</td>
</tr>
<tr>
<td>getrealimag</td>
<td>If True, return real and imaginary part instead of amplitude and phase</td>
<td>bool</td>
<td>False (return amplitude and phase)</td>
</tr>
</tbody>
</table>

**Returns**

dictionary

**Example**

1953
sd.scantable.flag.html

**sd.scantable.flag - Function**

5.1.2 Flag selected data using specified mask (channel based flag)

**Description**

Flag the selected data using an optional channel mask. The appropriate value of the mask can be created with `create_mask`. If no mask is specified, all channels are flagged/unflagged depending on the unflag argument. Note that the operation will always be applied to this scantable regardless of the value of sd.rcParams['insitu'].

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>An optional channel mask</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool array</td>
</tr>
<tr>
<td>Default:</td>
<td>None (all channels)</td>
</tr>
<tr>
<td>unflag</td>
<td>If True, unflag the data</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>False</td>
</tr>
</tbody>
</table>

**Returns**

**Example**
sd.scantable.flag_nans.html

 sd.scantable.flag_nans - Function

5.1.2 Flag NaN values

Description

Utility function to flag NaN values in the scantable.
Note that the operation will always be applied to this scantable regardless of the value of sd.rcParams['insitu'].

Arguments

Returns

Example
sd.scantable.flag_row.html

**sd.scantable.flag_row - Function**

5.1.2 Flag spectra based on specified rows (row based flag)

**Description**

Flag the selected data in row-based manner. Note that the operation will always be applied to this scantable regardless of the value of sd.rcParams['insitu'].

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rows</td>
<td>List of row numbers to be flagged</td>
</tr>
<tr>
<td></td>
<td>allowed: integer, integer array</td>
</tr>
<tr>
<td></td>
<td>Default: []</td>
</tr>
<tr>
<td>unflag</td>
<td>If True, unflag the data</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: False</td>
</tr>
</tbody>
</table>

**Returns**

**Example**
sd.scantable.freq_align.html

**sd.scantable.freq_align** - Function

5.1.2 Perform frequency alignment

**Description**

Return a scan where all rows have been aligned in frequency/velocity. The alignment frequency frame (e.g. LSRK) is set by function `set_freqframe`. The reference time to align can be specified as `reftime` argument. By default, the time of the first row of data is used. The method argument specifies interpolation method for regridding the spectra. Valid options are 'nearest', 'linear', 'cubic' (default), and 'spline'.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>reftime</code></td>
<td>Reference time to align at.</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: None (use first row of data)</td>
</tr>
<tr>
<td><code>method</code></td>
<td>Interpolation method for regridding the spectra</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: 'cubic'</td>
</tr>
<tr>
<td><code>in situ</code></td>
<td>If False a new scantable is returned</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
</tbody>
</table>
|          | Default: None                                                                           

**Returns**

`scantable`

**Example**
sd.scantable.freq_switch.html

sd.scantable.freq_switch - Function

5.1.2 Apply frequency switching to the data

Description

Apply frequency switching to the data.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>insitu</td>
<td>If False a new scantable is returned</td>
</tr>
<tr>
<td>allowed</td>
<td>bool</td>
</tr>
<tr>
<td>Default</td>
<td>None</td>
</tr>
</tbody>
</table>

Returns

scantable

Example

1958
sd.scantable.gain_el.html

sd.scantable.gain_el - Function

5.1.2 Apply gain-elevation correction based on user-provided correction factors

Description

Return a scan after applying a gain-elevation correction. The correction can be made via either a polynomial or a table-based interpolation (and extrapolation if necessary). You specify polynomial coefficients, an ascii table or neither. If you specify neither, then a polynomial correction will be made with built in coefficients known for certain telescopes (an error will occur if the instrument is not known). The data and Tsys are *divided* by the scaling factors.

The polynomial coefficients to compute a gain-elevation correction should be given as that of a function of elevation in degrees. Contents of the ascii table specified by filename argument is correction factors as a function of time and elevation (in degree). The first row of the ascii file must give the column names and these MUST include columns "ELEVATION" (degrees) and "FACTOR" (multiply data by this) somewhere. The second row must give the data type of the column. Use 'R' for Real and 'I' for Integer. An example file would be (actual factors are arbitrary) :

```
TIME   ELEVATION   FACTOR
R      R           R
0.1    0           0.8
0.2    20          0.85
0.3    40          0.9
0.4    60          0.85
0.5    80          0.8
0.6    90          0.75
```

The interpolation method can be specified by method argument. Valid options are 'nearest', 'linear' (default), 'cubic', and 'spline'.

Arguments
## Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>poly</td>
<td>Polynomial coefficients to compute a gain-elevation correction</td>
<td>float array</td>
<td>None</td>
</tr>
<tr>
<td>filename</td>
<td>The name of an ascii file holding correction factors</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>method</td>
<td>Interpolation method when correcting from a table</td>
<td>string</td>
<td>'linear'</td>
</tr>
<tr>
<td>insitu</td>
<td>If False a new scantable is returned</td>
<td>bool</td>
<td>None</td>
</tr>
</tbody>
</table>

## Returns

`scantable`

## Example

1960
sd.scantable.get_abcissa.html

sd.scantable.get_abcissa - Function

5.1.2 Get the abcissa values and format string that represents current coordinate setup

Description

Get the abcissa in the current coordinate setup for the currently selected Beam/IF/Pol. The method returns the abcissa values and the format string as a dictionary.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowno</td>
<td>An optional row number in the scantable</td>
</tr>
</tbody>
</table>

Allowed: integer
Default: 0

Returns
dictionary

Example
sd.scantable.get_antennaname.html

sd.scantable.get_antennaname - Function

5.1.2 Get antenna name

Description

Return a name of antenna observed.

Arguments

Returns

string

Example
5.1.2 Get a list of azimuth during the observation

**Description**

Get a list of azimuths for the observations. Return a float for each integration in the scantable. The unit is radian.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>Row no of integration</td>
</tr>
<tr>
<td></td>
<td>allowed: integer</td>
</tr>
<tr>
<td></td>
<td>Default: -1 (all rows)</td>
</tr>
</tbody>
</table>

**Returns**

float, float array

**Example**

1963
sd.scantable.get_column_names.html

sd.scantable.get_column_names - Function

5.1.2 Get a list of column names in the main table

Description

Return a list of column names in the main table, which can be used for selection.

Arguments

Returns
string array

Example
sd.scantable.get_coordinate.html

sd.scantable.get_coordinate - Function

5.1.2 Return the spectral coordinate for a given row as a coordinate object

Description

Return the (spectral) coordinate for a a given 'rowno'.

Notes:

- This coordinate is only valid until a scantable method modifies the frequency axis.
- This coordinate does contain the original frequency set-up NOT the new frame. The conversions however are done using the user specified frame (e.g. LSRK/TOPO). To get the ‘real’ coordinate, use scantable.freq_align first. Without it there is no closure, i.e.:

  ```python
  c = myscan.get_coordinate(0)
  c.to_frequency(c.get_reference_pixel()) !== c.get_reference_value()
  ```

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowno</td>
<td>The row number for the spectral coordinate</td>
</tr>
<tr>
<td>allowed:</td>
<td>integer</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

Returns

coordinate

Example
sd.scantable.get_direction.html

sd.scantable.get_direction - Function

5.1.2 Get a list of positions on the sky as a string

Description

Get a list of Positions on the sky (direction) for the observations. Return a string for each integration in the scantable.

To get float value for positions, use get_directionval.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>Row no of integration</td>
</tr>
<tr>
<td>allowed:</td>
<td>integer</td>
</tr>
<tr>
<td>Default:</td>
<td>-1 (all rows)</td>
</tr>
</tbody>
</table>

Returns

string, string array

Example

```python
s=sd.scantable('OrionS_rawACSmod_cal',average=False)
# get_direction() returns string expression of position
s.get_direction(0)
'05:35:13.5 -05.24.08.2'
# s.get_directionval() returns float value of position
s.get_directionval(0)
[1.4626913601468896, -0.0942875343295448]
```
sd.scantable.get_directionval.html

sd.scantable.get_directionval - Function

5.1.2 Get a list of positions on the sky as a float

Description

Get a list of Positions on the sky (direction) for the observations. Return a float for each integration in the scantable. The unit is radian. To get string expression for positions, use get_direction.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Row no of integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>allowed: integer</td>
</tr>
<tr>
<td></td>
<td>Default: -1 (all rows)</td>
</tr>
</tbody>
</table>

Returns

float array

Example

s=sd.scantable('OrionS_rawACSmod_cal',average=False)
# get_direction() returns string expression of position
s.get_direction(0)
'05:35:13.5 -05.24.08.2'
# s.get_directionval() returns float value of position
s.get_directionval(0)
[1.4626913601468896, -0.0942875343295448]
sd.scantable.get_elevation.html

**sd.scantable.get_elevation - Function**

5.1.2 Get a list of elevation during the observation

**Description**

Get a list of elevations for the observations. Return a float for each integration in the scantable. The unit is radian.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Row no of integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>integer</td>
</tr>
<tr>
<td>allowed</td>
<td>integer</td>
</tr>
<tr>
<td>Default</td>
<td>-1 (all rows)</td>
</tr>
</tbody>
</table>

**Returns**

float, float array

**Example**
sd.scantable.get_fit.html

**sd.scantable.get_fit - Function**

5.1.2 Get the stored fits for a row in scantable

**Description**

Print or return the stored fits for a row in the scantable

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>The row which the fit has been applied to</td>
</tr>
</tbody>
</table>

allowed: integer  
Default: 0

**Returns**

dictionary

**Example**

1969
sd.scantable.get_fluxunit.html

sd.scantable.get_fluxunit - Function

5.1.2 Get a flux unit

Description

Return a flux unit string.

Arguments

Returns

string

Example
**sd.scantable.get_inttime - Function**

5.1.2 Get a list of integration times for the observation

**Description**

Get a list of integration times for the observations. Return a time in seconds for each integration in the scantable.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>The row which the fit has been applied to</td>
</tr>
<tr>
<td>allowed</td>
<td>integer</td>
</tr>
<tr>
<td>Default</td>
<td>-1 (all rows)</td>
</tr>
</tbody>
</table>

**Returns**

float, float array

**Example**
sd.scantable.get_mask.html

**sd.scantable.get_mask - Function**

5.1.2 Get mask for the specified row as a bool list

**Description**

Return the mask for the current row in the scantable as a list.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowno</td>
<td>The row number to retrieve the mask from</td>
</tr>
<tr>
<td>allowed: integer</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

bool array

**Example**
sd.scantable.get_mask_indices.html

**sd.scantable.get_mask_indices - Function**

5.1.2 Compute and return lists of mask start indices and mask end indices from the given bool array

**Description**

Compute and Return lists of mask start indices and mask end indices. Returned value is a list of mask start indices and that of mask end indices, i.e., ([istart1,istart2,...], [iend1,iend2,...]).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>Channel mask, created with create_mask</td>
</tr>
<tr>
<td>allowed</td>
<td>bool array</td>
</tr>
<tr>
<td>Default</td>
<td>None</td>
</tr>
</tbody>
</table>

**Returns**

integer array

**Example**

```python
s=sd.scantable('OrionS_rawACSmod_cal',average=False)
# this is an example to show how get_mask_indices() works
s.get_mask_indices(s.create_mask([1000,2000],[4000,5000]))  
([1000, 4000], [2000, 5000])
```
sd.scantable.get_masklist.html

**sd.scantable.get_masklist - Function**

5.1.2 Compute and return a list of mask windows \([\text{min, max}]\) from the given bool array

**Description**

Compute and return a list of mask windows, \([\text{min, max}]\). Returned value is pairs of start/end points (inclusive) specifying the masked regions, i.e. \([\text{min, max}], [\text{min2, max2}], \ldots\)

The row argument specifies the row to use for unit conversions, default is row=0. It is only necessary if frequency varies over rows.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Channel mask, created with create_mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>allowed: bool array</td>
</tr>
<tr>
<td></td>
<td>Default: None</td>
</tr>
<tr>
<td>row</td>
<td>calculate the masklist using the specified row for unit conversion</td>
</tr>
<tr>
<td></td>
<td>allowed: integer</td>
</tr>
<tr>
<td></td>
<td>Default: 0</td>
</tr>
<tr>
<td>silent</td>
<td>True for silent mode</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: False</td>
</tr>
</tbody>
</table>

**Returns**

float array

**Example**

```python
s=sd.scantable('OrionS_rawACSmod_cal',average=False)
# this is an example to show how get_masklist() works
s.get_masklist(s.create_mask([[1000,2000],[4000,5000]]))
([(1000.0, 2000.0), [4000.0, 5000.0]])
```
sd.scantable.get_parangle.html

sd.scantable.get_parangle - Function

5.1.2 Get a list of parallactic angles for the observation

Description

Get a list of parallactic angles for the observations. Return a float for each integration in the scantable.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Row no of integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>integer</td>
</tr>
<tr>
<td>allowed</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>-1 (all rows)</td>
</tr>
</tbody>
</table>

Returns

float, float array

Example
5.1.2 Get the rest frequency(s) stored in the scantable

**Description**

Get the rest frequency(s) stored in this scantable. The return value(s) are always of unit 'Hz'. The method returns a dictionary containing ids and a list of doubles for each id. The rest frequency is stored in MOLECULES subtable of the scantable. Rows in MOLECULES subtable is referred from main table by MOLECULE_ID. You can specify a list of that id using ids argument to retrieve particular rest frequency(s).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>ids</td>
</tr>
<tr>
<td>A list of MOLECULE_ID for that rest frequency(s) to be retrieved</td>
</tr>
<tr>
<td>allowed:</td>
</tr>
<tr>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

dictionary

**Example**

1976
sd.scantable.get_rms.html

**sd.scantable.get_rms - Function**

5.1.2 Calculate rms of the spectrum

**Description**

Calculate rms of the spectrum. Mask can be specified.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>Optional mask for calculation</td>
<td>bool array</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td></td>
</tr>
<tr>
<td>whichrow</td>
<td>Row number to be processed</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

float

**Example**

1977
sd.scantable.get_row.html

**sd.scantable.get_row** - Function

5.1.2 Return a scantable with single row

**Description**

Select a row in the scantable. Return a scantable with single row. If insitu is True, or insitu is None and sd.rcParams['insitu'] is True, the method will not return the result, but apply operation on this scantable.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>Row no of integration</td>
<td>integer</td>
<td>0</td>
</tr>
<tr>
<td>insitu</td>
<td>If False a new scantable is returned</td>
<td>bool</td>
<td>None</td>
</tr>
</tbody>
</table>

**Returns**

scantable

**Example**

1978
sd.scantable.get_row_selector.html

**sd.scantable.get_row_selector** - Function

5.1.2 Return a selector object that only selects target row

**Description**

Return a selector object that only selects target row.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>The row number to select</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowno</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>integer</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

selector

**Example**

1979
sd.scantable.get_scan.html

**sd.scantable.get_scan - Function**

5.1.2 Return a specified scan(s) specified by scan number or source name as a new scantable

**Description**

Return a specific scan (by scanno) or collection of scans (by source name) in a new scantable. In both case, you can set selection criteria to scanid argument. For source name, unix-style patterns are accepted for source name matching, e.g. '*_R' gets all 'ref scans.

Note that [drop_scan](#) is an inverse operation.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>A (list of) scanno or a source name</th>
</tr>
</thead>
<tbody>
<tr>
<td>scanid</td>
<td>allowed: integer, integer array, string</td>
</tr>
<tr>
<td></td>
<td>Default: None</td>
</tr>
</tbody>
</table>

**Returns**

scantable

**Example**

```python
scan=sd.scantable('data.asap')
# get all scans containing the source '323p459'
newscan = scan.get_scan('323p459')
# get all 'off' scans
refscans = scan.get_scan('*_R')
# get a subset of scans by scanno (as listed in scan.summary())
newscan = scan.get_scan([0, 2, 7, 10])
```
sd.scantable.get_selection.html

sd.scantable.get_selection - Function

5.1.2 Get current selection that is currently set on this scantable

Description

Get the selection object currently set on this scantable.

Arguments

Returns
selector

Example

```python
scan = sd.scantable('OrionS_rawACSmod')
sel = scan.get_selection()
sel.set_ifs(0)   # select IF 0
scan.set_selection(sel)   # apply modified selection
```
sd.scantable.get_sourcename.html

**sd.scantable.get_sourcename - Function**

5.1.2 Get a list of source names for the observation

**Description**

Get a list source names for the observations. Return a string for each integration in the scantable.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Row no of integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>Default: -1 (all rows)</td>
</tr>
</tbody>
</table>

**Returns**

string, string array

**Example**
**sd.scantable.get_spectrum.html**

**sd.scantable.get_spectrum - Function**

5.1.2 Get the spectrum for the current row

**Description**

Return the spectrum for the current row in the scantable as a list.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowno</td>
<td>The row number to retrieve the spectrum from</td>
</tr>
</tbody>
</table>

| allowed: | integer |
| Default: |        |

**Returns**

float array

**Example**

1983
sd.scantable.get_time.html

sd.scantable.get_time - Function

5.1.2 Get a list of time stamps for the observation

Description

Get a list of time stamps for the observations. Return a datetime object or a string (default) for each integration time stamp in the scantable.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>Row no of integration</td>
<td>integer</td>
<td>-1 (all rows)</td>
</tr>
<tr>
<td>asdatetime</td>
<td>Return values as datetime objects rather than strings</td>
<td>bool</td>
<td>False</td>
</tr>
</tbody>
</table>

Returns

string, datetime, their array

Example
sd.scantable.get_tsys.html

**sd.scantable.get_tsys - Function**

5.1.2 Get a list of system temperatures

**Description**

Return the System temperatures.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>The rowno to get the information for</td>
</tr>
<tr>
<td></td>
<td>allowed: integer</td>
</tr>
<tr>
<td></td>
<td>Default: -1 (all rows)</td>
</tr>
</tbody>
</table>

**Returns**

float array

**Example**

1985
Get the default unit set for spectral axis in this scantable.

Arguments

Returns

string

Example
sd.scantable.get_weather.html

sd.scantable.get_weather - Function

5.1.2 Get the weather informations

Description

Return the weather informations.
The contents of returned dictionary is as follows:

- humidity: relative humidity
- pressure: atmospheric pressure
- temperature: atmospheric temperature
- windaz: wind direction in radian
- windspeed: wind speed in m/s

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>The rowno to get the information for</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>-1 (all rows)</td>
<td></td>
</tr>
</tbody>
</table>

Returns
dictionary, dictionary array

Example

1987
sd.scantable.getbeam.html

**sd.scantable.getbeam - Function**

5.1.2 Get beam number of the given row

**Description**

Return a beam number for the given row.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>The rowno to get the information for</td>
<td></td>
</tr>
<tr>
<td>allowed: integer</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

integer

**Example**

```python
scan = sd.scantable('OrionS_rawACSmod')
beam = scan.getbeam(0) # get beam number for the first row
```
sd.scantable.getbeamnos.html

**sd.scantable.getbeamnos** - Function

5.1.2 Get a list of beam numbers in the scantable

**Description**

Return a list of beam numbers in the scantable.

**Arguments**

**Returns**

integer array

**Example**

1989
sd.scantable.getcycle.html

sd.scantable.getcycle - Function

5.1.2 Get cycle number of the given row

Description

Return a cycle number for the given row.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>The rowno to get the information for</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>allowed: integer</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

integer

Example

```python
scan = sd.scantable('OrionS_rawACSmod')
cycle = scan.getcycle(0) # get cycle number for the first row
```
sd.scantable.getif.html

**sd.scantable.getif - Function**

5.1.2 Get IF number of the given row

**Description**

Return a IF number for the given row.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>The rowno to get the information for</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>integer</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

integer

**Example**

```python
scan = sd.scantable('OrionS_rawACSmod')
ifno = scan.getifs(0) # get IF number for the first row
```
sd.scantable.getifnos.html

sd.scantable.getifnos - Function

5.1.2 Get a list of IF numbers in the scantable

Description

Return a list of IF numbers in the scantable.

Arguments

Returns

integer array

Example
sd.scantable.getmolnos.html

sd.scantable.getmolnos - Function

5.1.2 Get a list of molecule ids in the scantable

Description

Return a list of molecule ids in the scantable.

Arguments

Returns

integer array

Example
Get polarization number of the given row

Return a polarization number for the given row.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>The rowno to get the information for</td>
</tr>
<tr>
<td>allowed: integer</td>
</tr>
<tr>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

integer

Example

```python
scan = sd.scantable('OrionS_rawACSmod')
polno = scan.getpol(0)  # get polarization number for the first row
```
sd.scantable.getpolnos.html

sd.scantable.getpolnos - Function

5.1.2 Get a list of polarization numbers in the scantable

Description

Return a list of polarization numbers in the scantable.

Arguments

Returns

integer array

Example
**sd.scantable.getscan.html**

**sd.scantable.getscan - Function**

5.1.2 Get scan number of the given row

**Description**

Return a scan number for the given row.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>The rowno to get the information for</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>allowed: integer</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

integer

**Example**

```python
scan = sd.scantable('OrionS_rawACSmod')
scanno = scan.getscan(0) # get scan number for the first row
```
sd.scantable.getscannos.html

sd.scantable.getscannos - Function

5.1.2 Get a list of scan numbers in the scantable

Description

Return a list of scan numbers in the scantable.

Arguments

Returns
integer array

Example
sd.scantable.history.html

sd.scantable.history - Function

5.1.2 Print a history

Description

Print the history. Optionally to a file.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>The name of the file to save the history to</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: None</td>
</tr>
</tbody>
</table>

Returns

string

Example

1998
sd.scantable.invert_phase.html

**sd.scantable.invert_phase** - Function

5.1.2 Invert the phase of the complex polarization

**Description**

Invert the phase of the complex polarization.

**Arguments**

**Returns**

**Example**
5.1.2 Perform Fourier filtering on the spectra

**Description**

Flag the data in 'lag' space by providing a frequency to remove. Flagged data in the scantable gets interpolated over the region. No taper is applied. If insitu is True, or insitu is None and sd.rcParams['insitu'] is True, the method will not return the result, but apply operation on this scantable. It is recommended to flag edges of the band or strong signals beforehand.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>The start frequency (really a period within the bandwidth) or period to remove</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td>The end frequency or period to remove</td>
<td>string</td>
<td>None</td>
</tr>
<tr>
<td>unit</td>
<td>The frequency unit or &quot; for explicit lag channels</td>
<td>string</td>
<td>'MHz'</td>
</tr>
<tr>
<td>insitu</td>
<td>If False a new scantable is returned</td>
<td>bool</td>
<td>None</td>
</tr>
</tbody>
</table>

**Returns**

scantable

**Example**

2000
sd.scantable.mx_quotient.html

sd.scantable.mx_quotient - Function

5.1.2 Form a quotient using "off" beams when observing in "MX" mode

Description

Form a quotient using "off" beams when observing in "MX" mode. The formula to get result depends on the preserve parameter. If it is True, the continuum will be preserved while if it is False, the continuum will be removed. The equation used are

\[ T_a = T_{sys}^{OFF} \frac{ON}{OFF} - T_{sys}^{OFF}, \]

if preserve is True, while

\[ T_a = T_{sys}^{OFF} \frac{ON}{OFF} - T_{sys}^{ON}, \]

if preserve is False.

The weight argument is used for time averaging off beams. You can set any options same as [average_time](#).

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>An optional mask to be used when weight == 'stddev'</td>
<td></td>
</tr>
<tr>
<td>allowed</td>
<td>bool array</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>weight</td>
<td>How to average the off beams</td>
<td></td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>'median'</td>
<td></td>
</tr>
<tr>
<td>preserve</td>
<td>Preserve the continuum or remove it</td>
<td></td>
</tr>
<tr>
<td>allowed</td>
<td>bool</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>True</td>
<td></td>
</tr>
</tbody>
</table>

Returns

scantable

Example

2001
sd.scantable.nbeam.html

**sd.scantable.nbeam - Function**

5.1.2 Return a number of beams

**Description**

Return a number of beams in this scantable.
If scanno argument is specified, the number of beams for that scan will be returned. Otherwise, the total number of beams, which is written in the header of the data, will be returned.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>scanno</td>
<td></td>
</tr>
</tbody>
</table>

**Scan number**

**allowed:** integer

**Default:** -1 (total)

**Returns**

integer

**Example**

2002
sd.scantable.nchan.html

**sd.scantable.nchan - Function**

5.1.2 Return a number of channels

**Description**

Return a number of channels in this scantable. If ifno argument is specified, the number of channels for that IF will be returned. Otherwise, the maximum number of channels, which is written in the header of the data, will be returned.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>IF number</th>
</tr>
</thead>
<tbody>
<tr>
<td>ifno</td>
<td>integer</td>
</tr>
</tbody>
</table>

Allowed: integer

Default: -1 (maximum)

**Returns**

integer

**Example**

2003
sd.scantable.ncycle.html

**sd.scantable.ncycle - Function**

5.1.2 Return a number of cycles

**Description**

Return a number of cycles in this scantable. If `scanno` argument is specified, the number of cycles for that scan will be returned. Otherwise, the total number of cycles will be returned.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>scanno</td>
<td>Scan number</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td>-1 (total)</td>
</tr>
</tbody>
</table>

**Returns**

integer

**Example**

2004
Return a number of IFs in this scantable. If scanno argument is specified, the number of IFs for that scan will be returned. Otherwise, the total number of IFs, which is written in the header of the data, will be returned.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scanno</td>
<td>Scan number</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>allowed: integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default: -1 (total)</td>
</tr>
</tbody>
</table>

Returns

integer

Example

2005
sd.scantable.npol.html

sd.scantable.npol - Function

5.1.2 Return a number of polarizations

Description

Return a number of polarizations in this scantable. If scanno argument is specified, the number of polarizations for that scan will be returned. Otherwise, the total number of polarizations, which is written in the header of the data, will be returned.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>scanno</td>
<td>Scan number</td>
</tr>
<tr>
<td></td>
<td>allowed: integer</td>
</tr>
<tr>
<td></td>
<td>Default: -1 (total)</td>
</tr>
</tbody>
</table>

Returns

integer

Example

2006
sd.scantable.nrow.html

**sd.scantable.nrow - Function**

5.1.2 Return a number of rows

**Description**

Return a total number of rows (= total number of spectra) in this scantable.

**Arguments**

**Returns**

integer

**Example**
sd.scantable.nscan.html

**sd.scantable.nscan - Function**

5.1.2 Return a number of scans

**Description**

Return a total number of scans in this scantable.

**Arguments**

**Returns**

integer

**Example**
sd.scantable.opacity.html

sd.scantable.opacity - Function

5.1.2 Apply an opacity correction

Description

Apply an opacity correction. The data and Tsys are multiplied by the correction factor. Correction includes an elevation dependence of the opacity. Actual correction factor is \( \exp(\tau \times ZD) \), where \( ZD \) is the zenith-distance and \( \tau \) is a value given as the tau argument. If a list of opacities is provided, it has to be of length nIF, nIF*nPol or 1 and in order of IF/POL, e.g. [opif0pol0, opif0pol1, opif1pol0 ...]. If tau is ‘None’ the opacities are determined from a model. If insitu is True, or insitu is None and sd.rcParams['insitu'] is True, the method will not return the result, but apply operation on this scantable.

Arguments

Inputs

<table>
<thead>
<tr>
<th>tau</th>
<th>A (list of) opacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>float, float array</td>
</tr>
<tr>
<td>Default:</td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>insitu</th>
<th>If False a new scantable is returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>None</td>
</tr>
</tbody>
</table>

Returns

scantable

Example
sd.scantable.parallactify.html

**sd.scantable.parallactify - Function**

5.1.2 Set a flag to indicate the data should be treated as "parallactified"

**Description**

Set a flag to indicate whether this data should be treated as having been 'parallactified' (total phase == 0.0)

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pflag</td>
<td>Bool indicating whether to turn this on (True) or off (False)</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

**Example**
sd.scantable.poltype.html

**sd.scantable.poltype** - Function

5.1.2 Get a polarization type

**Description**

Return a polarization type in this scantable.

**Arguments**

**Returns**

string

**Example**
sd.scantable.poly_baseline.html

**sd.scantable.poly_baseline - Function**

5.1.2 Perform a baseline subtraction using polynomial function

**Description**

Return a scan which has been baselined (all rows) by a polynomial. If insitu is True, or insitu is None and sd.rcParams['insitu'] is True, the method will not return the result, but apply operation on this scantable. You can verify and decide whether you apply the fit result or not, if plot argument is True. In that case, you have to answer 'y' or 'n' for each spectra so that setting True is not recommended for large dataset. This is an newer version of old_poly_baseline. The only difference between them is its performance. For larger dataset, it is recommended to use this method.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>An optional mask</td>
<td>bool array</td>
<td>None</td>
</tr>
<tr>
<td>order</td>
<td>The order of the polynomial</td>
<td>integer</td>
<td>0</td>
</tr>
<tr>
<td>insitu</td>
<td>If False a new scantable is returned</td>
<td>bool</td>
<td>False (use default value)</td>
</tr>
<tr>
<td>plot</td>
<td>Plot the fit and the residual</td>
<td>bool</td>
<td>False</td>
</tr>
<tr>
<td>getresidual</td>
<td>If False, return best-fit value instead of residual</td>
<td>bool</td>
<td>True</td>
</tr>
<tr>
<td>showprogress</td>
<td>Show progress status for large data</td>
<td>bool</td>
<td>True</td>
</tr>
<tr>
<td>minnrow</td>
<td>Minimum number of spectra to show progress status</td>
<td>integer</td>
<td>1000</td>
</tr>
<tr>
<td>outlog</td>
<td>Output the coefficients of the best-fit function to logger</td>
<td>bool</td>
<td>False</td>
</tr>
<tr>
<td>blfile</td>
<td>Name of text file in which the best-fit parameter values to be written</td>
<td>string</td>
<td>&quot;&quot;</td>
</tr>
</tbody>
</table>

| Returns         | scantable                                        |                          |             |

| Example         |                                                 |                          |             |

```python
scan = sd.scantable('OrionS_rawACSmod_cal')
# return a scan baselined by a third order polynomial,
# not using a mask
bscan = scan.poly_baseline(order=3)
```
5.1.2 Recalculate azimuth and elevation for each sky position

Description
Recalculate the azimuth and elevation for each position.
Note that the operation will always be applied to this scantable regardless of the value of sd.rcParams[‘insitu’].

Arguments

Returns

Example
sd.scantable.resample.html

sd.scantable.resample - Function

5.1.2 Perform a binning

Description

Return a scan where all spectra have been binned up.
The method argument specifies interpolation method when correcting from a table. Values are 'nearest', 'linear', 'cubic' (default), and 'spline'.
If insitu is True, or insitu is None and sd.rcParams['insitu'] is True, the method will not return the result, but apply operation on this scantable.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>The bin width in pixels</td>
<td>integer</td>
<td>5</td>
</tr>
<tr>
<td>method</td>
<td>Interpolation method when correcting from a table</td>
<td>string</td>
<td>'cubic'</td>
</tr>
<tr>
<td>insitu</td>
<td>If False a new scantable is returned</td>
<td>bool</td>
<td>None</td>
</tr>
</tbody>
</table>

Returns

scantable

Example
sd.scantable.rotate_linpolphase.html

sd.scantable.rotate_linpolphase - Function

5.1.2 Rotate a phase of the complex polarization

Description

Rotate the phase of the complex polarization O=Q+iU correlation. This is always done in situ in the raw data. So if you call this function more than once then each call rotates the phase further.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>angle</td>
<td>The angle (in degree) to rotate (add) by</td>
</tr>
<tr>
<td>allowed:</td>
<td>float</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

Returns

Example

```python
scan = sd.scantable('OrionS_rawACSmod_cal')
scan.rotate_linpolphase(2.3)
```
sd.scantable.rotate_xyphase - Function

5.1.2 Rotate a phase of the XY correlation

Description

Rotate the phase of the XY correlation. This is always done in situ in the data. So if you call this function more than once then each call rotates the phase further.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>The angle (in degree) to rotate (add) by</th>
</tr>
</thead>
<tbody>
<tr>
<td>angle</td>
<td>allowed: float</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

Example

```python
scan = sd.scantable('OrionS_rawACSmod_cal')
scan.rotate_xyphase(2.3)
```
sd.scantable.save.html

sd.scantable.save - Function

5.1.2 Store the scantable on disk

Description

Store the scantable on disk. This can be an asap (CASA Table), SDFITS or MS2 format.
The output filename can be specified as name. For format 'ASCII', this is the root file name (data in 'name'.txt and header in 'name'.header.txt). If no name is given, the default name will be used. The format is optional file format. Default is 'ASAP'. Allows are:

- 'ASAP' (save as ASAP [CASA Table])
- 'SDFITS' (save as SDFITS file)
- 'ASCII' (saves as ascii text file)
- 'MS2' (saves as an casacore MeasurementSet V2)
- 'FITS' (save as image FITS - not readable by class)
- 'CLASS' (save as FITS readable by CLASS)

If overwrite argument is True, the file should be overwritten if it exists. The default False is to return with warning without writing the output. USE WITH CARE.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the output file</td>
</tr>
<tr>
<td>format</td>
<td>An optional file format</td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite existing file</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>allowed</th>
<th>type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>None</td>
</tr>
<tr>
<td>format</td>
<td>string</td>
<td>None</td>
</tr>
<tr>
<td>overwrite</td>
<td>bool</td>
<td>False</td>
</tr>
</tbody>
</table>

Returns

2018
Example

```python
scan = sd.scantable('OrionS_rawACSmod_cal')
scan.save('myscan.asap')
scan.save('myscan.sdfits', 'SDFITS')
```
**sd.scantable.scale** - Function

5.1.2 Scale spectra by the given factor

**Description**

Return a scan where all spectra are scaled by the given 'factor'. The factor can be a float or one- or two-dimensional float array. If it is one-dimensional array, the scaling will be done in channel-by-channel manner. If it is two-dimensional, the scaling will be done in element-by-element or row-by-row manner. If insitu is True, or insitu is None and sd.rcParams['insitu'] is True, the method will not return the result, but apply operation on this scantable.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>factor</td>
<td>The scaling factor</td>
</tr>
<tr>
<td>allowed:</td>
<td>float, float array</td>
</tr>
<tr>
<td>Default:</td>
<td>None</td>
</tr>
<tr>
<td>insitu</td>
<td>If False a new scantable is returned</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>None</td>
</tr>
<tr>
<td>tsys</td>
<td>If True then apply the operation to Tsys as well as the data</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>True</td>
</tr>
</tbody>
</table>

**Returns**

scantable

**Example**
sd.scantable.set_dirframe.html

sd.scantable.set_dirframe - Function

5.1.2 Set the frame type of the direction on the sky

Description

Set the frame type of the Direction on the sky. The valid frames are 'J2000', 'B1950', 'GALACTIC'.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>An optional frame type</th>
</tr>
</thead>
<tbody>
<tr>
<td>frame</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: &quot;</td>
</tr>
</tbody>
</table>

Returns

Example

```python
scan = sd.scantable('OrionS_rawACSmod_cal')
scan.set_dirframe('GALACTIC')
```
sd.scantable.set_doppler.html

sd.scantable.set_doppler - Function

5.1.2 Set definition of the Doppler correction

Description

Set the doppler for all following operations on this scantable.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>doppler</th>
<th>One of 'RADIO', 'OPTICAL', 'Z', 'BETA', 'GAMMA'</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>'RADIO'</td>
<td></td>
</tr>
</tbody>
</table>

Returns

Example
sd.scantable.set_feedtype.html

sd.scantable.set_feedtype - Function

5.1.2 Set the feed type

Description

Overwrite the feed type, which might not be set correctly.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>feedtype</td>
<td>'linear' or 'circular'</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

Returns

Example
sd.scantable.set_fluxunit.html

sd.scantable.set_fluxunit - Function

5.1.2 Set flux unit

Description

Set flux unit to this scantable. Valid values are 'K' and 'Jy'.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>'K' or 'Jy'</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

Returns

Example
5.1.2 Set the frame type of the spectral axis

**Description**

Set the frame type of the Spectral Axis. Valid frames are 'TOPO', 'LSRD', 'LSRK', 'BARY', 'GEO', 'GALACTO', 'LGROUP', 'CMB'. The default is taken from sd.rcParams[‘scantable.freqframe’].

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>frame</td>
<td>An optional frame type</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>None</td>
</tr>
</tbody>
</table>

**Returns**

**Example**

```python
scan = sd.scantable('OrionS_rawACSmod_cal')
scan.set_freqframe('BARY')
```
sd.scantable.set_instrument.html

**sd.scantable.set_instrument** - Function

### 5.1.2 Set antenna name

**Description**

Set the instrument (antenna name) for subsequent processing.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Instrument (or antenna) name</th>
</tr>
</thead>
<tbody>
<tr>
<td>instr</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

**Example**
sd.scantable.set_restfreqs.html

**sd.scantable.set_restfreqs - Function**

5.1.2 Set rest frequency

**Description**

Set or replace the rest frequency specified and if the 'freqs' argument holds a scalar, then that rest frequency will be applied to all the selected data. If the 'freqs' argument holds a vector, then it MUST be of equal or smaller length than the number of IFs (and the available rest frequencies will be replaced by this vector). In this case, *all* data have the rest frequency set per IF according to the corresponding value you give in the 'freqs' vector. E.g. 'freqs=[1e9, 2e9]' would mean IF 0 gets restfreq 1e9 and IF 1 gets restfreq 2e9.

You can also specify the frequencies via a linecatalog.

Note that, to do more sophisticate Restfrequency setting, e.g. on a source and IF basis, use scantable.set_selection() before using this function:

```python
# provided your scantable is called scan
selection = sd.selector()
selection.set_name("ORION*")
selection.set_ifs([1])
scan.set_selection(selection)
scan.set_restfreqs(freqs=86.6e9)
```

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>freqs</td>
<td>List of rest frequency values or string identifiers</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>None</td>
</tr>
<tr>
<td>unit</td>
<td>Unit for rest frequency</td>
</tr>
<tr>
<td>allowed</td>
<td>string</td>
</tr>
<tr>
<td>Default</td>
<td>'Hz'</td>
</tr>
</tbody>
</table>

**Returns**

**Example**
scan = sd.scantable('OrionS_rawACSmod_cal')
# set the given restfrequency for the all currently selected IFs
scan.set_restfrefs(freqs=1.4e9)
# set restfrequencies for the n IFs (n > 1) in the order of the
# list, i.e
# IF0 -> 1.4e9, IF1 -> 1.41e9, IF3 -> 1.42e9
# len(list_of_restfreqs) == nIF
# for nIF == 1 the following will set multiple restfrequency for
# that IF
scan.set_restfreqs(freqs=[1.4e9, 1.41e9, 1.42e9])
# set multiple restfrequencies per IF. as a list of lists where
# the outer list has nIF elements, the inner s arbitrary
scan.set_restfreqs(freqs=[[1.4e9, 1.41e9], [1.67e9]])
5.1.2 Select a subset of the data

Description

Select a subset of the data. All following operations on this scantable are only applied to the selection.

The selection can be done via a selector object (default unset the selection), or any combination of "pols", "ifs", "beams", "scans", "cycles", "name", "query", "types" that will be passed to the constructor of the selector object to create a new selection.

Arguments

Inputs

<table>
<thead>
<tr>
<th>selection</th>
<th>A selector object</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowed:</td>
<td>selector</td>
</tr>
<tr>
<td>Default:</td>
<td>None</td>
</tr>
</tbody>
</table>

Returns

Example

```python
scan = sd.scantable('OrionS_rawACSmod_cal')
sel = sd.selector()  # create a selection object
self.set_scans([0, 3])  # select SCANNO 0 and 3
scan.set_selection(sel)  # set the selection
scan.summary()  # will only print summary of scanno 0 an 3
scan.set_selection()  # unset the selection
# or the equivalent
scan.set_selection(scans=[0,3])
scan.summary()  # will only print summary of scanno 0 an 3
scan.set_selection()  # unset the selection
```
5.1.2 Set the types of source to be source or reference scan

Description

Set the type of the source to be an source or reference scan using the provided pattern.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>match</td>
<td>A Unix style pattern, regular expression or selector</td>
</tr>
<tr>
<td></td>
<td>allowed: string, selector</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>matchtype</td>
<td>'pattern' for UNIX style pattern, 'regex' for regular expression</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: 'pattern'</td>
</tr>
<tr>
<td>sourcetype</td>
<td>The type of the source to use (source/reference)</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: 'reference'</td>
</tr>
</tbody>
</table>

Returns

Example

2030
sd.scantable.set_spectrum.html

**sd.scantable.set_spectrum - Function**

**5.1.2** Set spectrum for specified row

**Description**

Set the spectrum for the current row in the scantable.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>spec</td>
<td>The new spectrum</td>
<td>float array</td>
<td></td>
</tr>
<tr>
<td>rowno</td>
<td>The row number to set the spectrum for</td>
<td>integer</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

**Example**

2031
sd.scantable.set_unit.html

sd.scantable.set_unit - Function

5.1.2 Set unit for spectral axis

Description

Set the unit for all following operations on this scantable. Valid options are 'Hz', 'km/s', 'channel', or 'none'.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>unit</td>
<td>Optional unit</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>'channel'</td>
</tr>
</tbody>
</table>

Returns

Example
5.1.2 Shift the reference pixel of the spectral coordinate

**Description**

Shift the reference pixel of the Spectra Coordinate by an integer amount. Be careful using this with broadband data.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delta</td>
<td>The amount to shift by</td>
</tr>
<tr>
<td></td>
<td>allowed: integer</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

**Example**
5.1.2 Perform a baseline subtraction using sinusoidal function

Description

Return a scan which has been baselined (all rows) with sinusoidal functions. Fit will be done with 'sigma-clipping'. Spectral lines are detected first using linefinder and masked out to avoid them affecting the baseline solution. If applyfft is set to True, the function performs Fourier analysis to select wave numbers for sinusoidal fitting. Currently, 'fft' is only available to be used for the analysis. You can specify threshold for selection of wave number using fftthresh parameter. Both float and string is acceptable. Given a float value, the unit is set to sigma. For string values, allowed formats include:

- any decimal number plus 'sigma' (e.g. '3sigma')
- 'top' plus any decimal number (e.g. 'top10')

In addition, you can add or reject specific wave numbers from the fit using addwn and rejwn, respectively. You can specify wave numbers as an integer, string, or list of them. For string specification, syntax for those parameters are as follows:

- 'a-b' (= a, a+1, a+2, ..., b-1, b)
- '<a' (= 0, 1, ..., a-2, a-1)
- '>a' (= a+1, a+2, ... up to maximum wave number corresponding to the Nyquist frequency)

You can append '=' after inequality sign. When both addwn and rejwn are set, rejwn will take priority of addwn.

Note: The best-fit parameter values output in logger and/or blfile are now based on specunit of 'channel'.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>insitu</td>
<td>If False a new scantable is returned</td>
<td>bool</td>
<td>None (use default value)</td>
</tr>
<tr>
<td>mask</td>
<td>An optional mask retrieved from scantable</td>
<td>bool array</td>
<td>None (no mask)</td>
</tr>
<tr>
<td>applyfft</td>
<td>Perform Fourier analysis to find appropriate sinusoidal component</td>
<td>bool</td>
<td>True</td>
</tr>
<tr>
<td>fftmethod</td>
<td>Method to find sinusoidal component (currently only 'fft' is available)</td>
<td>string</td>
<td>'fft'</td>
</tr>
<tr>
<td>fftthresh</td>
<td>Threshold to select wave number in Fourier analysis</td>
<td>float, string</td>
<td>3.0</td>
</tr>
<tr>
<td>addwn</td>
<td>Additional wave numbers to be used for fitting</td>
<td>integer, string, any array</td>
<td>[]</td>
</tr>
<tr>
<td>rejwn</td>
<td>Wave numbers not to be used for fitting</td>
<td>integer, string, any array</td>
<td>[]</td>
</tr>
<tr>
<td>clipthresh</td>
<td>Clipping threshold in unit of sigma</td>
<td>float</td>
<td>3.0</td>
</tr>
<tr>
<td>clipniter</td>
<td>Maximum number of iteration of clipping</td>
<td>integer</td>
<td>0</td>
</tr>
<tr>
<td>plot</td>
<td>Plot the fit and the residual (currently unavailable)</td>
<td>bool</td>
<td>False</td>
</tr>
<tr>
<td>getresidual</td>
<td>If False, return best-fit value instead of residual</td>
<td>bool</td>
<td>True</td>
</tr>
<tr>
<td>showprogress</td>
<td>Show progress status for large data</td>
<td>bool</td>
<td>True</td>
</tr>
<tr>
<td>minnrow</td>
<td>Minimum number of spectra to show progress status</td>
<td>integer</td>
<td>1000</td>
</tr>
<tr>
<td>outlog</td>
<td>Output the coefficients of the best-fit function to logger</td>
<td>bool</td>
<td>False</td>
</tr>
<tr>
<td>blfile</td>
<td>Name of text file in which the best-fit parameter values to be written</td>
<td>string</td>
<td>2035</td>
</tr>
</tbody>
</table>
Example

```
scan = sd.scantable('OrionS_rawACSmod_cal',average=False)
# return a scan baselined by a combination of sinusoidal curves having
# wave numbers in spectral window up to 10,
# also with 3-sigma clipping, iteration up to 4 times
bscan = scan.sinusoid_baseline(addwn='<=10',clipthresh=3.0,clipniter=4)
```
Smooth the spectrum by the specified kernel (conserving flux).

The user should specify a type of smoothing kernel. Supported types are 'hanning' (default), 'gaussian', 'boxcar', 'rmedian', or 'poly'. The width of the kernel in pixel can be specified as width argument. For hanning this is ignored otherwise it defaults to 5 pixels. For 'gaussian' it is the Full Width Half Maximum. For 'boxcar' it is the full width. For 'rmedian' and 'poly' it is the half width. The order argument is an optional parameter for 'poly' kernel (default is 2), to specify the order of the polynomial. It is ignored by all other kernels.

You can verify and decide whether you apply the fit result or not, if plot argument is True. In that case, you have to answer 'y' or 'n' for each spectra so that setting True is not recommended for large dataset.

If insitu is True, or insitu is None and sd.rcParams['insitu'] is True, the method will not return the result, but apply operation on this scantable.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel</td>
<td>The type of smoothing kernel</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>'hanning'</td>
</tr>
<tr>
<td>width</td>
<td>The width of the kernel in pixels</td>
</tr>
<tr>
<td>allowed:</td>
<td>float</td>
</tr>
<tr>
<td>Default:</td>
<td>5.0</td>
</tr>
<tr>
<td>order</td>
<td>Order of the polynomial for 'poly' kernel</td>
</tr>
<tr>
<td>allowed:</td>
<td>integer</td>
</tr>
<tr>
<td>Default:</td>
<td>2</td>
</tr>
<tr>
<td>plot</td>
<td>Plot the fit and the residual</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>False</td>
</tr>
<tr>
<td>insitu</td>
<td>If False a new scantable is returned</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>None</td>
</tr>
</tbody>
</table>

Returns

scantable
Example
sd.scantable.stats.html

sd.scantable.stats - Function

5.1.2 Compute specified statistics of the spectra

Description

Determine the specified statistic of the current beam/if/pol Takes a 'mask' as an optional parameter to specify which channels should be excluded. Available statistics are 'min', 'max', 'min_abc', 'max_abc', 'sumsq', 'sum', 'mean', 'var', 'stddev', 'avdev', 'rms', 'median'.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat</td>
<td>Statistics to be calculated</td>
<td>string</td>
<td>'stddev'</td>
</tr>
<tr>
<td>mask</td>
<td>An optional mask specifying where the statistic should be determined</td>
<td>bool array</td>
<td>None</td>
</tr>
<tr>
<td>form</td>
<td>Format string to print statistic values</td>
<td>string</td>
<td>'3.3f'</td>
</tr>
<tr>
<td>row</td>
<td>Row number of spectrum to process</td>
<td>integer</td>
<td>None (all rows)</td>
</tr>
</tbody>
</table>

Returns

float array

Example

```python
scan = sd.scantable('OrionS_rawACSmod_cal')
scan.set_unit('channel')
msk = scan.create_mask([[100, 200], [500, 600]])
scan.stats(stat='mean', mask=msk)
```
**sd.scantable.stddev**

### sd.scantable.stddev - Function

**5.1.2 Compute standard deviation of the spectra**

**Description**

Determine the standard deviation of the current beam/if/pol Takes a ‘mask’ as an optional parameter to specify which channels should be excluded.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>An optional mask specifying where the standard deviation should be determined</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool array</td>
</tr>
<tr>
<td>Default:</td>
<td>None</td>
</tr>
</tbody>
</table>

**Returns**

float array

**Example**

```python
scan = sd.scantable('OrionS_rawACSmod_cal')
scan.set_unit('channel')
msk = scan.create_mask([100, 200], [500, 600])
scan.stddev(mask=msk)
```

2040
sd.scantable.summary.html

**sd.scantable.summary** - Function

5.1.2 Print a summary of the contents of the scantable

**Description**

Print a summary of the contents of this scantable. Optionally, output to the file specified.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>The name of a file to write the output</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: None (no file output)</td>
</tr>
</tbody>
</table>

**Returns**

string

**Example**
sd.scantable.swap_linears.html

sd.scantable.swap_linears - Function

5.1.2 Swap the linear polarizations XX and YY

Description

Swap the linear polarisations XX and YY, or better the first two polarisations as this also works for circulars.

Arguments

Returns

Example
5.1.3  sd.selector - Tool
Data selection tool for single-dish data

Description

The selector is a data selection object that affects a scantable object via set_selection function. Selection by the following attributes are available:

- scan number
- beam number
- cycle number
- IF number
- source name (allows regular expression)
- polarization number
- row number
- source type
- value of system temperature (minimum and maximum threshold)

Note that the selection by an integer is 0-based (scan, beam, cycle, IF, polarization, and row).

In addition, the selector provides interface for more flexible data selection using TaQL.

The selector also support sorting data. The selector sorts data based on values of columns specified by the user.

Summary of the current selection is stored in the object as a string. Thus, you can see that summary by using print (see example below).

The constructor takes some arguments to set selection criteria. So, the user can set selection either using constructor options and available setter functions.

Source types should be given as an integer or an enumerations that are properly defined for source type. That enumeration can be accessed via sd.srctype. The list of enumerations defined (following sd.srctype.) are as follows:

```
enum    int    description
```

2043
<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pson</td>
<td>0</td>
<td>on-source scan of position switched data</td>
</tr>
<tr>
<td>psoff</td>
<td>1</td>
<td>off-source scan of position switched data</td>
</tr>
<tr>
<td>nod</td>
<td>2</td>
<td>nod scan</td>
</tr>
<tr>
<td>fson</td>
<td>3</td>
<td>on-source scan of frequency switched data</td>
</tr>
<tr>
<td>fsoff</td>
<td>4</td>
<td>reference scan of frequency switched data</td>
</tr>
<tr>
<td>sky</td>
<td>6</td>
<td>sky scan for calibration</td>
</tr>
<tr>
<td>hot</td>
<td>7</td>
<td>hot load scan for calibration</td>
</tr>
<tr>
<td>warm</td>
<td>8</td>
<td>warm load scan for calibration</td>
</tr>
<tr>
<td>cold</td>
<td>9</td>
<td>cold load scan for calibration</td>
</tr>
<tr>
<td>poncal</td>
<td>10</td>
<td>on-source calibration scan of position switched data</td>
</tr>
<tr>
<td>poffcal</td>
<td>11</td>
<td>off-source calibration scan of position switched data</td>
</tr>
<tr>
<td>nodcal</td>
<td>12</td>
<td>nod calibration scan</td>
</tr>
<tr>
<td>foncal</td>
<td>13</td>
<td>on-source calibration scan of frequency switched data</td>
</tr>
<tr>
<td>foffcal</td>
<td>14</td>
<td>reference calibration scan of frequency switched data</td>
</tr>
<tr>
<td>fslo</td>
<td>20</td>
<td>lower frequency throw of symmetric frequency switching</td>
</tr>
<tr>
<td>flooff</td>
<td>21</td>
<td>off-source lower frequency throw of symmetric frequency switching</td>
</tr>
<tr>
<td>flosky</td>
<td>26</td>
<td>sky lower frequency throw of symmetric frequency switching</td>
</tr>
<tr>
<td>flohot</td>
<td>27</td>
<td>hot load lower frequency throw of symmetric frequency switching</td>
</tr>
<tr>
<td>flowarm</td>
<td>28</td>
<td>warm load lower frequency throw of symmetric frequency switching</td>
</tr>
<tr>
<td>flocold</td>
<td>29</td>
<td>cold load lower frequency throw of symmetric frequency switching</td>
</tr>
<tr>
<td>fshi</td>
<td>30</td>
<td>higher frequency throw of symmetric frequency switching</td>
</tr>
<tr>
<td>fhioff</td>
<td>31</td>
<td>off-source higher frequency throw of symmetric frequency switching</td>
</tr>
<tr>
<td>fhiisky</td>
<td>36</td>
<td>sky higher frequency throw of symmetric frequency switching</td>
</tr>
<tr>
<td>fhihot</td>
<td>37</td>
<td>hot load higher frequency throw of symmetric frequency switching</td>
</tr>
<tr>
<td>fhiwarm</td>
<td>38</td>
<td>warm load higher frequency throw of symmetric frequency switching</td>
</tr>
<tr>
<td>fhicold</td>
<td>39</td>
<td>cold load higher frequency throw of symmetric frequency switching</td>
</tr>
<tr>
<td>sig</td>
<td>90</td>
<td>signal scan</td>
</tr>
<tr>
<td>ref</td>
<td>91</td>
<td>reference scan</td>
</tr>
<tr>
<td>cal</td>
<td>92</td>
<td>calibration scan</td>
</tr>
<tr>
<td>notype</td>
<td>99</td>
<td>no type</td>
</tr>
</tbody>
</table>

**Arguments**

2044
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>beams</td>
<td>Beam number selection</td>
<td>integer, integer array</td>
<td>None (no selection)</td>
</tr>
<tr>
<td>cycles</td>
<td>Cycle number selection</td>
<td>integer, integer array</td>
<td>None (no selection)</td>
</tr>
<tr>
<td>ifs</td>
<td>IF number selection</td>
<td>integer, integer array</td>
<td>None (no selection)</td>
</tr>
<tr>
<td>name</td>
<td>Source name selection. Allows regular expression.</td>
<td>string</td>
<td>None (no selection)</td>
</tr>
<tr>
<td>pols</td>
<td>Polarization number selection</td>
<td>integer, integer array, string, string array</td>
<td>None (no selection)</td>
</tr>
<tr>
<td>query</td>
<td>TaQL selection query</td>
<td>string</td>
<td>None (no selection)</td>
</tr>
<tr>
<td>scans</td>
<td>Scan number selection</td>
<td>integer, integer array</td>
<td>None (no selection)</td>
</tr>
<tr>
<td>types</td>
<td>Source type selection</td>
<td>integer, integer array</td>
<td>None (no selection)</td>
</tr>
<tr>
<td>rows</td>
<td>Row number selection</td>
<td>integer, integer array</td>
<td>None (no selection)</td>
</tr>
</tbody>
</table>

**Example**

```python
# create a selector object
sel=sd.selector()
# These are equivalent if data is 'linear'
sel.set_polarisations(['XX','Re(XY)'])
sel.set_polarisations([0,2])
# reset the polarisation selection
sel.set_polarisations()
# these are equivalent
sel2=sd.selector(ifs=[0,1])
sel2=sd.selector()
sel2.set_ifs([0,1])
# check current selection
```

2045
print sel
IFNO: [0,1]
# apply selection on scantable
s=sd.scantable('OrionS_rawACSmod_cal',average=False)
s.set_selection(sel)

Methods

- **get_beams**: Return a list of current beam selection
- **get_cycles**: Return a list of current cycle selection
- **get_ifs**: Return a list of current IF selection
- **get_name**: Return a selection string for source name (not yet implemented)
- **get_order**: Return a list of columns used for sorting
- **get_pols**: Return a list of current polarization selection
- **get_poltypes**: Return a list of polarization types for current polarization selection
- **get_query**: Return current selection string for TaQL selection
- **get_rows**: Return a list of current row number selection
- **get_scans**: Return a list of current scan number selection
- **get_types**: Return a list of current source type selection
- **is_empty**: Check if any selection has been set
- **reset**: Unset all selections
- **set_beams**: Set a sequence of beam numbers
- **set_cycles**: Set a sequence of cycle numbers
- **set_ifs**: Set a sequence of IF numbers
- **set_name**: Set a selection based on a source name
- **set_order**: Set column names that is used for sorting data
- **set_polarisations**: Set a sequence of polarization numbers or strings
- **set_polarizations**: Set a sequence of polarization numbers or strings
- **set_pols**: Set a sequence of polarization numbers or strings
- **set_query**: Set a selection string based on TaQL
- **set_rows**: Set a sequence of row numbers
- **set_scans**: Set a sequence of scan numbers
- **set_tsys**: Set range of system temperature
- **set_types**: Set a sequence of source types
sd.selector.get_beams.html

sd.selector.get_beams - Function

5.1.3 Return a list of current beam selection

Description

Return a list of current beam selection.

Arguments

Returns

integer array

Example
sd.selector.get_cycles.html

sd.selector.get_cycles - Function

5.1.3 Return a list of current cycle selection

Description

Return a list of current cycle selection.

Arguments

Returns

integer array

Example
sd.selector.get_ifs.html

sd.selector.get_ifs - Function

5.1.3 Return a list of current IF selection

Description

Return a list of current IF selection.

Arguments

Returns

integer array

Example
sd.selector.get_name.html

sd.selector.get_name - Function

5.1.3 Return a selection string for source name (not yet implemented)

Description

Return a selection strings for source name. Not yet implemented.

Arguments

Returns

None

Example
sd.selector.get_order.html

sd.selector.get_order - Function

5.1.3 Return a list of columns used for sorting

Description

Return a list of columns used for sorting.

Arguments

Returns

string list

Example
sd.selector.get_pols.html

**sd.selector.get_pols - Function**

5.1.3 Return a list of current polarization selection

**Description**

Return a list of current polarization selection. It returns an integer array even if you set polarization selection by string (e.g. 'XX', 'I').

**Arguments**

**Returns**

integer array

**Example**
sd.selector.get_poltypes.html

**sd.selector.get_poltypes - Function**

5.1.3 Return a list of polarization types for current polarization selection

**Description**

Return a list of polarization types for current polarization selection. The returned array contains polarization types for each polarization selection strings. If you set polarization selection as integer, empty array will be returned since integer selection doesn’t specify polarization type.

**Arguments**

**Returns**

string array

**Example**
sd.selector.get_query.html

sd.selector.get_query - Function

5.1.3 Return current selection string for TaQL selection

Description

Return current selection string for TaQL selection.

Arguments

Returns

string

Example
sd.selector.get_rows.html

sd.selector.get_rows - Function

5.1.3 Return a list of current row number selection

Description
Return a list of current row number selection.

Arguments

Returns
integer array

Example
sd.selector.get_scans.html

**sd.selector.get_scans** - **Function**

5.1.3 Return a list of current scan number selection

**Description**

Return a list of current scan number selection.

**Arguments**

**Returns**

integer array

**Example**
sd.selector.get_types.html

**sd.selector.get_types** - Function

5.1.3 Return a list of current source type selection

**Description**

Return a list of current source type selection.

**Arguments**

**Returns**

type array

**Example**
5.1.3 Check if any selection has been set

**Description**

Check if any selection has been set.
If any selection is set, the method returns False. Otherwise, it returns True.

**Arguments**

**Returns**

`bool`

**Example**

```python
# create selector without any selection
sel = sd.selector()
# is_empty() returns True
sel.is_empty()
True
# set any selection
sel.set_ifs([0])
# then, is_empty() returns False
sel.is_empty()
False
```
**sd.selector.reset - Function**

5.1.3 Unset all selections

**Description**

Unset all selections. Note that the method doesn’t clear sorting order set by `set_order`. To unset sort order, you have to execute `set_order([])`.

**Arguments**

**Returns**

**Example**

```python
# create selector without any selection
sel=sd.selector()
# is_empty() returns True
sel.is_empty()
True
# set any selection
sel.set_ifs([0])
# then, is_empty() returns False
sel.is_empty()
False
# reset() clear all selections
sel.reset()
# thus, is_empty() returns True
sel.is_empty()
True
```
sd.selector.set_beams.html

**sd.selector.set_beams - Function**

5.1.3 Set a sequence of beam numbers

**Description**

Set a sequence of beam numbers to the selection.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Beam number selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>beams</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed: integer, integer array</td>
</tr>
<tr>
<td></td>
<td>Default: [] (no selection)</td>
</tr>
</tbody>
</table>

**Returns**

**Example**
sd.selector.set_cycles.html

sd.selector.set_cycles - Function

5.1.3 Set a sequence of cycle numbers

Description

Set a sequence of cycle numbers to the selection.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cycles</td>
<td>Cycle number selection</td>
</tr>
<tr>
<td></td>
<td>allowed: integer, integer array</td>
</tr>
<tr>
<td></td>
<td>Default: [] (no selection)</td>
</tr>
</tbody>
</table>

Returns

Example
sd.selector.set_ifs.html

sd.selector.set_ifs - Function

5.1.3 Set a sequence of IF numbers

Description

Set a sequence of IF numbers to the selection.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>IF number selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>ifs</td>
<td></td>
</tr>
</tbody>
</table>

allowed: integer, integer array
Default: [] (no selection)

Returns

Example
sd.selector.set_name.html

**sd.selector.set_name** - Function

5.1.3 Set a selection based on a source name

**Description**

Set a selection based on a source name. This can be a unix pattern, e.g. 
"*_R".

Note that the method overwrites TaQL selection string set by `set_query`.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>A string containing a source name or pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

**Example**

```python
sel=sd.selector()
# select all reference scans which start with "Orion"
sel.set_name("Orion*_R")
```
sd.selector.set_order.html

sd.selector.set_order - Function

5.1.3 Set column names that is used for sorting data

Description

Set the order the scantable should be sorted by. The user specify a list of column names that is used for sorting data (e.g. 'IFNO', 'SCANNO'). The data are sorted according to the value of these columns.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>order</td>
<td>The list of column names to sort by in order</td>
</tr>
<tr>
<td>allowed:</td>
<td>string array</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

Returns

Example
sd.selector.set_polarisations.html

**sd.selector.set_polarisations - Function**

5.1.3 Set a sequence of polarization numbers or strings

**Description**

Set the polarisations to be selected in the scantable. It allows to set polarization selection via integer or string (e.g. "XX", "Q"). Integer must be within the range of 0 to 3 since the number outside this range will result no selected data exception. If the selection is specified by an integer, the selection will refer polarization type of the original scantable. On the other hand, the selection by string contains both type and component so that the selection will refer its type, not the original scantable.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>A list of integers of 0-3, or strings, e.g. [&quot;I&quot;,&quot;Q&quot;]. allowed: integer, string, integer array, string array Default: [] (no selection)</th>
</tr>
</thead>
</table>

**Example**

```python
sel = sd.selector()
# These are equivalent if data is 'linear'
sel.set_polarisations(["XX","Re(XY)"])
sel.set_polarisations([0,2])
# reset the polarisation selection
sel.set_polarisations()
```
5.1.3 Set a sequence of polarization numbers or strings

Description

See `set_polarisations`.

Arguments

Returns

Example
sd.selector.set_pols.html

**sd.selector.set_pols - Function**

5.1.3 Set a sequence of polarization numbers or strings

**Description**

See [set_polarisations](#).

**Arguments**

**Returns**

**Example**
sd.selector.set_query.html

sd.selector.set_query - Function

5.1.3 Set a selection string based on TaQL

Description

Select by Column query. Power users only!
The method is used to set TaQL selection string directly. The user must be familiar with TaQL and data structure of scantable.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>TaQL selection string</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Returns

Example

```python
sel=sd.selector()
# select all off scans with integration times over 60 seconds.
sel.set_query("SRCTYPE == PSOFF AND INTERVAL > 60.0")
```
5.1.3 Set a sequence of row numbers

Description

Set a sequence of row numbers (0-based). Power users Only!
NOTICE row numbers can be changed easily by sorting, prior selection, etc.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rows</td>
<td>A list of integers</td>
</tr>
<tr>
<td></td>
<td>allowed: integer, integer array</td>
</tr>
<tr>
<td></td>
<td>Default: [] (no selection)</td>
</tr>
</tbody>
</table>

Returns

Example
sd.selector.set_scans.html

sd.selector.set_scans - Function

5.1.3 Set a sequence of scan numbers

Description

Set a sequence of Scan numbers to the selection.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scans</td>
<td>A list of integers</td>
</tr>
<tr>
<td>allowed:</td>
<td>integer, integer array</td>
</tr>
<tr>
<td>Default:</td>
<td>[] (no selection)</td>
</tr>
</tbody>
</table>

Returns

Example
5.1.3 Set range of system temperature

Description

Select by Tsys range. The method sets an upper and a lower limit of the Tsys value.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>tsysmin</td>
<td>The lower threshold</td>
<td>float</td>
<td>0.0</td>
</tr>
<tr>
<td>tsysmax</td>
<td>The upper threshold</td>
<td>float</td>
<td>None (no upper limit)</td>
</tr>
</tbody>
</table>

Returns

Example

```python
sel=sd.selector()
# select all spectra with Tsys <= 500.0
sel.set_tsys(tsysmax=500.0)
```
sd.selector.set_types.html

**sd.selector.set_types - Function**

5.1.3 Set a sequence of source types

**Description**

Set a sequence of source types to the selection. The types argument can contain integer and/or sd.srctype enum.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>A list of integers</th>
</tr>
</thead>
<tbody>
<tr>
<td>types</td>
<td>integer, integer array</td>
</tr>
<tr>
<td></td>
<td>Default: [] (no selection)</td>
</tr>
</tbody>
</table>

**Example**

```python
sel=sd.selector()
# select only on-source scans of position switched observation
sel.set_types(0)
# or use enum
sel.set_types(sd.srcType.pson)
```
5.1.4 sd.fitter - Tool

General fitting tool

Description

The fitter is an object to fit data. This contains both baseline fitting and spectral line fitting. For baseline fitting, simple polynomial fitting with arbitrary order is available. For spectral line fitting, Gaussian and Lorentzian fitting are supported. It allows fitting of multiple lines, but it doesn’t support hyperfine fitting, i.e. multiple line fitting under some constraints. The constructor doesn’t take any arguments. It just creates a fitter object that no state is set.

Example

```python
s = sd.scantable('myscan.asap')
s.set_cursor(thepol=1) # select second pol
f = sd.fitter() # create fitter object
f.set_scan(s)
f.set_function(poly=0)
f.fit(row=0) # fit first row
```

Methods

- `auto_fit`: Return a scan where the function is applied to all rows
- `commit`: Return a new scan where the fits have been subtracted
- `fit`: Execute actual fitting process
- `get_area`: Return the area under the fitted gaussian/lorentzian component
- `get_chi2`: Return \( \chi^2 \) value
- `get_errors`: Return the errors in parameters
- `get_estimate`: Return the parameter estimates
- `get_fit`: Return the fitted ordinate values
- `get_parameters`: Return the fit parameters
- `get_residual`: Return the residual of the fit
- `plot`: Plot fit
- `set_data`: Set the abcissa and ordinate for the fit
- `set_function`: Set the function to be fitted
- `set_gauss_parameters`: Set the parameters of Gaussian component
- `set_lorentz_parameters`: Set the parameters of Lorentzian component
- `set_parameters`: Set the parameters to be fitted
- `set_scan`: Set the data as a scantable
set_sinusoid_parameters
store_fit

Set the parameters of Sinusoidal component
Save fit parameters
sd.fitter.auto_fit.html

**sd.fitter.auto_fit - Function**

5.1.4 Return a scan where the function is applied to all rows

**Description**

This method executes fitting for all rows in the current selection of the data. It returns a scan where the function is applied to all rows for all Beams/IFs/Pols. If any data is not set and/or fitting function is not specified, the function throws an exception.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>insitu</td>
<td>Apply operation on the input scantable or not</td>
<td>bool</td>
<td>None (use default setting defined in sd.rcParams)</td>
</tr>
<tr>
<td>plot</td>
<td>Plot and verify result or not</td>
<td>bool</td>
<td>False</td>
</tr>
</tbody>
</table>

**Returns**

scantable

**Example**

```python
f=sd.fitter()
s=sd.scantable('OrionS_rawACSmod_cal',average=False)
f.set_scan(s)
f.set_function(poly=3)
s_bs=f.auto_fit()
```
Description

Return a new scan where the fits have been subtracted. You must fit the data before you execute this method. Otherwise, an exception will be thrown. If the data is set as a set of abcissa and ordinate values using `set_data`, it will not work. If you want to get data by this method, you must use `set_scan` that set the data as a scantable.

Arguments

Returns

scantable

Example

```python
f = sd.fitter()
s = sd.scantable('OrionS_rawACSmod_cal', average=False)
# you must set data as a scantable
f.set_scan(s)
# set polynomial function of order 3 for baseline subtraction
f.set_function(poly=3)
# do fit
f.fit()
# get baseline subtracted scans
s_bs = f.commit()
```
sd.fitter.fit.html

sd.fitter.fit - Function

5.1.4 Execute actual fitting process

Description

Execute the actual fitting process. All the state (data, selection, and function) has to be set before executing.
The method executes a fitting process for only one row that are specified by a row argument. By default, the first row will be fitted.
The estimate argument determines if the method computes an initial parameter set automatically. This can be used to compute estimates even if fit was called before.

Arguments

| Inputs | | |
|---|---|
| row | Specify the row in the scantable |
| allowed: | integer |
| Default: | 0 |
| estimate | Auto-compute an initial parameter set |
| allowed: | bool |
| Default: | False |

Returns

Example

```python
s = sd.scantable('myscan.asap')
s.set_cursor(thepol=1)  # select second pol (nb. 0-based)
f = sd.fitter()
f.set_scan(s)
f.set_function(poly=0)
f.fit(row=0)  # fit first row
```
sd.fitter.get_area.html

sd.fitter.get_area - Function

5.1.4 Return the area under the fitted gaussian/lorentzian component

Description

Return the area under the fitted gaussian/lorentzian component. The component argument specifies which component you want to get area. By default, the method will return a sum of all gaussian/lorentzian components. Note that this will only work for gaussian/lorentzian fits. Otherwise, None will be returned.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>The gaussian/lorentzian component selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>component</td>
<td>The gaussian/lorentzian component selection</td>
</tr>
<tr>
<td>allowed:</td>
<td>integer</td>
</tr>
<tr>
<td>Default:</td>
<td>None (sum of all components)</td>
</tr>
</tbody>
</table>

Returns

float

Example

2078
sd.fitter.get_chi2.html

sd.fitter.get_chi2 - Function

5.1.4 Return $\chi^2$ value

Description

Return $\chi^2$ value of the fit. You must execute fitting process before getting $\chi^2$.

Arguments

Returns

float

Example

s = scantable('myscan.asap')
s.set_cursor(thepol=1)  # select second pol (nb. 0-based)
f = fitter()
f.set_scan(s)
f.set_function(poly=0)
f.fit(row=0)  # fit first row
ch2=f.get_chi2()
sd.fitter.get_errors.html

**sd.fitter.get_errors - Function**

5.1.4 Return the errors in parameters

**Description**

Return the errors in the parameters. You must execute fitting process before getting errors.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>component</td>
<td>get the errors for the specified component only</td>
</tr>
<tr>
<td></td>
<td>allowed: integer</td>
</tr>
<tr>
<td></td>
<td>Default: None (all components)</td>
</tr>
</tbody>
</table>

**Returns**

float array

**Example**
Return the parameter estimates for non-linear functions. It works only if fit is executed with estimate=True.

Arguments

Returns
float array

Example
sd.fitter.get_fit.html

sd.fitter.get_fit - Function

5.1.4 Return the fitted ordinate values

Description

Return the fitted ordinate values. For spectral line fitting, it will represent model of the observed line. You must execute fitting process before getting fitted result.

Arguments

Returns

float array

Example
sd.fitter.get_parameters.html

sd.fitter.get_parameters - Function

5.1.4 Return the fit parameters

Description

Return the fit parameters as a dictionary. The returned value contains the following attributes:

- errors
  errors for each parameter

- fixed
  list of fixed parameters

- formatted
  formatted string that shows a fitting result

- params
  list of resulting parameter values

The component argument specifies which component the user want to get fit parameters. It is effective only for gaussian/lorentzian fitting.

The error argument controls contents of formatted string. If error is True, the string contains parameter values and their errors.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th>get the parameter values for the specified component only</th>
</tr>
</thead>
<tbody>
<tr>
<td>component</td>
<td>errors</td>
<td>allowed: integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: None (all components)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If True, include errors of fit parameters in formatted string</td>
</tr>
<tr>
<td></td>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: False</td>
</tr>
</tbody>
</table>

Returns
dictionary

Example

2083
sd.fitter.get_residual.html

**sd.fitter.get_residual** - Function

5.1.4 Return the residual of the fit

**Description**

Return the residual of the fit. For baseline fitting, it will represent a baseline-subtracted spectrum. You must execute fitting process before getting fitted result.

**Arguments**

**Returns**

float array

**Example**
sd.fitter.plot.html

sd.fitter.plot - Function

5.1.4 Plot fit

Description

Plot the last fit.
There are three arguments that control plot. If residual argument is True, the
plot contains residual in addition to original data and fit. If plotparms
argument is True, the parameter values will be written on the plot explicitly.
The components argument specifies a list of components to plot.
components=-1 means total fit (sum of all components) will be plotted.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>residual</td>
<td>An optional parameter indicating if the residual should be plotted</td>
<td>bool</td>
<td>False</td>
</tr>
<tr>
<td>components</td>
<td>a list of components to plot, e.g [0,1]</td>
<td>integer, integer array</td>
<td>None (plot total fit)</td>
</tr>
<tr>
<td>plotparms</td>
<td>Indicates if the parameter values should be present on the plot</td>
<td>bool</td>
<td>False</td>
</tr>
</tbody>
</table>

Returns

Example

2085
sd.fitter.set_data.html

**sd.fitter.set_data - Function**

5.1.4 Set the abcissa and ordinate for the fit

**Description**

Set the abcissa and ordinate for the fit. Also set the mask indicating valid points. This can be used for data vectors retrieved from a scantable. For scantable fitting use `set_scan`.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
</table>
| xdat   | The abcissa values  
|        | allowed: float array  
|        | Default:       |
| ydat   | The ordinate values  
|        | allowed: float array  
|        | Default:       |
| mask   | An optional mask  
|        | allowed: bool array  
|        | Default:       |

**Returns**

**Example**

2086
Set the function to be fitted

Description

Set the function to be fit. The argument to be set determines what the fitter will do. If you want to do polynomial fitting, you have to set poly or lpoly arguments. The value of poly or lpoly are interpreted as an order of the polynomial function to be used for fitting. The poly is for non-linear least squares fitting, while lpoly is for linear one. If you want to do line fitting, you have to set either gauss or lorentz arguments. In that case, values of gauss or lorentz arguments are interpreted as a number of gaussian/lorentzian components. Apparently, gauss=0 or lorentz=0 causes an exception.

Note that all the above arguments are exclusive each other.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>poly</td>
<td>Use a polynomial of the order given with nonlinear least squares fit</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>lpoly</td>
<td>Use a polynomial of the order given with linear least squares fit</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>gauss</td>
<td>Fit the number of gaussian specified</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>lorentz</td>
<td>Fit the number of lorentzian specified</td>
<td>integer</td>
<td></td>
</tr>
</tbody>
</table>

Returns

Example

2087
f=sd.fitter()
# will fit a 3rd order polynomial via nonlinear method
f.set_function(poly=3)
# will fit a 3rd order polynomial via linear method
f.set_function(lpoly=3)
# will fit two gaussians
f.set_function(gauss=2)
# will fit two lorentzians
f.set_function(lorentz=2)
sd.fitter.set_gauss_parameters.html

sd.fitter.set_gauss_parameters - Function

5.1.4 Set the parameters of Gaussian component

Description

Set the Parameters of a ‘Gaussian’ component, set with set function.
Three arguments, peak, centre, and fwhm, are mandatory to be set. They
specifies an initial model of Gaussian component.
The peakfixed, centrefixed, and fwhmfixed are optional parameters to indicate
if the parameters should be held fixed during the fitting process. The default is
to keep all parameters flexible. If you want to fix one of those parameters, you
can do it by setting corresponding arguments (peakfixed, centrefixed, and
fwhmfixed) to 1.
The component argument is only effective for multi-component Gaussian
fitting. It specifies the number of the component to set the specified
parameters.

Arguments
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>peak</td>
<td>The gaussian parameters (peak intensity)</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>centre</td>
<td>The gaussian parameters (line center)</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>fwhm</td>
<td>The gaussian parameters (FWHM)</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>peakfixed</td>
<td>Optional parameters to indicate if peak should be held fixed during the fitting process</td>
<td>integer</td>
<td>0 (flexible)</td>
</tr>
<tr>
<td>centrefixed</td>
<td>Optional parameters to indicate if centre should be held fixed during the fitting process</td>
<td>integer</td>
<td>0 (flexible)</td>
</tr>
<tr>
<td>fwhmfixed</td>
<td>Optional parameters to indicate if fwhm should be held fixed during the fitting process</td>
<td>integer</td>
<td>0 (flexible)</td>
</tr>
<tr>
<td>component</td>
<td>The number of the component</td>
<td>integer</td>
<td>0</td>
</tr>
</tbody>
</table>

## Returns

## Example

```python
f = sd.fitter()
s = sd.scantable('OrionS_rawACSmod_cal', average=False)
f.set_scan(s)
# set fit function as gaussian
f.set_function(gauss=1)
# set gaussian parameter with centre fixed
f.set_gauss_parameters(peak=0.5, centre=4100, fwhm=200, centrefixed=1)
f.fit()
```
sd.fitter.set_lorentz_parameters.html

**sd.fitter.set_lorentz_parameters - Function**

5.1.4 Set the parameters of Lorentzian component

**Description**

Set the Parameters of a 'Lorentzian' component, set with **sd.fitter.set_lorentz_parameters**. Three arguments, peak, centre, and fwhm, are mandatory to be set. They specify an initial model of Lorentzian component. The peakfixed, centrefixed, and fwhmfixed are optional parameters to indicate if the parameters should be held fixed during the fitting process. The default is to keep all parameters flexible. If you want to fix one of those parameters, you can do it by setting corresponding arguments (peakfixed, centrefixed, and fwhmfixed) to 1.

The component argument is only effective for multi-component Lorentzian fitting. It specifies the number of the component to set the specified parameters.

**Arguments**
Inputs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>peak</td>
<td>The lorentzian parameters (peak intensity)</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>centre</td>
<td>The lorentzian parameters (line center)</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>fwhm</td>
<td>The lorentzian parameters (FWHM)</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>peakfixed</td>
<td>Optional parameters to indicate if peak should be held fixed during the fitting process</td>
<td>integer</td>
<td>0 (flexible)</td>
</tr>
<tr>
<td>centrefixed</td>
<td>Optional parameters to indicate if centre should be held fixed during the fitting process</td>
<td>integer</td>
<td>0 (flexible)</td>
</tr>
<tr>
<td>fwhmfixed</td>
<td>Optional parameters to indicate if fwhm should be held fixed during the fitting process</td>
<td>integer</td>
<td>0 (flexible)</td>
</tr>
<tr>
<td>component</td>
<td>The number of the component</td>
<td>integer</td>
<td>0</td>
</tr>
</tbody>
</table>

Returns

Example

```python
f = sd.fitter()
s = sd.scantable('OrionS_rawACSmod_cal', average=False)
f.set_scan(s)
# set fit function as lorentzian
f.set_function(lorentz=1)
# set lorentzian parameter with centre fixed
f.set_lorentz_parameters(peak=0.5, centre=4100, fwhm=200, centrefixed=1)
f.fit()
```

2092
sd.fitter.set_parameters.html

**sd.fitter.set_parameters - Function**

5.1.4 Set the parameters to be fitted

**Description**

Set the parameters to be fitted. It takes an argument that indicates a list of parameter values and preference if those parameters are fixed or flexible during the fit. These lists should be given as a dictionary. The params and fixed in the following argument list doesn’t indicate argument itself here. Instead, they indicate keywords in the above dictionary.

This method is normally called from set_gauss_parameters, set_lorentz_parameters and set_sinusoide_parameters so that you may not need to call this method directly.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>params</td>
<td>A vector of parameters (peak, centre, fwhm)</td>
</tr>
<tr>
<td>fixed</td>
<td>A vector of which parameters are to be held fixed</td>
</tr>
<tr>
<td>component</td>
<td>In case of multiple gaussians/lorentzians, the index of the component</td>
</tr>
</tbody>
</table>

**Allowed Types**

- float array
- float array
- integer

**Default**

- None (all parameters are flexible)

**Returns**

**Example**
**sd.fitter.set_sinusoid_parameters.html**

**sd.fitter.set_sinusoid_parameters - Function**

5.1.4 Set the parameters of Sinusoidal component

**Description**

Set the Parameters of a 'Sinusoidal' component, set with `set_function`. Three arguments, ampl, period, and x0, are mandatory to be set. They specifies an initial model of Sinusoidal component.

The amplfixed, periodfixed, and x0fixed are optional parameters to indicate if the parameters should be held fixed during the fitting process. The default is to keep all parameters flexible. If you want to fix one of those parameters, you can do it by setting corresponding arguments (amplfixed, periodfixed, and x0fixed) to 1.

The component argument is only effective for multi-component Sinusoidal fitting. It specifies the number of the component to set the specified parameters.

**Arguments**
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ampl</td>
<td>The sinusoidal parameters (amplitude)</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>period</td>
<td>The sinusoidal parameters (period)</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>x0</td>
<td>The sinusoidal parameters (phase offset)</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>amplfixed</td>
<td>Optional parameters to indicate if ampl should be held fixed during fitting</td>
<td>integer</td>
<td>0 (flexible)</td>
</tr>
<tr>
<td>periodfixed</td>
<td>Optional parameters to indicate if period should be held fixed during fitting</td>
<td>integer</td>
<td>0 (flexible)</td>
</tr>
<tr>
<td>x0fixed</td>
<td>Optional parameters to indicate if x0 should be held fixed during fitting</td>
<td>integer</td>
<td>0 (flexible)</td>
</tr>
<tr>
<td>component</td>
<td></td>
<td>integer</td>
<td>0</td>
</tr>
</tbody>
</table>

**Returns**

**Example**

2095
sd.fitter.set_scan.html

**sd.fitter.set_scan - Function**

5.1.4 Set the data as a scantable

**Description**

Set the 'data' (a scantable) of the fitter.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>thescan</td>
<td>A scantable</td>
<td>scantable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mask</td>
<td>A retrieved from the scantable</td>
<td>bool array</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

**Example**
sd.fitter.store_fit.html

sd.fitter.store_fit - Function

5.1.4 Save fit parameters

Description

Save the fit parameters. 
If filename is specified, the method saves the result in an ASCII file. 
Otherwise, the result will be stored in the scantable. More specifically, the 
result will be stored in FIT subtable in the scantable. 
It works both for spectral line fitting (gaussian or lorentzian fitting) and 
polynomial fitting although it is not sure if the latter is useful or not.

Arguments

Inputs

<table>
<thead>
<tr>
<th>filename</th>
<th>If specified, save as an ASCII table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: None (store result in the scantable)</td>
</tr>
</tbody>
</table>

Returns

Example

```python
f=sd.fitter()

s=sd.scantable('OrionS_rawACSmod_cal',average=False)
f.set_scan(s)
# set fit function as gaussian
f.set_function(gauss=1)
# set gaussian parameter with centre fixed
f.set_gauss_parameters(peak=0.5,centre=4100,fwhm=200,centrefixed=1)
f.fit()
# store fit in the scantable
f.store_fit()
# store fit in an ASCII file
f.store_fit(filename='fitresult.txt')
```
5.1.5  sd.linecatalog - Tool

Line catalog

Description

The linecatalog is a wrapper for line catalog data base. These can be either ASCII tables or the tables saved from this class. The table consists of the following items:

- row index in the table
- name of the species
- line frequency
- frequency error
- line intensity

The user can filter lines by name, frequency range, or intensity range. It is possible to overlay line catalog on the spectral plot using `plotter.plot_lines`.

For ASCII type input table, Comments can be present through lines starting with '#'.

The constructor takes string that specifies a name of the catalog.

Known Issues

The name of species can’t contain spaces. If it does, it has to be wrapped in double-quotes.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Name of the catalog</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

Example

```python
l = sd.linecatalog('jpl_asap.tbl')
# set name restriction
l.set_name('NH3')
# print summary
l.summary()
```
Methods
get_frequency | Get frequency in a specified row
get_name | Get name of specie in a specified row
get_row | Get the values in a specified row
nrow | Get number of rows in the table
reset | Unset all filtering to the table
save | Save the subset of the table to disk
set_frequency_limits | Set frequency limit on the table
set_name | Set a name restriction on the table
set_strength_limits | Set line strength limit on the table
summary | Print the contents of the table
sd.linecatalog.get_frequency.html

sd.linecatalog.get_frequency - Function

5.1.5 Get frequency in a specified row

Description

The method returns a frequency value in the catalog for given row number.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Row number</td>
<td>integer</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

Returns

float

Example
sd.linecatalog.get_name.html

**sd.linecatalog.get_name - Function**

**5.1.5 Get name of specie in a specified row**

**Description**

The method returns a name of species in the catalog for given row number.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row number</td>
<td>allowed: integer</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

string

**Example**
**sd.linecatalog.get_row - Function**

Get the values in a specified row

**Description**

The method returns the values in a specified row of the catalog. Returned value is a dictionary that contains a name of the species and its frequency for that row.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>The row to retrieve</td>
</tr>
</tbody>
</table>

| allowed: | integer |
| Default: | 0 |

**Returns**

dictionary

**Example**

2103
sd.linecatalog.nrow.html

sd.linecatalog.nrow - Function

5.1.5 Get number of rows in the table

Description

The method returns a number of rows of the catalog.

Arguments

Returns

integer

Example
sd.linecatalog.reset.html

**sd.linecatalog.reset - Function**

5.1.5 Unset all filtering to the table

**Description**

This method resets the table to its initial state, i.e. undo all calls of filtering methods on the catalog.

**Arguments**

**Returns**

**Example**
sd.linecatalog.save - Function

5.1.5 Save the subset of the table to disk

Description

Save the subset of the table to disk. This uses an internal data format and can be read in again. If no filtering is done on the catalog, it just copies the original catalog.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the output catalog</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>overwrite</td>
<td>Overwrite existing table if True</td>
<td>bool</td>
<td></td>
</tr>
</tbody>
</table>

Returns

Example
sd.linecatalog.set_frequency_limits.html

sd.linecatalog.set_frequency_limits - Function

5.1.5 Set frequency limit on the table

Description

Set frequency limits on the table.
Note that the underlying table contains frequency values in MHz

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmin</td>
<td>The lower bound</td>
<td>float</td>
<td>1.0</td>
</tr>
<tr>
<td>fmax</td>
<td>The upper bound</td>
<td>float</td>
<td>120.0</td>
</tr>
<tr>
<td>unit</td>
<td>The frequency unit</td>
<td>string ('GHz' or 'MHz')</td>
<td>'GHz'</td>
</tr>
</tbody>
</table>

Returns

Example
5.1.5 Set a name restriction on the table

Description

Set a name restriction on the table. This can be a standard unix-style pattern or a regular expression.

Arguments

| Inputs | | |
| --- | --- | |
| name | The name pattern/regex allowed: string Default: | |
| mode | The matching mode, i.e. 'pattern' or 'regex' allowed: string Default: 'pattern' |

Returns

Example
5.1.5 Set line strength limit on the table

Description

Set line strength limits on the table (arbitrary units).

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed: float</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>smin</td>
<td>The lower bound</td>
<td></td>
<td></td>
</tr>
<tr>
<td>smax</td>
<td>The upper bound</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Returns

Example
sd.linecatalog.summary.html

sd.linecatalog.summary - Function

Print the contents of the table

Description

Print the contents of the table.

Arguments

Returns

Example

```python
l = sd.linecatalog('jpl_asap.tbl')
l.set_name('NH3')
l.summary()
# the output should look like the following
--------------------------------------------------------------------------------
| 0 | NH3 | 10293.659 | -7.9334 |
| 1 | NH3 | 10426.949 | -7.6822 |
| 2 | NH3 | 10536.183 | -7.3906 |
| 3 | NH3 | 10836.127 | -7.8889 |
| 4 | NH3 | 11132.722 | -7.6055 |
| 5 | NH3 | 11673.171 | -7.6559 |
| 6 | NH3 | 11947.244 | -7.2809 |
| 7 | NH3 | 12251.33  | -6.9818 |
| 8 | NH3 | 12336.462 | -7.4568 |
| 9 | NH3 | 12923.04  | -6.9522 |
|10 | NH3 | 12951.048 | -7.4088 |
|11 | NH3 | 13296.342 | -6.7014 |
|12 | NH3 | 13297.266 | -7.316  |
|13 | NH3 | 13612.146 | -7.5443 |
|14 | NH3 | 13700.883 | -6.6744 |
```

2110
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>NH3</td>
<td>13719.23</td>
<td>-6.9543</td>
</tr>
<tr>
<td>16</td>
<td>NH3</td>
<td>13974.605</td>
<td>-6.897</td>
</tr>
<tr>
<td>17</td>
<td>NH3</td>
<td>14224.647</td>
<td>-6.3276</td>
</tr>
<tr>
<td>18</td>
<td>NH3</td>
<td>14376.817</td>
<td>-6.1817</td>
</tr>
</tbody>
</table>
5.1.6 sd.linefinder - Tool

Line finder tool for single-dish spectrum

Description

The linefinder performs automated spectral line search. The algorithm involves a simple threshold criterion. The line is considered to be detected if a specified number of consecutive channels (default is 3) is brighter (with respect to the current baseline estimate) than the threshold times the noise level. This criterion is applied in the iterative procedure updating baseline estimate and trying reduced spectral resolutions to detect broad lines as well. The off-line noise level is determined at each iteration as an average of 80% of the lowest variances across the spectrum (i.e. histogram equalization is used to avoid missing weak lines if strong ones are present). For bad baseline shapes it is recommended to increase the threshold and possibly switch the averaging option off (see set_options) to detect strong lines only, fit a high order baseline and repeat the line search.

There are six parameters for the algorithm. These can be set by set_options method of this object.

- **threshold**
  A single channel S/N ratio above which the channel is considered to be a detection. Default is sqrt(3), which together with min_nchan=3 gives a 3-sigma criterion

- **min_nchan**
  A minimal number of consecutive channels, which should satisfy a threshold criterion to be a detection. Default is 3.

- **avg_limit**
  A number of consecutive channels not greater than this parameter can be averaged to search for broad lines. Default is 8.

- **box_size**
  A running mean/median box size specified as a fraction of the total spectrum length. Default is 1/5

- **noise_box**
  Area of the spectrum used to estimate noise stats. Both string values and numbers are allowed. Allowed string values are 'all' that use all the spectrum (default), and 'box' means noise box is the same as running mean/median box. Numeric values are defined as a fraction from the spectrum size. Values should be positive. (noise_box == box_size has the same effect as noise_box = 'box')
- **noise_stat**
  Statistics used to estimate noise. Allowed values are 'mean80' that use the 80% of the lowest deviations in the noise box (default) and 'median' means median of deviations in the noise box.

The constructor doesn't take any arguments. It creates linefinder object without any settings for line finding.

**Example**

```python
fl = sd.linefinder()
sc = sd.scantable('sddata.asap', average=False)
# set data
fl.set_scan(sc)
# set linefinder options
fl.set_options(threshold=3)
# search lines
nlines = fl.find_lines(edge=(50, 0))
# get range of lines found
if nlines != 0:
    print "Found ", nlines, " spectral lines"
    print fl.get_ranges(False)
else:
    print "No lines found!"
# baseline subtraction using masks provided by linefinder
sc2 = sc.poly_baseline(fl.get_mask(), 7)
```

**Methods**

- **find_lines** Search for spectral lines in the scan
- **get_mask** Get the mask to mask all lines that have been found
- **get_ranges** Get ranges for all spectral lines found
- **set_data** Set the data as an array
- **set_options** Set the parameters of the line finding algorithm
- **set_scan** Set the data as a scantable
**sd.linefinder.find_lines.html**

**sd.linefinder.find_lines - Function**

5.1.6 Search for spectral lines in the scan

**Description**

Search for spectral lines in the scan assigned in `set_scan`. A number of lines found will be returned.

The method allows to set optional masks for the search. The mask parameter is a bool array that sets channel-by-channel masking. On the other hand, the edge parameter sets a number of channels to drop at the edge of the spectrum. The number can be set for each side as an integer array with length of two. If only one value is specified, the same number will be dropped from both sides of the spectrum.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>nRow</strong></td>
<td>A row number in the scantable to work with</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>mask</strong></td>
<td>An optional mask</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>bool array</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>[] (no mask)</td>
<td></td>
</tr>
<tr>
<td><strong>edge</strong></td>
<td>An optional number of channels to drop at the edge of the spectrum</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>integer, integer array</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>(0,0) (keep all channels)</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

integer

**Example**
sd.linefinder.get_mask.html

sd.linefinder.get_mask - Function
5.1.6 Get the mask to mask all lines that have been found

Description

By default, the method returns the mask to mask out all lines that have been found. If the invert option is True, inverted mask that only channels belong to lines are unmasked, will be returned.
Note that all channels originally masked by the input mask or dropped out by the edge parameter will still be excluded regardless on the invert option.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invert</td>
<td>If True, only channels belong to lines will be unmasked</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>False</td>
</tr>
</tbody>
</table>

Returns

bool array

Example
The method returns ranges (start and end channels or velocities) for all spectral lines found.
By default, the unit of returned values will be used the same unit as set for the scan that is set by this object using `set_scan`. If `defunits` option is set to False, the range will always be expressed in channels.

### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>defunits</code></td>
<td>If False, the returned range will be expressed in channels</td>
</tr>
<tr>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td>Default:</td>
<td>True</td>
</tr>
</tbody>
</table>

### Returns

`bool` array

### Example
sd.linefinder.set_data.html

**sd.linefinder.set_data - Function**

5.1.6 Set the data as an array

**Description**

Set the 'data' (spectrum) to work with Parameters: a method to allow linefinder work without setting scantable for the purpose of using linefinder inside some method in scantable class. (Dec 22, 2010 by W.Kawasaki)

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spectrum</td>
<td>The data to be set</td>
</tr>
<tr>
<td></td>
<td>allowed: float array</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

**Example**
**sd.linefinder.set_options - Function**

*5.1.6 Set the parameters of the line finding algorithm*

**Description**

This method is used to set the parameters of the line finding algorithm. There are six parameters that can be set by this method. See description of this object for details about parameters.

Note that, for bad baselines, threshold should be increased, and avg_limit decreased (or even switched off completely by setting this parameter to 1) to avoid detecting baseline undulations instead of real lines.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>threshold</strong></td>
<td>A single channel S/N ratio above which the channel is considered to be a detection.</td>
<td>float</td>
<td>sqrt(3)</td>
</tr>
<tr>
<td><strong>min_nchan</strong></td>
<td>A minimal number of consecutive channels, which should satisfy a threshold criterion to be a detection.</td>
<td>integer</td>
<td>3</td>
</tr>
<tr>
<td><strong>avg_limit</strong></td>
<td>A number of consecutive channels not greater than this parameter can be averaged to search for broad lines.</td>
<td>integer</td>
<td>8</td>
</tr>
<tr>
<td><strong>box_size</strong></td>
<td>A running mean/median box size specified as a fraction of the total spectrum length.</td>
<td>float</td>
<td>0.2</td>
</tr>
<tr>
<td><strong>noise_box</strong></td>
<td>Area of the spectrum used to estimate noise stats</td>
<td>float, string ('all' or 'box')</td>
<td>'all'</td>
</tr>
<tr>
<td><strong>noise_stat</strong></td>
<td>Statistics used to estimate noise</td>
<td>string ('mean80' or 'median')</td>
<td>'mean80'</td>
</tr>
</tbody>
</table>

**Returns**

2118
sd.linefinder.set_scan.html

**sd.linefinder.set_scan - Function**

Set the data as a scantable

**Description**

The method sets the data to work with. The data must be given as a scantable.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>The data to be set</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>scantable</td>
</tr>
</tbody>
</table>

**Returns**

**Example**
Simplified line finder tool for single-dish spectrum

Description

A simplified class to search for spectral features. The algorithm assumes that the bandpass is taken out perfectly and no spectral channels are flagged (except some edge channels). It works with a list or tuple rather than a scantable and returns the channel pairs. There is an optional feature to attempt to split the detected lines into components, although it should be used with caution. This class is largely intended to be used with scripts. The fully featured version of the algorithm working with scantables is called linefinder.

Methods

channelRange: Convert supplied velocity range into the channel range
find_lines: Search for spectral lines in the spectrum
invertChannelSelection: Invert channel range selection
median: Return a median of the last spectrum passed to find_lines
rms: Return rms calculated during last find_lines call
writeLog: Write user defined string into log file
sd.simplelinefinder.channelRange - Function

5.1.7 Convert supplied velocity range into the channel range

Description

A helper method which works on a tuple with abcissa/flux vectors (i.e. as returned by uvSpectrum). It allows to convert supplied velocity range into the channel range.

The argument spc specifies abcissa/flux vectors as tuple. First and second elements of the tuple should be abcissa value and spectrum itself, respectively.

The argument vel_range must be 2-element tuple that indicates start and stop velocity of range.

Note, if supplied range is completely outside the spectrum, an empty tuple will be returned.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>spc</td>
<td>tuple with the abcissa and spectrum</td>
<td>tuple</td>
<td></td>
</tr>
<tr>
<td>vel_range</td>
<td>a 2-element tuple with start and stop velocity of the range</td>
<td>tuple</td>
<td></td>
</tr>
</tbody>
</table>

Returns

a 2-element tuple with channels

Example
A simple spectral line detection routine, which assumes that bandpass has been taken out perfectly and no channels are flagged within the spectrum. A detection is reported if consecutive minchan number of channels is consistently above or below the median value. The threshold is given in terms of the rms calculated using 80% of the lowest data points by the absolute value (with respect to median).

This method returns a list of tuples each containing start and stop 0-based channel number of every detected line. Empty list if nothing has been detected.

Note. The median and rms about this median is stored inside this class and can be obtained with rms and median methods.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spc</td>
<td>a list or tuple with the spectrum, no default</td>
</tr>
<tr>
<td>edge</td>
<td>detection threshold</td>
</tr>
<tr>
<td>minchan</td>
<td>minimum number of consecutive channels exceeding threshold to claim the detection</td>
</tr>
<tr>
<td>tailsearch</td>
<td>if True (default), the algorithm attempts to widen each line until its flux crosses the median. It merges lines if necessary</td>
</tr>
<tr>
<td>splitFeatures</td>
<td>if True, the algorithm attempts to split each detected spectral feature into a number of spectral lines (just one local extremum)</td>
</tr>
</tbody>
</table>

- **spc**
  - allowed: tuple
  - Default: tuple

- **edge**
  - allowed: integer
  - Default: 0

- **minchan**
  - allowed: integer
  - Default: 3

- **tailsearch**
  - allowed: bool
  - Default: True

- **splitFeatures**
  - allowed: bool
  - Default: False
**Returns**
list of tuples that indicates ranges of detected lines

**Example**
sd.simplelinefinder.invertChannelSelection.html

sd.simplelinefinder.invertChannelSelection - Function

5.1.7 Invert channel range selection

Description

This method converts a tuple with channel ranges to a tuple which covers all channels not selected by the original tuple (optionally edge channels can be discarded).
Note, at this stage channel ranges are assumed to be sorted and without overlap.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>nchan</td>
<td>number of channels in the spectrum</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>allowed: integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>chans</td>
<td>list of start and stop channels for all selected ranges</td>
<td>tuple</td>
</tr>
<tr>
<td></td>
<td>allowed: tuple</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
</tr>
<tr>
<td>edge</td>
<td>how many channels to reject from the edge of spectrum</td>
<td>integer or tuple</td>
</tr>
<tr>
<td></td>
<td>allowed: (0,0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

Returns

tuple with inverted channel selection

Example
sd.simplelinefinder.median.html

**sd.simplelinefinder.median - Function**

5.1.7 Return a median of the last spectrum passed to `find_lines`

**Description**

Return the median of the last spectrum passed to `find_lines`. Note, this method throws an exception if `find_lines` has never been called.

**Arguments**

**Returns**

float

**Example**
sd.simplelinefinder.rms.html

**sd.simplelinefinder.rms - Function**

5.1.7 Return rms calculated during last find_lines call

**Description**

Return rms scatter of the spectral points (with respect to the median) calculated during last find_lines call. Note, this method throws an exception if find_lines has never been called.

**Arguments**

**Returns**

float

**Example**

2127
sd.simplelinefinder.writeLog.html

sd.simplelinefinder.writeLog - Function

5.1.7 Write user defined string into log file

Description

Write user defined string into log file.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>log message</td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
</tr>
<tr>
<td></td>
<td>string</td>
</tr>
</tbody>
</table>

Returns

Example
5.1.8 sd.plotter - Tool

Single-dish specific plotter tool

Description

This is a plotter object that is properly designed for single-dish tool. It supports stacking, multi-panel plotting, and multi-page plotting of spectral plot (channel/frequency/velocity versus spectral data). For total power (single-channel) data, it provides special function to plot total power data versus time. It also supports to plot time variation of azimuth and elevation and to plot pointing directions on the sky.

Known Issues

The multi-page plotting doesn’t support to go back previous page at the moment, it only allows to go forward.

Methods

- annotate: Annotate text at specified location
- arrow: Draw arrow on specified axis
- axhline: Draw a horizontal line
- axhspan: Draw a horizontal rectangle
- axvline: Draw a vertical line
- axvspan: Draw a vertical rectangle
- casabar_exists: Check if casa toolbar exists or not
- clear_header: Clear header
- create_mask: Interactively define a mask
- figtext: Add text to figure at specified location
- gca: Get current axes
- plot: Plot a scantable
- plot_lines: Plot a line catalog
- plotazel: Plot azimuth and elevation versus time of a scantable
- plotpointing: Plot telescope pointings in a scantable
- plotppi: Plot total power data
- print_header: Print header of the scantable on the plot and/or logger
- refresh: Do a soft refresh on the plot
- save: Save the plot to a file
- set_abcissa: Set the x-axis label of the plot
- set_colors: Set the colors to be used
- set_colours: Set the colors to be used
- set_data: Set a scantable to plot
- set_font: Set font properties
- set_histogram: Enable/Disable histogram plotting
- set_layout: Set the multi-panel layout
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set_legend</td>
<td>Specify a mapping for the legend</td>
</tr>
<tr>
<td>set_linestyles</td>
<td>Set the linestyles to be used</td>
</tr>
<tr>
<td>set_mask</td>
<td>Set a plotting mask for a specific selection of the data</td>
</tr>
<tr>
<td>set_mode</td>
<td>Set the plots look and feel</td>
</tr>
<tr>
<td>set_ordinate</td>
<td>Set y-axis label of the plot</td>
</tr>
<tr>
<td>set_panelling</td>
<td>Set the panelling mode</td>
</tr>
<tr>
<td>set_range</td>
<td>Set the range of interest on the abscissa of the plot</td>
</tr>
<tr>
<td>set_selection</td>
<td>Set selection to the data</td>
</tr>
<tr>
<td>set_stacking</td>
<td>Set the stacking mode</td>
</tr>
<tr>
<td>set_title</td>
<td>Set the title of the plot</td>
</tr>
<tr>
<td>text</td>
<td>Add text in a specified location</td>
</tr>
</tbody>
</table>
sd.plotter.annotate.html

**sd.plotter.annotate - Function**

5.1.8 Annotate text at specified location

**Description**

Annotate text at specified location. This is an interface for matplotlib.axes.Axes.annotate function. The interactive argument is specific to this method (not available from matplotlib). If it is True, you can set positions interactively using GUI panel.

**Arguments**

See matplotlib help about detailed description of arguments.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>Annotate text</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>xy</td>
<td>Position of the annotation</td>
</tr>
<tr>
<td></td>
<td>allowed: float array</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>xytext</td>
<td>Position of the text</td>
</tr>
<tr>
<td></td>
<td>allowed: float array</td>
</tr>
<tr>
<td></td>
<td>Default: None (use xy value)</td>
</tr>
<tr>
<td>xycoords</td>
<td>Coordinate of xy</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: 'data'</td>
</tr>
<tr>
<td>textcoords</td>
<td>Coordinate of xytext</td>
</tr>
<tr>
<td></td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: 'data'</td>
</tr>
<tr>
<td>arrowprops</td>
<td>Line properties for the arrow</td>
</tr>
<tr>
<td></td>
<td>allowed: dictionary</td>
</tr>
<tr>
<td></td>
<td>Default: None</td>
</tr>
<tr>
<td>interactive</td>
<td>Interactively set text position if True</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
</tbody>
</table>

**Returns**

**Example**

2131
sd.plotter.arrow.html

sd.plotter.arrow - Function

5.1.8 Draw arrow on specified axis

Description

Draws arrow on specified axis from (x,y) to (x+dx,y+dy). This is an interface for matplotlib.axes.Axes.arrow function. The interactive argument is specific to this method (not available from matplotlib). If it is True, you can set positions interactively using GUI panel.

Arguments

See matplotlib help about detailed description of arguments.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>x</th>
<th>x position for the origin of the arrow allowed: float Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>y</td>
<td>y position for the origin of the arrow allowed: float Default:</td>
</tr>
<tr>
<td>dx</td>
<td>dx</td>
<td>x position for the terminal of the arrow (offset from origin) allowed: float Default:</td>
</tr>
<tr>
<td>dy</td>
<td>dy</td>
<td>y position for the terminal of the arrow (offset from origin) allowed: float Default:</td>
</tr>
<tr>
<td>interactive</td>
<td>interactive</td>
<td>Interactively set origin and terminal of the arrow if True allowed: bool Default:</td>
</tr>
</tbody>
</table>

Returns

Example
sd.plotter.axhline.html

**sd.plotter.axhline - Function**

5.1.8 Draw a horizontal line

**Description**

Draw a horizontal line. This is an interface for matplotlib.axes.Axes.axhline function.
The interactive argument is specific to this method (not available from matplotlib). If it is True, you can set positions interactively using GUI panel.

**Arguments**

See matplotlib help about detailed description of arguments.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>y position of the horizontal line</td>
</tr>
<tr>
<td>xmin</td>
<td>Origin of the horizontal line</td>
</tr>
<tr>
<td>xmax</td>
<td>Terminate of the horizontal line</td>
</tr>
<tr>
<td>interactive</td>
<td>Interactively set origin and terminal of the line if True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>xmin</td>
<td>float</td>
<td>0</td>
</tr>
<tr>
<td>xmax</td>
<td>float</td>
<td>1</td>
</tr>
<tr>
<td>interactive</td>
<td>bool</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

Line2D object

**Example**
**sd.plotter.axhspan - Function**

**5.1.8** Draw a horizontal rectangle

**Description**

Draw a horizontal rectangle. This is an interface for matplotlib.axes.Axes.axhspan function. The interactive argument is specific to this method (not available from matplotlib). If it is True, you can set positions interactively using GUI panel.

**Arguments**

See matplotlib help about detailed description of arguments.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ymin</td>
<td>Bottom edge of the rectangle</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>ymax</td>
<td>Top edge of the rectangle</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>xmin</td>
<td>Left edge of the rectangle</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>xmax</td>
<td>Right edge of the rectangle</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>interactive</td>
<td>Interactively set the shape of the rectangle if True</td>
<td>bool</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

Polygon object

**Example**
sd.plotter.axvline.html

sd.plotter.axvline - Function

5.1.8 Draw a vertical line

Description

Draw a vertical line. This is an interface for matplotlib.axes.Axes.axvline function. The interactive argument is specific to this method (not available from matplotlib). If it is True, you can set positions interactively using GUI panel.

Arguments

See matplotlib help about detailed description of arguments.

Inputs

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x position of the vertical line</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>ymin</td>
<td>Origin of the vertical line</td>
<td>float</td>
<td>0</td>
</tr>
<tr>
<td>ymax</td>
<td>Terminate of the vertical line</td>
<td>float</td>
<td>1</td>
</tr>
<tr>
<td>interactive</td>
<td>Interactively set origin and terminal of the line if True</td>
<td>bool</td>
<td></td>
</tr>
</tbody>
</table>

Returns

Line2D object

Example
sd.plotter.axvspan.html

**sd.plotter.axvspan - Function**

**5.1.8** Draw a vertical rectangle

**Description**

Draw a vertical rectangle. This is an interface for matplotlib.axes.Axes.axvspan function. The interactive argument is specific to this method (not available from matplotlib). If it is True, you can set positions interactively using GUI panel.

**Arguments**

See matplotlib help about detailed description of arguments.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmin</td>
<td>Left edge of the rectangle</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>xmax</td>
<td>Right edge of the rectangle</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>ymin</td>
<td>Bottom edge of the rectangle</td>
<td>float</td>
<td>0</td>
</tr>
<tr>
<td>ymax</td>
<td>Top edge of the rectangle</td>
<td>float</td>
<td>1</td>
</tr>
<tr>
<td>interactive</td>
<td>Interactively set the shape of the rectangle if True</td>
<td>bool</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

Polygon object

**Example**
sd.plotter.casabar_exists.html

sd.plotter.casabar_exists - Function

5.1.8 Check if casa toolbar exists or not

Description

The function checks if plotter object associates with casa toolbar. It returns True if the object associates with casa toolbar. Otherwise, it returns False.

Arguments

Returns

bool

Example
sd.plotter.clear_header.html

sd.plotter.clear_header - Function

5.1.8 Clear header

Description

The function clears out header information from the plotter object. Nothing is done if plotter object doesn’t have header.

Arguments

Returns

Example
sd.plotter.create_mask.html

**sd.plotter.create_mask** - Function

5.1.8 Interactively define a mask

**Description**

Interactively define a mask. It retruns a mask that is equivalent to the one created manually with `scantable.create_mask`.

The `nwin` argument indicates a number of mask windows to create interactively. The `panel` argument specifies which panel to use for mask selection (0-based). This is useful if different IFs are spread over panels.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nwin</code></td>
<td>Number of mask windows to create interactively</td>
<td>integer</td>
<td>1</td>
</tr>
<tr>
<td><code>panel</code></td>
<td>Which panel to use for mask selection</td>
<td>integer</td>
<td>0</td>
</tr>
</tbody>
</table>

**Returns**

bool array

**Example**
sd.plotter.figtext.html

sd.plotter.figtext - Function

Add text to figure at specified location

Description

Add text to figure at location x,y (relative 0-1 coords). This method forwards *args and **kwargs to a Matplotlib method, matplotlib.Figure.text. See the method help for detailed information.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x position of the text</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>y position of the text</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>Text to be added</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

Returns

Example
sd.plotter.gca.html

sd.plotter.gca - Function

5.1.8 Get current axes

Description

Return the current axis object.

Arguments

Returns

Subplot object

Example
Plot a scantable.

If a scantable was specified in a previous call to plot, no argument has to be given to ’replot’ NO checking is done that the abcissas of the scantable are consistent e.g. all 'channel' or all 'velocity' etc.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>A scantable</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>A scantable</td>
</tr>
<tr>
<td>allowed:</td>
<td>scantable</td>
</tr>
<tr>
<td>Default:</td>
<td>None</td>
</tr>
</tbody>
</table>

Returns

Example

```python
# create scantable
s = sd.scantable('OrionS_rawACSmod_cal', average=False)
# plot
sd.plotter.plot(s)
```
sd.plotter.plot_lines.html

sd.plotter.plot_lines - Function

5.1.8 Plot a line catalog

Description

Plot a line catalog.
The linecat argument specifies actual catalog to be plot. It must be given as a linecatalog object.
Note that if the spectrum is flagged no line will be drawn in that location.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>linecat</td>
<td>The linecatalog to plot</td>
</tr>
<tr>
<td>doppler</td>
<td>The velocity shift to apply to the frequencies</td>
</tr>
<tr>
<td>deltachen</td>
<td>The number of channels to include each side of the line to determine a local maximum/minimum</td>
</tr>
<tr>
<td>rotate</td>
<td>The rotation (in degrees) for the text label</td>
</tr>
<tr>
<td>location</td>
<td>The location of the line annotation from the 'top', 'bottom' or alternate</td>
</tr>
</tbody>
</table>

Returns

Example
**sd.plotter.plotazel - Function**

5.1.8 Plot azimuth and elevation versus time of a scantable

**Description**

Plot azimuth and elevation versus time of a scantable. If outfile is specified, the plot will be saved to a disk.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>A scantable</td>
</tr>
<tr>
<td>outfile</td>
<td>Output file name</td>
</tr>
</tbody>
</table>

| allowed      | scantable                             |
| Default      | None                                  |

| allowed      | string                                |
| Default      | None                                  |

**Returns**

**Example**
sd.plotter.plotpointing.html

**sd.plotter.plotpointing - Function**

5.1.8 Plot telescope pointings in a scantable

**Description**

Plot telescope pointings in a scantable. If outfile is specified, the plot will be saved to a disk.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>A scantable</td>
<td>scantable</td>
<td>None</td>
</tr>
<tr>
<td>outfile</td>
<td>Output file name</td>
<td>string</td>
<td>None</td>
</tr>
</tbody>
</table>

**Returns**

**Example**
5.1.8 Plot total power data

Description

Plot total power data versus row number (or time). If outfile is specified, the plot will be saved to a disk.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>A scantable</td>
</tr>
<tr>
<td>outfile</td>
<td>Output file name</td>
</tr>
</tbody>
</table>

Returns

Example
sd.plotter.print_header - Function

5.1.8 Print header of the scantable on the plot and/or logger

Description

Print data (scantable) header on the plot and/or logger.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>plot</td>
<td>Whether or not print header info on the plot</td>
<td>bool</td>
<td>True</td>
</tr>
<tr>
<td>fontsize</td>
<td>Header font size (valid only plot=True)</td>
<td>integer</td>
<td>9</td>
</tr>
<tr>
<td>logger</td>
<td>Whether or not print header info on the logger</td>
<td>bool</td>
<td>False</td>
</tr>
<tr>
<td>selstr</td>
<td>Additional selection string (not verified)</td>
<td>string</td>
<td>&quot;</td>
</tr>
<tr>
<td>extrastr</td>
<td>Additional string to print (not verified)</td>
<td>string</td>
<td>&quot;</td>
</tr>
</tbody>
</table>

Returns

Example

2148
sd.plotter.refresh.html

**sd.plotter.refresh - Function**

Do a soft refresh on the plot

**Description**

Do a soft refresh on the plot.

**Arguments**

**Returns**

**Example**
sd.plotter.save.html

**sd.plotter.save - Function**

Save the plot to a file

**Description**

Save the plot to a file. The known formats are 'png', 'ps', 'eps'. If filename have suffix, the image format will be automatically detected. If no filename is specified a file called 'yyyymmdd_hhmmss.png' is created in the current directory.

The orientation argument is an optional parameter for postscript only (not eps). 'landscape', 'portrait' or None (default) are valid. If None is chosen for 'ps' output, the plot is automatically oriented to fill the page.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>The name of the output file</td>
</tr>
<tr>
<td>orientation</td>
<td>Optional parameter for postscript that specifies orientation of the plot</td>
</tr>
<tr>
<td>dpi</td>
<td>The dpi of the output non-postscript plot</td>
</tr>
</tbody>
</table>

*Allowed:* bool, string ('landscape' or 'portrait'), integer

*Default:* True, None, False (150 dpi)

**Returns**

**Example**

2150
sd.plotter.set_abcissa.html

**sd.plotter.set_abcissa - Function**

5.1.8 Set the x-axis label of the plot

**Description**

Set the x-axis label of the plot. If multiple panels are plotted, multiple labels have to be specified.

If no abcissa labels are specified (i.e. None), data determine the labels.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abcissa</td>
<td>A list of abcissa labels</td>
</tr>
<tr>
<td>fontsize</td>
<td>A font size of the label</td>
</tr>
<tr>
<td>refresh</td>
<td>If True, the plot is replotted based on the new parameter setting(s)</td>
</tr>
</tbody>
</table>

- **abcissa**: string, string array
- **Default**: None
- **fontsize**: integer
- **Default**: None
- **refresh**: bool
- **Default**: True

**Returns**

**Example**

```python
# two panels are visible on the plotter
sd.plotter.set_ordinate(["First X-Axis", "Second X-Axis"])
```
sd.plotter.set_colors.html

**sd.plotter.set_colors - Function**

5.1.8 Set the colors to be used

**Description**

Set the colours to be used. The plotter will cycle through these colours when lines are overlaid (stacking mode).

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>colmap</td>
<td>A list of color names</td>
</tr>
<tr>
<td>refresh</td>
<td>If True, the plot is replotted based on the new parameter setting(s)</td>
</tr>
</tbody>
</table>

**Returns**

**Example**

```python
sd.plotter.set_colors("red green blue")
# If for example four lines are overlaid e.g I Q U V
# 'I' will be 'red', 'Q' will be 'green', U will be 'blue'
# and 'V' will be 'red' again.
```
sd.plotter.set_colours.html

sd.plotter.set_colours - Function

5.1.8 Set the colors to be used

Description

See `set_colors`

Arguments

Returns

Example
Set a scantable to plot

Description

Set a scantable to plot.
Note that the user specified masks and data selections will be reset if a new
scantable is set. This method should be called before setting data selections
(set_selection) and/or masks (set_mask).

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>A scantable</td>
</tr>
<tr>
<td></td>
<td>allowed: scantable</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>refresh</td>
<td>If True, the plot is replotted based on the new parameter setting(s)</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: True</td>
</tr>
</tbody>
</table>

Returns

Example
sd.plotter.set_font.html

sd.plotter.set_font - Function

5.1.8 Set font properties

Description

Set font properties.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>family</td>
<td>One of 'sans-serif', 'serif', 'cursive', 'fantasy', 'monospace'</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>style</td>
<td>One of 'normal' (or 'roman'), 'italic' or 'oblique'</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>weight</td>
<td>One of 'normal' or 'bold'</td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>size</td>
<td>the 'general' font size, individual elements can be adjusted separately</td>
<td>integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>refresh</td>
<td>If True, the plot is replotted based on the new parameter setting(s)</td>
<td>bool</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Returns

Example

2155
sd.plotter.set_histogram.html

**sd.plotter.set_histogram - Function**

5.1.8 Enable/Disable histogram plotting

**Description**

Enable/Disable histogram-like plotting.
The first default is taken from sd.rcParams[plotter.histogram].

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>hist</td>
<td>Enable/Disable histogram plotting</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td>True</td>
</tr>
<tr>
<td>refresh</td>
<td>If True, the plot is replotted based on the new parameter setting(s)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td>True</td>
</tr>
</tbody>
</table>

**Returns**

**Example**
sd.plotter.set_layout.html

sd.plotter.set_layout - Function

5.1.8 Set the multi-panel layout

Description

Set the multi-panel layout, i.e. how many rows and columns plots are visible. Note that if no argument is given, the plotter reverts to its auto-plot behaviour.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rows</td>
<td>Number of rows of plots</td>
</tr>
<tr>
<td>cols</td>
<td>Number of columns of plots</td>
</tr>
<tr>
<td>refresh</td>
<td>If True, the plot is replotted based on the new parameter setting(s)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>None</td>
</tr>
<tr>
<td>integer</td>
<td>None</td>
</tr>
<tr>
<td>bool</td>
<td>True</td>
</tr>
</tbody>
</table>

Returns

Example
**sd.plotter.set_legend**

**sd.plotter.set_legend - Function**

### 5.1.8 Specify a mapping for the legend

**Description**

Specify a mapping for the legend instead of using the default indices. The list of legends should be given to the mp argument as a string. This should have the same length as the number of elements on the legend and then maps to the indices in order. It is possible to uses latex math expression. These have to be enclosed in $r$, e.g. $r'x^{2}'$

The mode argument controls where to display the legend. It should be specified as an integer. The following list shows a meaning of each integer.

- 0: auto
- 1: upper right
- 2: upper left
- 3: lower left
- 4: lower right
- 5: right
- 6: center left
- 7: center right
- 8: lower center
- 9: upper center
- 10: center

**Arguments**

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>mp</td>
<td>A list of legend strings</td>
<td>string array</td>
<td>None</td>
</tr>
<tr>
<td>fontsize</td>
<td>The font size of the label</td>
<td>integer</td>
<td>None</td>
</tr>
<tr>
<td>mode</td>
<td>Where to display the legend</td>
<td>integer</td>
<td>0 (auto)</td>
</tr>
<tr>
<td>refresh</td>
<td>If True, the plot is replotted based on the new parameter setting(s)</td>
<td>bool</td>
<td>True</td>
</tr>
</tbody>
</table>
Returns

Example

```python
# If the data has two IFs/rest frequencies with index 0 and 1
# for CO and SiO:
sd.plotter.set_stacking('i')
sd.plotter.set_legend(['CO','SiO'])
sd.plotter.plot()
sd.plotter.set_legend([r'$^{12}CO$', r'SiO'])
```

2159
sd.plotter.set_linestyles.html

sd.plotter.set_linestyles - Function

5.1.8 Set the linestyles to be used

Description

Set the linestyles to be used. The plotter will cycle through these linestyles when lines are overlaid (stacking mode) AND only one color has been set. Accepted linestyles are ‘line’, ‘dashed’, ‘dotted’, ‘dashdot’, ‘dashdotdot’ and ‘dashdashdot’.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>linestyles</td>
<td>A list of linestyles to use</td>
<td>string array</td>
</tr>
<tr>
<td>linewidth</td>
<td>A width of the line</td>
<td>integer</td>
</tr>
<tr>
<td>refresh</td>
<td>If True, the plot is replotted based on the new parameter setting(s)</td>
<td>bool</td>
</tr>
</tbody>
</table>

Returns

Example

```
sd.plotter.set_colors("black")
sd.plotter.set_linestyles("line dashed dotted dashdot")
# If for example four lines are overlaid e.g I Q U V
# 'I' will be 'solid', 'Q' will be 'dashed',
# U will be 'dotted' and 'V' will be 'dashdot'.
```
**sd.plotter.set_mask** - Function

5.1.8 Set a plotting mask for a specific selection of the data

**Description**

Set a plotting mask for a specific polarization. This is useful for masking out "noise" Pangle outside a source.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>A mask from <code>scantable.create_mask</code></td>
</tr>
<tr>
<td></td>
<td>allowed: bool array</td>
</tr>
<tr>
<td></td>
<td>Default: None</td>
</tr>
<tr>
<td>selection</td>
<td>The spectra to apply the mask to</td>
</tr>
<tr>
<td></td>
<td>allowed: selector</td>
</tr>
<tr>
<td></td>
<td>Default: None</td>
</tr>
<tr>
<td>refresh</td>
<td>If True, the plot is replotted based on the new parameter setting(s)</td>
</tr>
<tr>
<td></td>
<td>allowed: bool</td>
</tr>
<tr>
<td></td>
<td>Default: True</td>
</tr>
</tbody>
</table>

**Returns**

**Example**

```python
select = sd.selector()
select.setpolstrings("Pangle")
sd.plotter.set_mask(mymask, select)
```
sd.plotter.set_mode.html

sd.plotter.set_mode - Function

5.1.8 Set the plots look and feel

Description

Set the plots look and feel, i.e. what you want to see on the plot. Stacking is a plot as line color overlays, while panelling is a plot across multiple panels. By default, the stacking is set to 'pol' and the panelling is set to 'scan'. Valid modes are:

- 'beam' 'Beam' 'b': Beams
- 'if' 'IF' 'i': IFs
- 'pol' 'Pol' 'p': Polarisations
- 'scan' 'Scan' 's': Scans
- 'time' 'Time' 't': Times

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>stacking</td>
<td>Tell the plotter which variable to plot as line colour overlays</td>
<td>string</td>
<td>None</td>
</tr>
<tr>
<td>panelling</td>
<td>Tell the plotter which variable to plot across multiple panels</td>
<td>string</td>
<td>None</td>
</tr>
<tr>
<td>refresh</td>
<td>If True, the plot is replotted based on the new parameter setting(s)</td>
<td>bool</td>
<td>True</td>
</tr>
</tbody>
</table>

Returns

Example

2162
sd.plotter.set_ordinate.html

sd.plotter.set_ordinate - Function

5.1.8 Set y-axis label of the plot

Description

Set the y-axis label of the plot. If multiple panels are plotted, multiple labels have to be specified. If no ordinate is set, data determine the labels.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ordinate</td>
<td>A list of ordinate labels</td>
<td>string, string array</td>
<td>None</td>
</tr>
<tr>
<td>fontsize</td>
<td>A font size of the label</td>
<td>integer</td>
<td>None</td>
</tr>
<tr>
<td>refresh</td>
<td>If True, the plot is replotted based on the new parameter setting(s)</td>
<td>bool</td>
<td>True</td>
</tr>
</tbody>
</table>

Returns

Example

```py
# two panels are visible on the plotter
sd.plotter.set_ordinate(["First Y-Axis", "Second Y-Axis"])```
sd.plotter.set_panelling.html

**sd.plotter.set_panelling - Function**

5.1.8 Set the panelling mode

**Description**

Set the 'panelling' mode i.e. which type of spectra should be spread across different panels.

Valid modes are:

- 'beam' 'Beam' 'b': Beams
- 'if' 'IF' 'i': IFs
- 'pol' 'Pol' 'p': Polarisations
- 'scan' 'Scan' 's': Scans
- 'time' 'Time' 't': Times

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>what (what)</td>
<td>Which type of spectra should be spread across different panels</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>None</td>
</tr>
</tbody>
</table>

**Returns**

**Example**
sd.plotter.set_range.html

**sd.plotter.set_range - Function**

Set the range of interest on the abcissa of the plot

**Description**

Set the range of interest on the abcissa of the plot. These become non-sensical when the unit changes. In that case, use `plotter.set_range()` without parameters to reset.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>xstart</td>
<td>The start point of the 'zoom' window</td>
<td>float</td>
<td>None</td>
</tr>
<tr>
<td>xend</td>
<td>The end point of the 'zoom' window</td>
<td>float</td>
<td>None</td>
</tr>
<tr>
<td>ystart</td>
<td>The start point of the 'zoom' window</td>
<td>float</td>
<td>None</td>
</tr>
<tr>
<td>yend</td>
<td>The end point of the 'zoom' window</td>
<td>float</td>
<td>None</td>
</tr>
<tr>
<td>refresh</td>
<td>If True, the plot is replotted based on the new parameter setting(s)</td>
<td>bool</td>
<td>True</td>
</tr>
<tr>
<td>offset</td>
<td>Shift the abcissa by the given amount. The abcissa label will have '(relative)' appended to it.</td>
<td>float</td>
<td>True</td>
</tr>
</tbody>
</table>

**Returns**

**Example**

2165
sd.plotter.set_selection.html

**sd.plotter.set_selection** - Function

5.1.8 Set selection to the data

**Description**

Set selection to the data. The method allows both to set selector object directly and to pass arguments for constructor of the selector to create new selector object internally.

See [selector](#) for valid arguments for constructor of the selector.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>selection</td>
<td>A selector object</td>
<td>refresh</td>
</tr>
<tr>
<td>allowed:</td>
<td>selector</td>
<td>If True, the plot is replotted based on the new parameter setting(s)</td>
</tr>
<tr>
<td>Default:</td>
<td>None</td>
<td>allowed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>True</td>
</tr>
</tbody>
</table>

**Returns**

**Example**

```python
# these are equivalent
sel = sd.selector(if$=0$)
sd.plotter.set_selection(sel)
sd.plotter.set_selection(if$=0$)
```
Set the 'stacking' mode i.e. which type of spectra should be overlayed. Valid modes are:

- 'beam' 'Beam' 'b': Beams
- 'if' 'IF' 'i': IFs
- 'pol' 'Pol' 'p': Polarisations
- 'scan' 'Scan' 's': Scans
- 'time' 'Time' 't': Times

### Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>what</td>
<td>Which type of spectra should be overlayed</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>None</td>
</tr>
</tbody>
</table>

### Example

---

2167
**sd.plotter.set.title - Function**

Set the title of the plot

**Description**

Set the title of the plot. If multiple panels are plotted, multiple titles have to be specified.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>The title</td>
<td>string, string array</td>
<td>None</td>
</tr>
<tr>
<td>fontsize</td>
<td>A font size of the title</td>
<td>integer</td>
<td>None</td>
</tr>
<tr>
<td>refresh</td>
<td>If True, the plot is replotted based on the new parameter setting(s)</td>
<td>bool</td>
<td>True</td>
</tr>
</tbody>
</table>

**Returns**

**Example**

```python
# two panels are visible on the plotter
sd.plotter.set_title(["First Panel", "Second Panel"])
```
sd.plotter.text.html

**sd.plotter.text - Function**

5.1.8 Add text in a specified location

**Description**

Add text in string s to axis at location x,y (data coords). This is an interface for matplotlib.axes.Axes.text function.
The interactive argument is specific to this method (not available from matplotlib). If it is True, you can set positions interactively using GUI panel.

**Arguments**

See matplotlib help about detailed description of arguments.

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th>allowed:</th>
<th>Default:</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x position of the text</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>y position of the text</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>The text to be drawn</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>interactive</td>
<td>Interactively set the position of the text if True</td>
<td>bool</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

**Example**

sd.coordinate-Tool.html

5.1.9 sd.coordinate - Tool

Single-dish specific spectral coordinate conversion
Description

The coordinate class is a representation of the spectral coordinate (frequency axis) of the data. It handles a conversion between pixel/channel values and frequency/velocity ones under the current spectral coordinate.
Spectral coordinate is composed of a set of three values: a reference pixel, a frequency in Hz at the reference pixel, and an increment in Hz (width of each pixel).

Normally, the object of this class is not created from scratch. Instead, the object can be obtained from scantable object using get_coordinate method. Returned spectral coordinate contains spectral coordinate information and rest frequency value, which is needed to handle velocity conversion, of the given row.

Example

```python
# create a scantable object
s = sd.scantable('OrionS_rawACSmod', average=False)
# get coordinate system
c = s.get_coordinate(0)
# get coordinate system values
c.get_increment()  # 6104.2329828208008
c.get_reference_pixel()  # 4096.0
c.get_reference_value()  # 45489353563.344795
# get pixel value
s.to_pixel(4.5489e10)  # 4038.0789891175605
# get frequency at channel 0
s.to_frequency(0)  # 45464350625.063545
# get velocity at channel 0
s.to_velocity(0)  # 170.73624010940924
```

Methods

- `coordinate` Constructor
- `coordinate.get_increment` Get increment of this coordinate system
- `coordinate.get_reference_pixel` Get reference pixel of this coordinate system
- `coordinate.get_reference_value` Get reference value of this coordinate system
- `coordinate.to_frequency` Convert a channel/pixel value to a frequency
- `coordinate.to_pixel` Convert a frequency value to a channel/pixel
- `coordinate.to_velocity` Convert a channel/pixel value to a velocity
sd.coordinate.coordinate.html

sd.coordinate.coordinate - Function

5.1.9 Constructor

Description

Constructor of this class. Not useful any more.

Arguments

Returns

Example
sd.coordinate.coordinate.get_increment.html

sd.coordinate.coordinate.get_increment - Function

5.1.9 Get increment of this coordinate system

Description

Return an increment of this spectral coordinate. The unit is Hz.

Arguments

Returns

float

Example
sd.coordinate.coordinate.get_reference_pixel.html

sd.coordinate.coordinate.get_reference_pixel - Function

5.1.9 Get reference pixel of this coordinate system

Description

Return a reference pixel of this spectral coordinate.

Arguments

Returns

float

Example
sd.coordinate.coordinate.get_reference_value.html

sd.coordinate.coordinate.get_reference_value - Function

5.1.9 Get reference value of this coordinate system

Description

Return a reference frequency of this spectral coordinate. The unit is Hz.

Arguments

Returns

Example
sd.coordinate.coordinate.to_frequency.html

**sd.coordinate.coordinate.to_frequency - Function**

**5.1.9 Convert a channel/pixel value to a frequency**

**Description**

The method converts a given channel/pixel value to a frequency under this spectral coordinate. Default unit of returned value is Hz.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixel</td>
<td>Channel/pixel value where frequency wants to know</td>
</tr>
<tr>
<td></td>
<td>allowed: float</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
</tr>
<tr>
<td>unit</td>
<td>Unit of the returned value</td>
</tr>
<tr>
<td></td>
<td>allowed: string ('Hz', 'kHz', 'MHz', 'GHz')</td>
</tr>
<tr>
<td></td>
<td>Default: 'Hz'</td>
</tr>
</tbody>
</table>

**Returns**

float

**Example**

2175
5.1.9 Convert a frequency value to a channel/pixel

Description

The method converts a given frequency value to a channel/pixel under this spectral coordinate.

Arguments

| Inputs | A frequency value that want to convert to channel/pixel allowed: float
| Default: |

Returns

float

Example
sd.coordinate.coordinate.to_velocity.html

**sd.coordinate.coordinate.to_velocity - Function**

5.1.9 Convert a channel/pixel value to a velocity

**Description**

The method converts a given channel/pixel value to a velocity under this spectral coordinate. Default unit of returned value is km/s.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixel</td>
<td>Channel/pixel value where velocity wants to know</td>
</tr>
<tr>
<td></td>
<td>allowed: float</td>
</tr>
<tr>
<td>unit</td>
<td>Unit of the returned value</td>
</tr>
<tr>
<td></td>
<td>allowed: string ('km/s', 'm/s')</td>
</tr>
<tr>
<td></td>
<td>Default: 'km/s'</td>
</tr>
</tbody>
</table>

**Returns**

float

**Example**

2177
5.1.10  sd.opacity_model - Tool

Single dish specific opacity model

Description

This class implements opacity/atmospheric brightness temperature model equivalent to the model available in MIRIAD. The actual math is a conversion of the Fortran code written by Bob Sault for MIRIAD. It implements a simple model of the atmosphere and Liebe’s model (1985) of the complex refractive index of air.

The model of the atmosphere is one with an exponential fall-off in the water vapour content (scale height of 1540 m) and a temperature lapse rate of 6.5 mK/m. Otherwise the atmosphere obeys the ideal gas equation and hydrostatic equilibrium.

Note that the model includes atmospheric lines up to 800 GHz, but was not rigorously tested above 100 GHz and for instruments located at a significant elevation. For high-elevation sites it may be necessary to adjust scale height and lapse rate.

The constructor takes several arguments that specifies observatory and weather informations.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>temperature</td>
<td>Air temperature at the observatory (K)</td>
</tr>
<tr>
<td>allowed:</td>
<td>float</td>
</tr>
<tr>
<td>Default:</td>
<td>288.0</td>
</tr>
<tr>
<td>pressure</td>
<td>Air pressure at the sea level if the observatory elevation is set to non-zero value (note, by default is set to 700m) or at the observatory ground level if the elevation is set to 0. The value is in Pascals or hPa.</td>
</tr>
<tr>
<td>allowed:</td>
<td>float</td>
</tr>
<tr>
<td>Default:</td>
<td>101325.0</td>
</tr>
<tr>
<td>humidity</td>
<td>Air humidity at the observatory (fractional)</td>
</tr>
<tr>
<td>allowed:</td>
<td>float</td>
</tr>
<tr>
<td>Default:</td>
<td>0.5</td>
</tr>
<tr>
<td>elevation</td>
<td>Observatory elevation about sea level (in meters)</td>
</tr>
<tr>
<td>allowed:</td>
<td>float</td>
</tr>
<tr>
<td>Default:</td>
<td>700.0</td>
</tr>
</tbody>
</table>

Example

2178
```python
o = sd.opacity_model(temperature=300.0, elevation=1300.0)
tau = sd.get_opacities([1.0e11, 1.1e11])
print tau
[0.2363221976673901, 0.3068089293871521]
```

Methods

- **get_opacities**: Get the opacity value(s) for the given frequency(ies)
- **set_observatory_elevation**: Update the model using the given observatory elevation
- **set_weather**: Update the model using the given environmental parameters
**sd.opacity_model.get_opacities - Function**

5.1.10 Get the opacity value(s) for the given frequency(ies)

**Description**

Get the opacity value(s) for the given frequency(ies). If no elevation is given the opacities for the zenith are returned. If an elevation is specified refraction is also taken into account. The user is able to set frequency value(s) where the user want to compute opacity(ies). One opacity value per frequency is returned as a scalar or list.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>freq</td>
<td>A frequency value in Hz, or a list of frequency values.</td>
<td>float, float array</td>
<td>None</td>
</tr>
<tr>
<td>elevation</td>
<td>The elevation in radian at which to compute the opacity.</td>
<td>float</td>
<td>None (zenith opacity)</td>
</tr>
</tbody>
</table>

**Returns**

float, float array

**Example**

2180
sd.opacity_model.set_observatory_elevation.html

sd.opacity_model.set_observatory_elevation - Function

5.1.10 Update the model using the given observatory elevation

Description

Update the model using the given observatory elevation in meter. Note that, in constructor, the default value for the observatory elevation is 700m.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elevation</td>
<td>The site elevation in meter at which to compute the opacity.</td>
</tr>
<tr>
<td>allowed:</td>
<td>float</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

Returns

Example
5.1.10 Update the model using the given environmental parameters

**Description**

Update the model using the given environmental parameters. The pressure value will be a pressure at sea level if the observatory elevation is set to non-zero value, while it will be a pressure at the observatory ground level if the elevation is set to 0.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>temperature</td>
<td>Air temperature at the observatory (K)</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>pressure</td>
<td>Air pressure in Pascals or hPa.</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>humidity</td>
<td>Air humidity at the observatory (fractional)</td>
<td>float</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

**Example**
sd.asapgrid-Tool.html

5.1.11 sd.asapgrid - Tool
Tool to convolve map data onto regularly spaced grid

Description

The asapgrid class is defined to convolve data onto regular spatial grid.

Arguments

Inputs

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>infile</td>
<td>input data as a string or string list</td>
<td>string, list of string</td>
<td></td>
</tr>
</tbody>
</table>

Returns

asapgrid instance

Example

```python
# create asapgrid instance with two input data
g = asapgrid( ['testimage1.asap','testimage2.asap'] )
# set IFNU if necessary
g.setIF( 0 )
# set POLNOs if necessary
g.setPolList( [0,1] )
# set SCANNOs if necessary
g.setScanList( [22,23,24] )
# define image with full specification
# you can skip some parameters (see help for defineImage)
g.defineImage( nx=12, ny=12, cellx='10arcsec', celly='10arcsec',
               center='J2000 10h10m10s -5d05m05s' )
# set convolution function
g.setFunc( func='sf', width=3 )
# enable min/max clipping
g.enableClip()
# or, disable min/max clipping
#g.disableClip()
# actual gridding
```
g.grid()
# save result
g.save( outfile='grid.asap' )
# plot result
g.plot( plotchan=1246, plotpol=-1, plotgrid=True, plotobs=True )

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sd.asapgrid.defineImage</td>
<td>Define spatial grid</td>
</tr>
<tr>
<td>sd.asapgrid.disableClip</td>
<td>Disable min/max clipping</td>
</tr>
<tr>
<td>sd.asapgrid.enableClip</td>
<td>Enable min/max clipping</td>
</tr>
<tr>
<td>sd.asapgrid.grid</td>
<td>Do gridding</td>
</tr>
<tr>
<td>sd.asapgrid.plot</td>
<td>Plot result</td>
</tr>
<tr>
<td>sd.asapgrid.save</td>
<td>Save result</td>
</tr>
<tr>
<td>sd.asapgrid.setData</td>
<td>Set data to be processed</td>
</tr>
<tr>
<td>sd.asapgrid.setFunc</td>
<td>Set convolution function</td>
</tr>
<tr>
<td>sd.asapgrid.setIF</td>
<td>Set IFNO to be processed</td>
</tr>
<tr>
<td>sd.asapgrid.setPolList</td>
<td>Set polarizations to be processed</td>
</tr>
<tr>
<td>sd.asapgrid.setScanList</td>
<td>Set scans to be processed</td>
</tr>
<tr>
<td>sd.asapgrid.setWeight</td>
<td>Set weight type</td>
</tr>
</tbody>
</table>
sd.asapgrid.defineImage.html

sd.asapgrid.defineImage - Function

5.1.11 Define spatial grid

Description

Define spatial grid.
First two parameters, nx and ny, define number of pixels of the grid. If which
of those is not specified, it will be set to the same value as the other. If none
of them are specified, it will be determined from map extent and cell size.
Next two parameters, cellx and celly, define size of pixel. You should set those
parameters as string, which is constructed numerical value and unit, e.g.
'0.5arcmin', or numerical value. If those values are specified as numerical
value, their units will be assumed to 'arcmin'. If which of those is not
specified, it will be set to the same value as the other. If none of them are
specified, it will be determined from map extent and number of pixels, or set
to '1arcmin' if neither nx nor ny is set.
The last parameter, center, define the central coordinate of the grid. You
should specify its value as a string, like,
'J2000 05h08m50s -16d23m30s'
or
'J2000 05:08:50 -16.23.30'
You can omit equinox when you specify center coordinate. In that case, J2000
is assumed. If center is not specified, it will be determined from the observed
positions of input data.

Arguments
Inputs

nx  number of pixels along x (R.A.) direction
    allowed: int
    Default: -1

ny  number of pixels along y (Dec.) direction
    allowed: int
    Default: -1

cellx  size of pixel in x (R.A.) direction
    allowed: string, float
    Default: ""

celly  size of pixel in y (Dec.) direction.
    allowed: string, float
    Default: ""

center  central position of the grid
    allowed: string
    Default: ""

Returns

Example
sd.asapgrid.disableClip.html

sd.asapgrid.disableClip - Function

5.1.11 Disable min/max clipping

Description

Disable min/max clipping.

Arguments

Returns

Example
sd.asapgrid.enableClip.html

**sd.asapgrid.enableClip - Function**

5.1.11 Enable min/max clipping

**Description**

Enable min/max clipping.

**Arguments**

**Returns**

**Example**
sd.asapgrid.grid.html

sd.asapgrid.grid - Function

5.1.11 Do gridding

Description

Actual gridding which will be done based on several user inputs.

Arguments

Returns

Example
sd.asapgrid.plot.html

**sd.asapgrid.plot - Function**

5.1.11 Plot result

**Description**

Plot gridded data. Data must be saved using `save` method.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>plotchan</code></td>
<td>Which channel you want to plot. Default is to average all the channels.</td>
</tr>
<tr>
<td><code>plotpol</code></td>
<td>Which polarization component you want to plot. Default is to average all the</td>
</tr>
<tr>
<td></td>
<td>polarization components.</td>
</tr>
<tr>
<td><code>plotobs</code></td>
<td>Also plot observed position if True. Setting True for large amount of spectra</td>
</tr>
<tr>
<td></td>
<td>might be time consuming.</td>
</tr>
<tr>
<td><code>plotgrid</code></td>
<td>Also plot grid center if True. Setting True for large number of grids might</td>
</tr>
<tr>
<td></td>
<td>be time consuming.</td>
</tr>
</tbody>
</table>

**Returns**

**Example**

2190
sd.asapgrid.save.html

**sd.asapgrid.save - Function**

5.1.11 Save result

**Description**

Save result. By default, output data name will be constructed from first element of input data name list (e.g. 'input.asap.grid').

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>outfile</td>
<td>output data name</td>
<td>2191</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>Default:</td>
<td>”</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

**Example**
sd.asapgrid.setData.html

sd.asapgrid.setData - Function

5.1.11 Set data to be processed

Description

Set data to be processed.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>infile</td>
<td>input data as a string or string list if you want to grid more than one data at once.</td>
</tr>
<tr>
<td>allowed:</td>
<td>string, string list</td>
</tr>
<tr>
<td>Default:</td>
<td>&quot;</td>
</tr>
</tbody>
</table>

Returns

Example
Set convolution function. Possible options are 'box' (Box-car, default), 'sf'
(prolate spheroidal), and 'gauss' (Gaussian). Width of convolution function

can be set using width parameter. By default (-1), width is automatically set
depending on each convolution function. Default values for width are:
'box': 1 pixel 'sf': 3 pixels 'gauss': 1 pixel (width is used as HWHM)

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>func</strong></td>
<td>Function type ('box', 'sf', 'gauss')</td>
</tr>
<tr>
<td>allowed:</td>
<td>string</td>
</tr>
<tr>
<td>Default:</td>
<td>box</td>
</tr>
<tr>
<td><strong>width</strong></td>
<td>Width of convolution function. Default (-1) is to choose</td>
</tr>
<tr>
<td>pre-defined value for each convolution function.</td>
<td></td>
</tr>
<tr>
<td>allowed:</td>
<td>int</td>
</tr>
<tr>
<td>Default:</td>
<td>-1</td>
</tr>
</tbody>
</table>

Returns

Example

2193
sd.asapgrid.setIF - Function

Set IFNO to be processed

Description

Set IFNO to be processed. Currently, asapgrid allows to process only one IFNO for one gridding run even if the data contains multiple IFs. If you didn’t specify IFNO, default value, which is IFNO in the first spectrum, will be processed.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ifno</td>
<td>IFNO to be processed</td>
</tr>
</tbody>
</table>

allowed: int

Default:

Returns

Example
sd.asapgrid.setPolList.html

**sd.asapgrid.setPolList - Function**

5.1.11 Set polarizations to be processed

**Description**

Set list of polarization components you want to process. If not specified, all POLNOs will be processed.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pollist</td>
<td>list of POLNOs to be processed</td>
</tr>
<tr>
<td>allowed:</td>
<td>int list</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

**Example**
sd.asapgrid.setScanList.html

sd.asapgrid.setScanList - Function

**5.1.11** Set scans to be processed

**Description**

Set list of scans you want to process. If not specified, all scans will be processed.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scanlist</td>
<td>list of SCANNOs to be processed</td>
</tr>
<tr>
<td>allowed:</td>
<td>int list</td>
</tr>
<tr>
<td>Default:</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

**Example**
sd.asapgrid.setWeight.html

sd.asapgrid.setWeight - Function

5.1.11 Set weight type

Description

Set weight type. Possible options are 'uniform' (default), 'tint' (weight by integration time), 'tsys' (weight by Tsys: 1/Tsys**2), and 'tintsys' (weight by integration time as well as Tsys: tint/Tsys**2).

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>weight type (‘uniform’, ‘tint’, ‘tsys’, ‘tintsys’)</th>
</tr>
</thead>
<tbody>
<tr>
<td>weightType</td>
<td>allowed: string</td>
</tr>
<tr>
<td></td>
<td>Default: uniform</td>
</tr>
</tbody>
</table>

Returns

Example
sd.asaplog - Tool

Wrapper object to allow for both casapy and asap logging

Description

Wrapper object to allow for both casapy and asap logging.
Inside casapy this will connect to 'taskinit.casalog'. Otherwise it will create its own casa log sink.
.. note:: Do not instantiate a new one - use the :obj:`asaplog` instead.
In the ASAP logging system, log messages are accumulated to the log buffer when you post any message using :func:`sd.asaplog.push`. Buffered messages are flushed when you call :func:`sd.asaplog.post`

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sd.asaplog.clear</code></td>
<td>Clear buffer</td>
</tr>
<tr>
<td><code>sd.asaplog.disable</code></td>
<td>Disable (or enable) logging</td>
</tr>
<tr>
<td><code>sd.asaplog.enable</code></td>
<td>Enable (or disable) logging</td>
</tr>
<tr>
<td><code>sd.asaplog.is_enabled</code></td>
<td>Query if logging is enabled</td>
</tr>
<tr>
<td><code>sd.asaplog.post</code></td>
<td>Post message to the logger</td>
</tr>
<tr>
<td><code>sd.asaplog.push</code></td>
<td>Push logs into the buffer</td>
</tr>
</tbody>
</table>
sd.asaplog.clear.html

**sd.asaplog.clear - Function**

5.1.12 Clear buffer

**Description**

Clear buffer. This is only effective for standalone ASAP.

**Arguments**

**Returns**

**Example**
sd.asaplog.disable.html

sd.asaplog.disable - Function

5.1.12 Disable (or enable) logging

Description

Disable (or enable) logging.

Arguments

Inputs
flag
Set False to disable logging
allowed: bool
Default: False

Returns

Example
sd.asaplog.enable.html

**sd.asaplog.enable - Function**

5.1.12 Enable (or disable) logging

**Description**

Enable (or disable) logging.

**Arguments**

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th>Set True to enable logging</th>
</tr>
</thead>
<tbody>
<tr>
<td>flag</td>
<td>allowed:</td>
<td>bool</td>
</tr>
<tr>
<td></td>
<td>Default:</td>
<td>True</td>
</tr>
</tbody>
</table>

**Returns**

**Example**
sd.asaplog.is_enabled.html

sd.asaplog.is_enabled - Function

5.1.12 Query if logging is enabled

Description

Query if logging is enabled.

Arguments

Returns

Example

bool
sd.asaplog.post.html

sd.asaplog.post - Function

5.1.12 Post message to the logger

Description

Post the messages to the logger. This will clear the buffered logs.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>level</td>
<td>The log level (severity). One of INFO, WARN, ERROR.</td>
</tr>
<tr>
<td>origin</td>
<td>Origin of the log message.</td>
</tr>
</tbody>
</table>

| allowed: | string |
| Default: | INFO   |

| allowed: | string |
| Default: |        |

Returns

Example
sd.asaplog.push.html

sd.asaplog.push - Function
5.1.12 Push logs into the buffer

Description

Push logs into the buffer. post needs to be called to send them.

Arguments

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
<th>allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg</td>
<td>the log message</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>newline</td>
<td>should we terminate with a newline</td>
<td>bool</td>
<td>True</td>
</tr>
</tbody>
</table>

Returns

Example

2204
Permissions

- Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

- Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

- Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by AUI.