Arguments

agentid	Agent id Allowed:Int Default: no default

Returns
Bool

---

servers.methods - Function

Package	**utility**
Module	**tasking**
Tool	**servers**

Return the methods of an object

Synopsis
methods(agentid, objectid)

Description
Return the methods of a given object.

Arguments

agentid	Agent id Allowed:Int Default: no default
objectid	Object id Allowed:Int Default: no default

Returns
String

1.9 benchmark - Module

Module for standard AIPS++ performance benchmarks
include "benchmark.g"

Description

`benchmark` provides a standard set of AIPS++ performance benchmarks.

Performance benchmarks

This module defines the standard AIPS++ performance benchmarks. The primary purpose of the `benchmark` tool is to allow routine and automated performance monitoring of representative data reduction threads. This, in turn, allows performance problems due to unrelated code evolution to be identified and corrected early in development.
Performance benchmarks are invariably an imperfect measure of the performance of any software package. They are inherently difficult to define for a number of reasons, including:

Coverage Benchmarks need to cover a set of operations which together are representative of the average use profile of the system. The heterogeneous nature of large radio astronomy reduction packages, including AIPS++, makes this very difficult to define in any unique or absolute sense. This is further compounded by the broad range of applications for the package, including near real-time use at telescopes and use in off-line reduction environments. The benchmarks defined here attempt to address this balance by focusing on the common scientific reduction threads most frequently in use.

Scientific relevance Benchmarks need to be cast in terms of their scientific relevance if they are to be useful. In a pipeline environment, for example, the reduction time of any given thread is most sensibly expressed as the fraction of the observing time of the data under reduction. In an interactive environment, the relevant time-scale is the time it takes to set input parameters for the operation being performed. In either case, there is a point at which performance improvements no longer improve scientific throughput in any meaningful sense. Conversely, expressing benchmarks in terms of their scientific relevance isolates those cases which should be and can be profitably improved.

Dependence Benchmarks typically depend on a large number of parameters, including uv-data set size, image size and deconvolution algorithm type, amongst many others. Scientific throughput is therefore a scalar function over many dimensions. In addition, the constituent components in the overall performance function are often non-linear in any individual parameter (e.g. $O^n(p)$), so the overall performance function cannot be linearly extrapolated from one part of parameter space to another. Any performance benchmark is only sampling the overall performance function at one point in the parameter hyperspace. This reinforces the need for benchmarks to be representative of a broad range of common data reduction threads.

Environment Benchmark measurements may be very sensitive to the immediate configuration of the local computing environment. This includes factors such as disk I/O protocols (SCSI versus IDE), background operating system I/O caching policies, operating system and architecture, networked (NFS) or non-networked disk access, available memory and memory utilization policies in the applications. All influence the measurement obtained for any given benchmark, and need to be carefully controlled for reliable benchmark measurements.

Inter-comparisons Performance comparisons between different software systems are particularly difficult, primarily because it is seldom possible to inter-compare exactly the same operation in each package, due to differences in implementation and approach. In addition, great care needs to be taken in controlling the environment in making such comparisons. Packages which have small memory models are difficult to compare with packages which have large memory models, for example, given the substantial difference in access times for main memory and disk. The benchmarks defined here do however include scripts for comparable (or similar) operations in other packages in order to provide an approximate estimate of relative performance. However such numbers are invariably highly unreliable for the reasons stated here.

Overview of `benchmark` tool functions:

This section contains a list of the `benchmark` tool functions by category.

All benchmarks all

Clark CLEAN single-field imaging benchmarks ccsf ccsfvl4uc

Example
The following example shows how to run all benchmarks.

```
include 'benchmark.g'
#
# Create a benchmark tool
#
bench:=benchmark();
#
# Run all defined benchmarks
#
bench.all();
#
# Close the benchmark tool
#
bench.done()
```

Description of individual benchmarks

Each benchmark is identified by a unique benchmark code. These are hierarchical, and specify all relevant parameters applicable to the benchmark. This section includes any scripts from other packages which perform the same reduction. Package inter-comparison is subject to the caveats mentioned above however.

Clark Clean single-field imaging

This benchmark has the following code descriptor:

$$\text{CC-SF-}data\text{−SP}win - stokes - weight - npixel - nclean$$

where *data* is the input data description, further broken down as *data*=[*instrument*][*size*][*compression*][*obsmode*]. The parameters are specified as follows: i) *instrument* - telescope abbreviation (VL=VLA); ii) *size* - dataset size on an integer scale of 1 to 10 (4 $\sim$ 630k UVFITS visibilities); iii) *compression* - MS compression (U=uncompressed; C=compressed); iv) *obsmode* - observing mode (C=continuum, L=line); v) *win* - number of spectral windows imaged; vi) *stokes* - Stokes parameters imaged (e.g. IQUV or I); vii) *weight* - imaging weight (UN=uniform, NA=natural); viii) *npixel* - image size in pixels; vix) *nclean* - number of Clean components in deconvolution.
The individual Clark Clean single-field benchmarks implemented in the default run of the *ccsf*() method at present are enumerated in the tables below. The common portion of benchmark code mentioned in the table caption. The variable part of the code is listed in the "Benchmark" column of the tables. Full benchmark code can be constructed by concatenating the constant and the variable part of the code

The corresponding AIPS script for these benchmarks is as follows. The environment variable `DATA` must be set to the point to the directory where the benchmark related data files are kept (`/aips++/data/demo/benchmark` by default). Note that since AIPS++ datafile names have lower case letters, you will have either make a symbolic link (speaking for UNIX only) or copy the relevant data files to a files with all capital letters. This is required since AIPS internally converts all letters to upper case.
`<FITSFILE>` in the following script is the name of the UV FITS file to be loaded (E.g. VLAC125K.FITS).
`<N>` is the AIPS catalog number corresponding to the UVDATA file loaded using `FITLD`.
`<NPIX>` is the size of the image in pixels.
`<STOKES>` is 'I', 'Q', 'U', or 'V'. `IMAGR` task in AIPS accepts a single stokes value per execution.
`<WEIGHT>` is 'U' for uniform weighting or 'N' for natural weighting.
`<NITER>` is the number of Clean iterations required.

```
TASK 'FITLD'
```

Table 1.862: Table for continuum VLA 125K rows dataset benchmark. Common part of the benchmark code: 'CC-SF-VLAC-U125K-SP1-'

Benchmark code	Dataset	N_{spw}	Stokes	Wgt.	N_{pixel}	N_{chan}	N_{clean}
I-UN-512-C1-1000	vlac125K	1	I	UN	512	1	1000
IQUV-UN-C1-512-1000	vlac125K	1	IQUV	UN	512	1	1000
I-UN-1024-C1-1000	vlac125K	1	I	UN	1024	1	1000
IQUV-UN-1024-C1-1000	vlac125K	1	IQUV	UN	1024	1	1000
I-UN-2048-C1-1000	vlac125K	1	I	UN	2048	1	1000
IQUV-UN-2048-C1-1000	vlac125K	1	IQUV	UN	2048	1	1000
I-NA-512-C1-1000	vlac125K	1	I	NA	512	1	1000
IQUV-NA-512-C1-1000	vlac125K	1	IQUV	NA	512	1	1000
I-NA-1024-C1-1000	vlac125K	1	I	NA	1024	1	1000
IQUV-NA-1024-C1-1000	vlac125K	1	IQUV	NA	1024	1	1000
I-NA-2048-C1-1000	vlac125K	1	I	NA	2048	1	1000
IQUV-NA-2048-C1-1000	vlac125K	1	IQUV	NA	2048	1	1000

Table 1.863: Table for continuum VLA 1M rows dataset benchmark. Common part of the benchmark code: 'CC-SF-VLAC-U1M-SP1-'

Benchmark code	Dataset	N_{spw}	Stokes	Wgt.	N_{pixel}	N_{chan}	N_{clean}
I-UN-512-C1-1000	vlac1M	1	I	UN	512	1	1000
IQUV-UN-512-C1-1000	vlac1M	1	IQUV	UN	512	1	1000
I-UN-1024-C1-1000	vlac1M	1	I	UN	1024	1	1000
IQUV-UN-1024-C1-1000	vlac1M	1	IQUV	UN	1024	1	1000
I-UN-2048-C1-1000	vlac1M	1	I	UN	2048	1	1000
IQUV-UN-2048-C1-1000	vlac1M	1	IQUV	UN	2048	1	1000
I-NA-512-C1-1000	vlac1M	1	I	NA	512	1	1000
IQUV-NA-512-C1-1000	vlac1M	1	IQUV	NA	512	1	1000
I-NA-1024-C1-1000	vlac1M	1	I	NA	1024	1	1000
IQUV-NA-1024-C1-1000	vlac1M	1	IQUV	NA	1024	1	1000
I-NA-2048-C1-1000	vlac1M	1	I	NA	2048	1	1000
IQUV-NA-2048-C1-1000	vlac1M	1	IQUV	NA	2048	1	1000

Table 1.864: Table for spectral line VLA 125K rows, 64 channel dataset benchmark. Common part of the benchmark code: 'CC-SF-VLAL-U125K-SP1-'

Benchmark code	Dataset	N_{spw}	Stokes	Wgt.	N_{pixel}	N_{chan}	N_{clean}
I-UN-512-C64-1000	vlac125K	1	I	UN	512	64	1000
I-UN-1024-C64-1000	vlac125K	1	I	UN	1024	64	1000
I-UN-2048-C64-1000	vlac125K	1	I	UN	2048	64	1000
I-NA-512-C64-1000	vlac125K	1	I	NA	512	64	1000
I-NA-1024-C64-1000	vlac125K	1	I	NA	1024	64	1000
I-NA-2048-C64-1000	vlac125K	1	I	NA	2048	64	1000

Table 1.865: Table for continuum BIMA 139M rows dataset benchmark. Common part of the benchmark code: 'CC-SF-BIMAL-U139M-SP1-'

Benchmark code	Dataset	N_{spw}	Stokes	Wgt.	N_{pixel}	N_{chan}	N_{clean}
I-NA-256-C20-1000	BIMADATA	1	I	NA	256	20	1000

```
INFILE 'DATA:<FITSFILE>'

GO FITLD

TASK 'IMAGR'
GETN        <N>
INSEQ        1
INDISK       1
SOURCES     ' '
QUAL        -1
CALCODE     ' '
TIMERANG     0
SELBAND     -1
SELFREQ     -1
FREQID       1
SUBARRAY     0
DOCALIB     -1
GAINUSE      0
DOPOL       -1
BLVER       -1
FLAGVER      0
DOBAND      -1
BPVER       -1
SMOOTH       0
STOKES    <STOKES>
```

```
BCHAN          1
ECHAN          1
CHANNEL        0
NPOINTS        1
CHINC          1
BIF            1
EIF            1
OUTNAME       ' '
OUTDISK        1
OUTSEQ         0
OUTVER         0
IN2NAME       ' '
IN2CLASS      ' '
IN2SEQ         0
IN2DISK        0
CELLSIZE       5,5
IMSIZE      <NPIX>, <NPIX>
NFIELD         1
DO3DIMAG      -1
FLDSIZE        0
RASHIFT        0
DECSHIFT       0
UVTAPER        0
UVRANGE        0
GUARD          0
ROTATE         0
ZEROSP         0
UVWTFN      <WEIGHT>
UVSIZE         0
ROBUST         0
UVBOX          0
UVBXFN         1
XTYPE          5
YTYPE          5
XPARM          0
YPARM          0
NITER       <NITER>
BCOMP          0
ALLOKAY        0
NBOXES         0
CLBOX          0
BOXFILE       ' '
OBOXFILE      ' '
GAIN           0.1
FLUX           0
MINPATCH      51
```

Table 1.866: Table for VLA calibrator benchmark (for 'G' and 'D' Jones).

Benchmark code	Data	Compressed?	Jones	NAnt	SNR	NSolInt
CALVLAU-G-27-10-100	calvlac27s10.fits	Nope	G	27	10	100
CALVLAU-D-27-10-100	calvlac27s10.fits	Nope	D	27	10	100

```
BMAJ          0
BMIN          0
BPA           0
OVERLAP       0
PHAT          0
FACTOR        0
CMETHOD      ' '
IMAGRPRM      0
NGAUSS        0
WGAUSS        0
FGAUSS        0
MAXPIXEL      0
DOTV         -1

GO IMAGR
```

The equivalent script for spectral line benchmarks would set the keywords `BCHAN=C0`, `ECHAN=C1` where `C0` and `C1` are the start and end frequency channel numbers. This will generate a spectral cube. For multi-frequency synthesis, `NPOINTS` must be set to a value `>=C1-C0+1` (this will grid all the specified channels on a single uv-grid).

VLA G and D term calibration

This benchmark has the following code descriptor:

CALVLA*COMPRESSION*-*JONES*-*NAnt*-*SNR*-*NSolInt*

where *COMPRESSION* is the compression state of the input data (compressed or uncompressed), *JONES* is the name of the Jones matrix being solved for ('G', 'D', or 'B'), *NAnt* is the total number of antennas in the data, *SNR* the signal-to-noise ratio of the input data, and *NSolInt* is the number of solution intervals involved.
The individual calibrator benchmarks implemented in the default run of the *calvla*() method at present are enumerated in the Table 1.866.

The corresponding AIPS script for these benchmarks is as follows. The environment variable `DATA` must be set to the point to the directory where the benchmark related data files are kept (`/aips++/data/demo/benchmark` by default). Note that since AIPS++ datafile names have lower case letters, you

will have either make a symbolic link (speaking for UNIX only) or copy the relevant data files to a files with all capital letters. This is required since AIPS internally converts all letters to upper case.
`<FITSFILE>` in the following script is the name of the UV FITS file to be loaded (E.g. CALVLAU27S10.FITS).
`<N>` is the AIPS catalog number corresponding to the UVDATA file loaded using `FITLD`.
Following is the script for solve for the antenna based complex gain terms (GJones) as a function of time.

```
TASK 'FITLD'
INFILE 'DATA:<FITSFILE>'

GO FITLD
```

This must be followed by a few, simple operations to get the UV database inside AIPS in proper order. The AIPS++ generated FITS files, though are in Time-baseline order naturally, don't seem to indicate so via whatever keyword AIPS looks for. Hence, one needs to run the AIPS task `UVSRT` to write out a Time-baseline (`TB`) sorted UV database as follows:

```
TASK 'UVSRT'
GETN        <N>
SORT        'TB'
GO
```

Next, since the simulated database is a single source database, it does not have the equivalent of the AIPS default `CL` table. This is required by `PCAL` later. To generate the blank `CL` table, one needs to convert the database into a multi-source database (even though it has only one source in it). This is done by running the AIPS task `MULTI`.

```
TASK 'MULTI'

GETN        <N>
GO
```

Next, run the AIPS task `INDXR` as follows:

```
TASK 'INDXR'

GETN        <N>
INFILE      ' '
PRTLEV       0
CPARM        0
BPARM        0
IN2FILE    ' '
GO
```

Finally, run the task SETJY to put the source full stokes flux density in the AIPS SU tables.

```
TASK 'INDXR'

GETN        <N>
SOURCES     ' '
QUAL         -1
BIF           0
EIF           0
ZEROSP       14.76,0.66,1.48,0
OPTYPE      ' '
CALCODE     ' '
SYSVEL        0
RESTFREQ      0 0
VELTYP      ' '
VELDEF      ' '
FREQID       -1
APARM         0
GO
```

The data is now ready to run the tasks CALIB (for the equivalent of GJones calibration) and PCAL (for the equivalent of DJones calibration).

```
TASK 'CALIB'
GETN         <N>
CALSOUR       ' '
QUAL         -1
CALCODE       '    '
SELBAND       0
SELFREQ       0
FREQID        0
TIMERANG      0
BCHAN         64
ECHAN         64
ANTENNAS      0
DOFIT         0
ANTUSE        0
SUBARRAY      0
UVRANGE       0  0
WTUV          0
DOCALIB      -1
GAINUSE       0
FLAGVER       0
DOBAND       -1
BPVER        -1
SMOOTH        0
```

```
IN2NAME      '     '
IN2CLASS     '     '
IN2SEQ        0
IN2DISK       0
INVERS        0
NCOMP         0
FLUX          0
NMAPS         0
CMETHOD      '    '
CMODEL       '    '
SMODEL       0
OUTNAME      '    '
OUTCLASS     '    '
OUTSEQ        0
OUTDISK       1
REFANT        0
SOLINT        1.16 #to get 100 solution intervals for the checked-in benchmark dataset
APARM         0
SOLTYPE      '    '
SOLMODE      'A&P'
SOLCON        0
MINAMPER      0
MINPHSER      0
CPARM         0
SNVER         0
ANTWT         0
GAINERR       0
BADDISK       0

GO CALIB
```

Following is the script for solve for the antenna based polarization leakage term.

```
TASK 'PCAL'
GETN         <N>
CALSOUR      ' '
TIMERANG      0
SELBAND      -1
SELFREQ      -1
FREQID       -1
BIF           0
EIF           0
ANTENNAS      0
UVRANGE       0  0
SUBARRAY      0
```

```
FLAGVER      0
DOCALIB      1
GAINUSE      0
IN2NAME     ' '
IN2CLASS    ' '
IN2SEQ       0
IN2DISK      0
INVERS       0
NCOMP        0
FLUX         0
NMAPS        0
PMODEL       0
SOLINT       0
SOLTYPE     'APPR'
PRTLEV       0
REFANT       0
BPARM        0
CPARM        0
BADDISK      0

GO PCAL
```

Importing visibility data from FITS files
This benchmark has the following code descriptor:

FUV-RD-*data*

where *data* is the input data description, further broken down as *data*=[*instrument*][*size*][*compression*][*obsmode*][*nchan*]. The parameters are specified as follows: i) *instrument* - telescope abbreviation ii); *size* - dataset size on an integer scale of 1 to 10 (4 $\sim$ 630k UVFITS visibilities); iii) *compression* - MS compression (U=uncompressed; C=compressed); iv) *obsmode* - observing mode (C=continuum, L=line); v) *nchan* - number of frequency channels.
The individual FITS Filler benchmarks implemented in the default run of the *fuvrd*() method at present are enumerated in Table 1.867.

Tools

atmosphere	Tool for calculating atmospheric effects in the mm regime
benchmark	tool for standard AIPS++performance benchmarks

1.9.1 benchmark - Tool

Package utility
Module benchmark

Table 1.867: Table for FITS UV read benchmark (import visibility data into AIPS++ from FITS files). Common part of the code is 'FUV-RD-VLA'.

Benchmark code	Dataset	Mode	N_{chans}	Compressed?	Data size
C-U125K-C1	vlac125k.fits	Continuum	1	False	125K
C-U1M-C1	vlac1m.fits	Continuum	1	False	1M
L-U125K-C64	vlal125K64Chan.fits	Line	64	False	125K

tool for standard AIPS++performance benchmarks

Description
The `benchmark` tool provides access to the standard AIPS++ performance benchmarks. A `benchmark` tool is created without any user-specified parameters.

Constructors

benchmark	Construct an benchmark tool

Functions

all	Run all defined benchmarks
almafuvrd	Run all ALMA-TI almati2ms FITS UV read benchmarks
almatifuvrd	Run the almati2ms FITS Filler, IRAM data, ALMA-TI format, uncompressed benchmarks
calvla	Run all default VLA calibrator benchmarks
calvlau27s10	Run the VLA calibrator benchmarks for G and D Jones
ccsf	Run all Clark Clean, single-field benchmarks
ccsfbi4nl	Run the Clark Clean, single-field, BIMA, 139M vis, uncompressed, continuum benchmarks
ccsfvlacu125k	Run the Clark Clean, single-field, VLA, 125k vis, uncompressed, continuum benchmarks
ccsfvlacu1m	Run the Clark Clean, single-field, VLA, 1M vis, uncompressed, continuum benchmarks
ccsfvlalu125k	Run the Clark Clean, single-field, VLA, 125K vis, 64 frequency channel, uncompressed, spectral
done	Terminate the benchmark process
fuvrd	Run all FITS UV read benchmarks
fuvrdvlacu125k	Run the FITS Filler, VLA, 125k vis, uncompressed, continuum benchmarks
fuvrdvlacu1m	Run FITS Filler, VLA, 1M vis, uncompressed, continuum benchmarks
fuvrdvlalu125k	Run the FITS Filler VLA, 125K vis, 64 frequency channel, uncompressed, spectral line benchma
sdall	Run the single dish benchmark tests
sdbench	Run the single dish benchmark tests for a particular data set

benchmark - Constructor

Package	**utility**
Module	**benchmark**
Tool	**benchmark**

Construct an benchmark tool

Synopsis
benchmark(datarepospath)

Description

This is used to construct a `benchmark` tool. This tool can then be used to run individual benchmarks or all defined benchmarks.
It can be used as `benchmark()` to construct the benchmark tool with the data repository set to the standard AIPS++ directory.
This returns a Glish variable containing the tool functions of `benchmark`.

Returns
benchmark tool

Arguments

datarepospath	Path for the test data repository Allowed:String Default: '$AIPSROOT/data/demo/benchmark'

Example

```
bench:=benchmark()
bench.all()
bench.done()
```

---

benchmark.all - Function

Package	**utility**
Module	**benchmark**
Tool	**benchmark**

Run all defined benchmarks

Synopsis
all()

Description
Run all benchmarks currently defined.

Arguments

Returns
Bool

Example

```
bench:=benchmark()
bench.all()
bench.done()
```

---

benchmark.done - Function

Package	**utility**
Module	**benchmark**
Tool	**benchmark**

Terminate the benchmark process

Synopsis
done()

Description
This is used to totally stop the `benchmark` process. It is a good idea to conserve memory use on your machine by stopping the process once you no longer need it.

Returns
Bool

Example

```
bench:=benchmark()
bench.all()
bench.done()
```

---

benchmark.ccsf - Function

Package	**utility**
Module	**benchmark**
Tool	**benchmark**

Run all Clark Clean, single-field benchmarks

Synopsis
ccsf()

Description
This function will run all Clark Clean, single-field imaging benchmarks which are defined.

Returns
Bool

Example

```
bench:=benchmark()
bench.ccsf()
bench.done()
```

---

benchmark.ccsfvlacu125k - Function

Package	**utility**
Module	**benchmark**
Tool	**benchmark**

Run the Clark Clean, single-field, VLA, 125k vis, uncompressed, continuum benchmarks

Synopsis
ccsfvlacu125k(imsizes, stokes, weight, spwid, niter, mode, nchan, start, step, fieldid, facets, tile, cache)

Description
Run the Clark Clean, single-field imaging benchmarks defined for the VLA dataset of 125k uncompressed continuum visibilities.

Returns
Bool

Arguments

imsizes	List of image sizes to be used for benchmarking Allowed:Vector of ints Default: [512,1024,2048]
stokes	List of stokes parameters to be used for benchmarking Allowed:Vector of 'I' and/or 'IQUV' Default: ['I','IQUV']
weight	List of weighting schemes to be used for benchmarking Allowed:Vector of 'uniform' and/or 'natural' Default: ['natural','uniform']
spwid	List of spectral windows to be used for benchmarking Allowed:Vector of ints Default: [1]
niter	The number of CLEAN iterations to be used Allowed:Integer Default: 1000
mode	Imaging mode Allowed:'mfs' or 'channel' Default: 'mfs'
nchan	Number of frequency channels to use Allowed:Integer Default: 1
start	Start channel number Allowed:Integer Default: 1
step	Step to use for stepping through channels Allowed:Integer Default: 1
fieldid	Field ID parameter of imager Allowed:Integer Default: 1
facets	Number of facets to be used for imaging Allowed:Integer Default: 1
tile	Tile size used internally by imager for PagedArrays Allowed:Integer Default: 16
cache	The cache size internally by imager for PagedArrays Allowed:Integer Default: 4194304 (4MB)

Example

```
bench:=benchmark()
bench.ccsfvlacu125k()
bench.done()
```

---

benchmark.ccsfvlacu1m - Function

Package	**utility**
Module	**benchmark**
Tool	**benchmark**

Run the Clark Clean, single-field, VLA, 1M vis, uncompressed, continuum benchmarks

Synopsis
ccsfvlacu1m(imsizes, stokes, weight, spwid, niter, mode, nchan, start, step, fieldid, facets, tile, cache)

Description
Run the Clark Clean, single-field imaging benchmarks defined for the VLA dataset of 1M uncompressed continuum visibilities.

Returns
Bool

Arguments

imsizes	List of image sizes to be used for benchmarking Allowed:Vector of ints Default: [512,1024,2048]
stokes	List of stokes parameters to be used for benchmarking Allowed:Vector of 'I' and/or 'IQUV' Default: ['I','IQUV']
weight	List of weighting schemes to be used for benchmarking Allowed:Vector of 'uniform' and/or 'natural' Default: ['natural','uniform']
spwid	List of spectral windows to be used for benchmarking Allowed:Vector of ints Default: [1]
niter	The number of CLEAN iterations to be used Allowed:Integer Default: 1000
mode	Imaging mode Allowed:'mfs' or 'channel' Default: 'mfs'
nchan	Number of frequency channels to use Allowed:Integer Default: 1
start	Start channel number Allowed:Integer Default: 1
step	Step to use for stepping through channels Allowed:Integer Default: 1
fieldid	Field ID parameter of imager Allowed:Integer Default: 1
facets	Number of facets to be used for imaging Allowed:Integer Default: 1
tile	Tile size used internally by imager for PagedArrays Allowed:Integer Default: 16
cache	The cache size internally by imager for PagedArrays Allowed:Integer Default: 4194304 (4MB)

Example

```
bench:=benchmark()
bench.ccsfvlacu1m()
```

```
bench.done()
```

---

benchmark.ccsfvlalu125k - Function

Package	**utility**
Module	**benchmark**
Tool	**benchmark**

Run the Clark Clean, single-field, VLA, 125K vis, 64 frequency channel, uncompressed, spectral line benchmarks

Synopsis
ccsfvlalu125k(imsizes, stokes, weight, spwid, niter, mode, nchan, start, step, fieldid, facets, tile, cache)

Description
Run the Clark Clean, single-field imaging benchmarks defined for the VLA dataset of 125K uncompressed spectral line visibilities.

Returns
Bool

Arguments

imsizes	List of image sizes to be used for benchmarking Allowed:Vector of ints Default: [512]
stokes	List of stokes parameters to be used for benchmarking Allowed:Vector of 'I' and/or 'IQUV' Default: ['I']
weight	List of weighting schemes to be used for benchmarking Allowed:Vector of 'uniform' and/or 'natural' Default: ['natural','uniform']
spwid	List of spectral windows to be used for benchmarking Allowed:Vector of ints Default: [1]
niter	The number of CLEAN iterations to be used Allowed:Integer Default: 1000
mode	Imaging mode Allowed:'mfs' or 'channel' Default: 'channel'
nchan	Number of frequency channels to use Allowed:Integer Default: 64
start	Start channel number Allowed:Integer Default: 1
step	Step to use for stepping through channels Allowed:Integer Default: 1
fieldid	Field ID parameter of imager Allowed:Integer Default: 1
facets	Number of facets to be used for imaging Allowed:Integer Default: 1
tile	Tile size used internally by imager for PagedArrays Allowed:Integer Default: 16
cache	The cache size internally by imager for PagedArrays Allowed:Integer Default: 4194304 (4MB)

Example

```
bench:=benchmark()
bench.ccsfvlalu125k()
```

```
bench.done()
```

---

benchmark.ccsfbi4nl - Function

Package	**utility**
Module	**benchmark**
Tool	**benchmark**

Run the Clark Clean, single-field, BIMA, 139M vis, uncompressed, continuum benchmarks

Synopsis
ccsfbi4nl(imsizes, stokes, weight, spwid, niter, mode, nchan, start, step, fieldid, facets, tile, cache)

Description
Run the Clark Clean, single-field imaging benchmarks defined for the BIMA dataset of 139M uncompressed continuum visibilities.

Returns
Bool

Arguments

imsizes	List of image sizes to be used for benchmarking Allowed:Vector of ints Default: [256]
stokes	List of stokes parameters to be used for benchmarking Allowed:Vector of 'I' and/or 'IQUV' Default: ['I']
weight	List of weighting schemes to be used for benchmarking Allowed:Vector of 'uniform' and/or 'natural' Default: ['natural']
spwid	List of spectral windows to be used for benchmarking Allowed:Vector of ints Default: [1]
niter	The number of CLEAN iterations to be used Allowed:Integer Default: 1000
mode	Imaging mode Allowed:'mfs' or 'channel' Default: 'channel'
nchan	Number of frequency channels to use Allowed:Integer Default: 20
start	Start channel number Allowed:Integer Default: 40
step	Step to use for stepping through channels Allowed:Integer Default: 1
fieldid	Field ID parameter of imager Allowed:Integer Default: 1
facets	Number of facets to be used for imaging Allowed:Integer Default: 1
tile	Tile size used internally by imager for PagedArrays Allowed:Integer Default: 16
cache	The cache size internally by imager for PagedArrays Allowed:Integer Default: 4194304 (4MB)

Example

```
bench:=benchmark()
bench.ccsfbi4nL()
```

```
bench.done()
```

---

benchmark.fuvrd - Function

Package	**utility**
Module	**benchmark**
Tool	**benchmark**

Run all FITS UV read benchmarks

Synopsis
fuvrd()

Description
This function will run all FITS UV read benchmarks which are defined.

Returns
Bool

Example

```
bench:=benchmark()
bench.fuvrd()
bench.done()
```

---

benchmark.fuvrdvlacu125k - Function

Package	**utility**
Module	**benchmark**
Tool	**benchmark**

Run the FITS Filler, VLA, 125k vis, uncompressed, continuum benchmarks

Synopsis
fuvrdvlacu125k(fitsin, compress)

Description
Run the FITS Filler benchmarks defined for the VLA dataset of 125k uncompressed continuum visibilities.

Returns
Bool

Arguments

fitsin	FITS file name Allowed:string Default: vlac125k.fits
compress	MS compression mode Allowed:Bool Default: F

Example

```
bench:=benchmark()
bench.fuvrdvlacu125k()
bench.done()
```

---

benchmark.fuvrdvlacu1m - Function

Package	**utility**
Module	**benchmark**
Tool	**benchmark**

Run FITS Filler, VLA, 1M vis, uncompressed, continuum benchmarks

Synopsis
fuvrdvlacu1m(fitsin, compress)

Description
Run FITS Filler benchmarks defined for the VLA dataset of 1M uncompressed continuum visibilities.

Returns

Bool

Arguments

fitsin	FITS file name Allowed:string Default: vlac1M.fits
compress	MS compression mode Allowed:Bool Default: F

Example

```
bench:=benchmark()
bench.fuvrdvlacu1m()
bench.done()
```

---

benchmark.fuvrdvlalu125k - Function

Package	**utility**
Module	**benchmark**
Tool	**benchmark**

Run the FITS Filler VLA, 125K vis, 64 frequency channel, uncompressed, spectral line benchmarks

Synopsis
fuvrdvlalu125k(fitsin, compress)

Description
Run the FITS Filler benchmarks defined for the VLA dataset of 125K uncompressed spectral line visibilities with 65 channels.

Returns
Bool

Arguments

fitsin	FITS file name Allowed:string Default: vlal125K64Chan.fits
compress	MS compression mode Allowed:Bool Default: F

Example

```
bench:=benchmark()
bench.fuvrdvlalu125k()
bench.done()
```

---

benchmark.almafuvrd - Function

Package	**utility**
Module	**benchmark**
Tool	**benchmark**

Run all ALMA-TI almati2ms FITS UV read benchmarks

Synopsis
almafuvrd()

Description
This function will run all almati2ms ALMA-TI FITS UV read benchmarks which are defined.

Returns
Bool

Example

```
bench:=benchmark()
bench.almafuvrd()
bench.done()
```

---

benchmark.almatifuvrd - Function

Package	**utility**
Module	**benchmark**
Tool	**benchmark**

Run the almati2ms FITS Filler, IRAM data, ALMA-TI format, uncompressed benchmarks

Synopsis
almatifuvrd(fitsin, compress)

Description
Run the almati2ms FITS Filler benchmarks defined for an IRAM dataset presented in ALMA-TI format. Expects to find dataset in /aips++/data/alma/test.

Returns
Bool

Arguments

fitsin	FITS file name Allowed:string Default: 07-feb-1997-g067-04.fits
compress	MS compression mode Allowed:Bool Default: F

Example

```
bench:=benchmark()
bench.almatifuvrd()
bench.done()
```

---

benchmark.calvla - Function

Package	**utility**
Module	**benchmark**
Tool	**benchmark**

Run all default VLA calibrator benchmarks

Synopsis
calvla()

Description
This function will run all the default VLA calibrator benchmarks.

Returns
Bool

Example

```
bench:=benchmark()
bench.calvla()
bench.done()
```

---

benchmark.calvlau27s10 - Function

Package	**utility**
Module	**benchmark**
Tool	**benchmark**

Run the VLA calibrator benchmarks for G and D Jones

Synopsis
calvlau27s10(datafile, myms, jones, nsolint, nchan, start, step, tablename)

Description
Run the calibrator benchmarks defined for VLA uncompressed dataset with 27 antennas and 128 frequency channels.

Returns
Bool

Arguments

datafile	FITS file name Allowed:string Default: calvlac27s10.fits
myms	Name of the MS. Will bypass the fitstoms() operation Allowed:string Default: ' '
jones	List of Jones matrices to be solved for Allowed:array of strings Default: ['G','D']
nsolint	No. of solution intervals Allowed:integer Default: 100
nchan	No. of frequency channels to be used Allowed:integer Default: 1
start	The starting frequency channel number Allowed:integer Default: 1
step	Increment for iterating through frequency channels Allowed:integer Default: 1
tablename	Name of a calibration table to be applied before the solver is invoked Allowed:string Default: "

Example

```
bench:=benchmark()
bench.calvlac27s10()
bench.done()
```

---

benchmark.sdall - Function

Package	**utility**
Module	**benchmark**
Tool	**benchmark**

Run the single dish benchmark tests

Synopsis
sdall()

Description
Run the single dish benchmark tests. This uses GBT data to test the timings of the import, scan access, scan plotting, calibration, averaging and baselining.

Returns
Bool

Example

```
bench:=benchmark()
bench.sdall()
bench.done()
```

---

benchmark.sdbench - Function

Package	**utility**
Module	**benchmark**
Tool	**benchmark**

Run the single dish benchmark tests for a particular data set

Synopsis
sdbench(code)

Description
Run the single dish benchmark tests for a specific data set. The available codes and matching data are:

- 42256 - Spectral Processor data - [17,18] - OffOn PSWITCH
- 121k - Spectral Processor data - [36,37] - Track FSWITCH
- 114k - Spectrometer data - [20,21] - OffOn PSWITCH
- 224k - Spectrometer data - [43,44] - OffOn PSWITCH
- 128k - Spectrometer data - [48,48] - Track FSWITCH

- 1216k - Spectrometer data - [22,23] - OffOn PSWITCH
- 12125k- Spectrometer data - [51,52] - OffOn PSWITCH

The code delineates:

- first number: number of IFs (1, 2 or 4)
- second number: number of polarizations (1 or 2)
- third number: number of channels (256, 1k, 4k, 8k, 16k, 125k)

Returns
Bool

Arguments

code	code for test Allowed:string Default: none (see above)

Example

```
bench:=benchmark()
bench.sdbench('42256')
bench.done()
```

1.9.2 atmosphere - Tool

Package utility
Module benchmark

Tool for calculating atmospheric effects in the mm regime
include "atmosphere.g"

Description

`atmosphere`, for now, does calculations of atmospheric profile, the integrated opatical depth on each band, the integrated optical depth for each channel of each band, the absorption coefficients, the dispersive and non-dispersive phase delay factor, the sky brightness, etc, based on a model of the atmosphere. There are five atmosphere models: tropical, mid latitude summer, mid latitude winter, subarctic summer, subarctic winter. This Distributed Object integrates the atmospheric model of Juan R. Pardo, written in FORTRAN, into the AIPS++ code base. This model, which is also used by the ALMA TELCAL group, is important for accurate simulation of millimeter observations with ALMA. The tool has to be constructed with the altitude at site, temperature at site, pressure at site, maximum altitude (top of the modelled atmposphere), humidity, temperature gradient with height (dT/dh),initial pressure step (P[1]-P[0]), pressure multiplicative factor for steps (if named dPm, then (P[i+1]-P[i] = dPm*(P[i]-P[i-1])), scale height for water, and atmospheric type.

Example

The following show the use of the atmosphere tool.

```
include 'atmosphere.g'
#
# Wait for results before proceeding to the next step
#
#
# Make an atmosphere tool
#

myatm:= atmosphere(altitude='2550.m', temperature='270.32K',
                   pressure='73585Pa', maxAltitude = '45000m',
                   humidity=20, dTem_dh='-0.0056.K/m',
                dP='500Pa', dPm=1.25, h0='2000.m',atmtype=2);

#
# get startup water content
```

```
#
water := myatm.getStartupWaterContent()
print spaste('Guessed water content is:   ', water)

Guessed water content is:   [value=0.00160052609, unit=m]

#
# get profile
#
myatm.getProfile( thickness=thickness, temperature=temperature,
                  water=water, pressure=pressure,
                  O3=O3, CO=CO, N2O=N2O )
print spaste( 'thickness = ', thickness )
print spaste( '(thickness.value)[1],thickness.unit = ',
               (thickness.value)[1], thickness.unit )

thickness = [value=[54.2820552 68.3014666 86.0898268 108.749638
 137.76137 175.148792 223.740915 287.605456 372.804489 488.798233
 651.269636 888.430939 1257.26693 1895.2737 3257.98328 1750 2000
 2000 3000 3000 3000 4250 4500 4500] , unit=m]

(thickness.value)[1],thickness.unit = 54.2820552m
#
# initialize window
#
nbands := 2;
for ( i in 1:nbands) {
     (fCenter.value)[i] := 88.e9;
     (fWidth.value)[i] := 5.e8;
     (fRes.value)[i] := (fWidth.value)[1]/4.0;   # 4 channels
}

fCenter.unit := 'Hz';
fWidth.unit := 'Hz';
fRes.unit := 'Hz';

myatm.initWindow(nbands=nbands,fCenter=fCenter,fWidth=fWidth,fRes=fRes )

#
# get the number of channels of ith band
#
n:=myatm.getNdata(iband=1);
print spaste( 'getNdata() = ', n );

getNdata() = 4
#
```

```
# get the integrated optical depth on each band
#
myatm.getOpacity( dryOpacity=dryOpacity, wetOpacity=wetOpacity);
print spaste('dryOpacity = ', dryOpacity );
print spaste('wetOpacity = ', wetOpacity );

dryOpacity = [0.0181100218 0.0181100218]
wetOpacity = [value=[0.00665036438 0.00665036438] , unit=mm-1]
#
# get the integrated optical depth for each channel of each band
#
myatm.getOpacitySpec( dryOpacitySpec=dryOS, wetOpacitySpec=wetOS );
print spaste( 'dryOpacitySpec = ', dryOS );
print spaste( 'wetOpacitySpec = ', wetOS );

dryOpacitySpec = [[1:2,]
    0.018275032 0.0181644629 0.0180543786 0.0179462136
    0.018275032 0.0181644629 0.0180543786 0.0179462136]
wetOpacitySpec = [value=[[1:2,]
    0.0066217328 0.0066217328 0.0066217328 0.0066217328
    0.0066217328 0.0066217328 0.0066217328 0.0066217328], unit=mm-1]
#
# get absorption coefficients
#
myatm.getAbsCoeff( kH2OLines=kH2OLines, kH2OCont=kH2OCont, kO2=kO2,
                   kDryCont=kDryCont, kO3=kO3, kCO=kCO, kN2O=kN2O )
print spaste('(kH2OLines.value[1],kH2OLines.unit) = ',
              (kH2OLines.value)[1], kH2OLines.unit );

(kH2OLines.value[1],kH2OLines.unit) = 0.000105215009m-1
#
# get absorption coefficient derivatives
#
myatm.getAbsCoeffDer( kH2OLinesDer=kH2OLinesDer,
                      kH2OContDer=kH2OContDer,kO2Der=kO2Der,
                      kDryContDer=kDryContDer, kO3Der=kO3Der,
                      kCODer=kCODer, kN2ODer=kN2ODer )
print spaste( '(kH2OLinesDer.value[1],kH2OLinesDer.unit) = ',
                kH2OLinesDer.value[1], kH2OLinesDer.unit )

(kH2OLinesDer.value[1],kH2OLinesDer.unit) =  -9.73882945e-08m-1
#
# get dispersive and non-dispersive phase delay factor
#
myatm.getPhaseFactor( dispPhase=dispPhase, nonDispPhase=nonDispPhase );
print spaste( 'dispPhase = ' , dispPhase );
```

```
print spaste( 'nonDispPhase = ', nonDispPhase );

dispPhase = [value=[740.282096 740.282096] , unit=deg.m-1]
nonDispPhase = [value=[1.60255787 1.60255787] , unit=deg.m-1]
#
# computer sky brightness
#
myatm.computeSkyBrightness(airMass=1.51, tbgr='2.73K', precWater='4.05e-3m')
#
# get sky brightness computed by computeSkyBrightness()
#
tBand1 := myatm.getSkyBrightness(iopt=1);
tBand2 := myatm.getSkyBrightness(iopt=2);
print spaste("SkyBrightness for Blackbody = ",tBand1 );
print spaste("SkyBrightness for Rayleigh Jeans = ",tBand2 )

SkyBrightness for Blackbody =[value=[19.9391734 19.9391734] , unit=K]
SkyBrightness for Rayleigh Jeans =[value=[17.9019838 17.9019838] , unit=K]
#
# get sky brightness by integrating the transfer equation for each channel
#
tBandSpec1 := myatm.getSkyBrightnessSpec(iopt=1)
tBandSpec2 := test_atm.getSkyBrightnessSpec(iopt=2)
print spaste('For blackbody, SkyBrightness = ', tBandSpec1)
print spaste('For Rayleigh Jeans,  SkyBrightness = ', tBandSpec2)

For Blackbody, SkyBrightness = [value=[[1:2,]
    19.9541615 19.9439307 19.9339303 19.9246711
    19.9541615 19.9439307 19.9339303 19.9246711], unit=K]
For Rayleigh Jeans, SkyBrightness = [value=[[1:2,]
    17.9210985 17.9081171 17.895366 17.8833538
    17.9210985 17.9081171 17.895366 17.8833538], unit=K]
#
# set sky coupling
#
myatm.setSkyCoupling( c=1.0 )
#
# get sky coupling
#
sc_out := myatm.getSkyCoupling();
print spaste( 'SkyCoupling = ', sc_out )

SkyCoupling = 1.0
```

Constructors

atmosphere Construct an atmosphere tool

Functions

computeSkyBrightness	compute the sky brightness by integrating the transfer equation
getAbsCoeff	get the absorption coefficients
getAbsCoeffDer	get the derivatives of absorption coefficients
getNdata	return the number of channels of ith band
getOapcity	get the integrated optical depth on each band
getOapcitySpec	get the integrated optical depth for each channel of each band
getPhaseFactor	get the dipersive and non-dispersive phase delay factor
getProfile	get atmospheric profile
getSkyBrightness	get sky brightness computed by computeSkyBrightness()
getSkyBrightnessSpec	get sky brightness by integrating the transfer equation for each channel
getSkyCoupling	return the sky coupling
getStartupWaterContent	get the guessed startup water content
initWindow	initialize window
setSkyCoupling	set sky coupling

atmosphere - Constructor

Package	**utility**
Module	**benchmark**
Tool	**atmosphere**

Construct an atmosphere tool

Synopsis
atmosphere(altitude, temperature, pressure, maxAltitude, humidity, dTem_dh, dP, dPm, h0, atmType)

Description
This is used to construct `atmosphere` tool with the condition at the position the user wants to calculate the profile, the integrated opatical depth on each band, the integrated optical depth for each channel of each band, the absorption coefficients, the dispersive and non-dispersive phase delay factor, the sky brightness, etc. This returns a Glish variable containing the tool of atmosphere.

Arguments

altitude	in	Site altitude Allowed:Quantity with units of altitude, meter Default: 2550m
temperature	in	Ambient Temperature Allowed:Quantity with units of temperature, K Default: 270.32K
pressure	in	Ambient pressure Allowed:Quantity with units of pressure, Pascal Default: 73585Pa
maxAltitude	in	altitude of the top pf the modelled atmosphere Allowed:Quantity with dimension of length, meter Default: 45000m
humidity	in	used to guess water Allowed:0 - 100 Default: 20
dTem_dh	in	the derivative of temperature with respect to height Allowed:Quantity with units of K/m Default: -0.0056K/m
dP	in	initial pressure step Allowed:Quantity with the units of pressure, Pa Default: 500Pa
dPm	in	pressure multiplicative factor for steps Allowed:real value Default: 1.25
h0	in	scale height for water(exp distribution) Allowed:Quantity with the dimension of length, meter Default: 2000m
atmType	in	atmospheric type Allowed:1(tropical),2(mid latitude summer),3(mid latitude winter) ,4(subarctic summer),5(subarctic winter), dimensionless Default: 2

Returns
atmosphere tool

Example

```
myatm:= atmosphere(altitude='2550.m', temperature='270.32K',
                   pressure='73585Pa',maxAltitude='45000m',
                   humidity=20, dTem_dh='-0.0056K/m',
                dP='500Pa', dPm=1.25, h0='2000.m',atmtype=2)
```

atmosphere.getStartupWaterContent - Function

Package	**utility**
Module	**benchmark**
Tool	**atmosphere**

get the guessed startup water content

Synopsis
getStartupWaterContent()

Description
Get the guessed startup water content.

Returns
Double, gussed precipitable water content, in meters.

Example

```
water:= myatm.getStartupWaterContent()
print spaste('Guessed water content is:   ', water)

Guessed water content is:   [value=0.00160052609, unit=m]
```

---

atmosphere.getProfile - Function

Package	**utility**
Module	**benchmark**
Tool	**atmosphere**

get atmospheric profile

Synopsis
getProfile(thickness, temperature, water, pressure, O3, CO, N2O)

Description

Get the atmospheric profile.

Arguments

thickness	out	thickness of every atmospheric layer Allowed: Quantum with a vector value and dimension of length, m Default: none
temperature	out	temperature of every atmospheric layer Allowed: Quantum with a vector value and unit of temperature, K Default: none
water	out	water content of every atmospheric layer Allowed: Quantum with a vector value and unit of kg.m-3 Default: none
pressure	out	pressure of every atmospheric layer Allowed: Quantum with a vector value and unit of Pascal Default: none
O3	out	O3 of every atmospheric layer Allowed: Quantum with a vector value and unit of m-3 Default: none
CO	out	CO of every atmospheric layer Allowed: Quantum with a vector value and unit of m-3 Default: none
N2O	out	N2O of every atmospheric layer Allowed: Quantum with a vector value and unit of m-3 Default: none

Example

```
myatm.getProfile( thickness, temperature, water, pressure, O3, CO, N2O )
print spaste( 'thickness = ', thickness )
print spaste( '(thickness.value)[1],thickness.unit = ',
              (thickness.value)[1], thickness.unit )

thickness = [value=[54.2820552 68.3014666 86.0898268 108.749638
 137.76137 175.148792 223.740915 287.605456 372.804489 488.798233
 651.269636 888.430939 1257.26693 1895.2737 3257.98328 1750 2000
 2000 3000 3000 3000 4250 4500 4500] , unit=m]

(thickness.value)[1],thickness.unit = 54.2820552m
```

atmosphere.initWindow - Function

Package	**utility**
Module	**benchmark**
Tool	**atmosphere**

initialize window

Synopsis
initWindow(nbands, fCenter, fWidth, fRes)

Description

function that defines a spectral window, computes absorption and emmision coefficients for this window, using the above atmospheric parameters.

Arguments

nbands	in	number of bands Allowed:Int, standing for the number of bands Default: 1
fCenter	in	frequencies Allowed:Quantum with a vector value and unit of frequency, Hz Default: '88.e9Hz'
fWidth	in	frequency width Allowed:Quantum with a vector value and unit of frequency, Hz Default: '5.0e8Hz'
fRes	in	resolution inside band Allowed:Quantum with a vector value and unit frequency, Hz Default: '1.25Hz'

Example

```
nbands := 2;
for ( i in 1:nbands) {
     (fCenter.value)[i] := 88.e9;
```

```
        (fWidth.value)[i] := 5.e8;
        (fRes.value)[i] := (fWidth.value)[1]/4.0;   # 4 channels
}

fCenter.unit := 'Hz';
fWidth.unit := 'Hz';
fRes.unit := 'Hz';

myatm.initWindow(nbands=nbands,fCenter=fCenter,fWidth=fWidth,fRes=fRes )
```

---

atmosphere.getNdata - Function

Package	**utility**
Module	**benchmark**
Tool	**atmosphere**

return the number of channels of ith band

Synopsis
getNdata(ibands)

Description

Return the number of channels of ith band (passes in as parameter).

Arguments

ibands	in	identifier of bands Allowed:Int, standing for the identifier of bands Default: 1

Returns
Int, the number of channels of the given band.

Example

```
n:=test_atm.getNdata(iband=1)
print spaste( 'getNdata() = ', n )

getNdata() = 4
```

---

atmosphere.getOapcity - Function

Package	**utility**
Module	**benchmark**
Tool	**atmosphere**

get the integrated optical depth on each band

Synopsis
getOapcity(dryOpacity, wetOpacity)

Description

Get the integrated optical depth on each band.

Arguments

dryOpacity	out	dry opacity for each band Allowed:Vector of Double Default: none
wetOpacity	out	wet ipacity for each band Allowed:Quantum with a vector value and unit of mm-1 Default: none

Example

```
myatm.getOpacity( dryOpacity=dryOpacity, wetOpacity=wetOpacity);
print spaste('dryOpacity = ', dryOpacity );
print spaste('wetOpacity = ', wetOpacity );

dryOpacity = [0.0181100218 0.0181100218]
wetOpacity = [value=[0.00665036438 0.00665036438] , unit=mm-1]
```

---

atmosphere.getOapcitySpec - Function

Package	**utility**
Module	**benchmark**
Tool	**atmosphere**

get the integrated optical depth for each channel of each band

Synopsis
getOapcitySpec(dryOpacitySpec, wetOpacitySpec)

Description

Get the integrated optical depth for each channel of each band.

Arguments

dryOpacitySpec	out	dry opacity spectrum for each channel of each band Allowed:Array of Double Default: none
wetOpacitySpec	out	wet ipacity spectrum for each channel of each band Allowed:Quantum with a Array value and unit of mm-1 Default: none

Example

```
myatm.getOpacitySpec( dryOpacitySpec=dryOS, wetOpacitySpec=wetOS );
print spaste( 'dryOpacitySpec = ', dryOS );
print spaste( 'wetOpacitySpec = ', wetOS );

dryOpacitySpec = [[1:2,]
    0.018275032 0.0181644629 0.0180543786 0.0179462136
    0.018275032 0.0181644629 0.0180543786 0.0179462136]
wetOpacitySpec = [value=[[1:2,]
    0.0066217328 0.0066217328 0.0066217328 0.0066217328
    0.0066217328 0.0066217328 0.0066217328 0.0066217328], unit=mm-1]
```

---

atmosphere.getAbsCoeff - Function

Package	**utility**
Module	**benchmark**
Tool	**atmosphere**

get the absorption coefficients

Synopsis
getAbsCoeff(kH2OLines, kH2OCont, kO2, kDryCont, kO3, kCO, kN2O)

Description

Get the absorption coefficients.

Arguments

kH2OLines	out	absorption coefficients for H2O lines Allowed:Quantum with a vector value and unit of m-1 Default: none
kH2OCont	out	absorption coefficients for H2O continum Allowed:Quantum with a vector value and unit of m-1 Default: none
kO2	out	absorption coefficients for O2 Allowed:Quantum with a vector value and unit of m-1 Default: none
kDryCont	out	absorption coefficients for dry continum Allowed:Quantum with a vector value and unit of m-1 Default: none
kO3	out	absorption coefficients for O3 Allowed:Quantum with a vector value and unit of m-1 Default: none
kCO	out	absorption coefficients for CO Allowed:Quantum with a vector value and unit of m-1 Default: none
kN2O	out	absorption coefficients for N2O Allowed:Quantum with a vector value and unit of m-1 Default: none

Example

```
myatm.getAbsCoeff( kH2OLines=kH2OLines,kH2OCont=kH2OCont, kO2=kO2,
                   kDryCont=kDryCont, kO3=kO3, kCO=kCO, kN2O=kN2O )
print spaste('(kH2OLines.value[1],kH2OLines.unit) = ',
              (kH2OLines.value)[1], kH2OLines.unit );

(kH2OLines.value[1],kH2OLines.unit) = 0.000105215009m-1
```

---

atmosphere.getAbsCoeffDer - Function

Package	**utility**
Module	**benchmark**
Tool	**atmosphere**

get the derivatives of absorption coefficients

Synopsis
getAbsCoeffDer(kH2OLinesDer, kH2OContDer, kO2Der, kDryContDer, kO3Der, kCODer, kN2ODer)

Description

Get the derivatives of absorption coefficients.

Arguments

kH2OLinesDer	out	derivatives of absorption coefficients for H2O lines Allowed:Quantum with a vector value and unit of m-1 Default: none
kH2OContDer	out	derivatives of absorption coefficients for H2O continum Allowed:Quantum with a vector value and unit of m-1 Default: none
kO2Der	out	derivatives of absorption coefficients for O2 Allowed:Quantum with a vector value and unit of m-1 Default: none
kDryContDer	out	derivatives of absorption coefficients for dry continum Allowed:Quantum with a vector value and unit of m-1 Default: none
kO3Der	out	derivatives of absorption coefficients for O3 Allowed:Quantum with a vector value and unit of m-1 Default: none
kCODer	out	derivatives of absorption coefficients for CO Allowed:Quantum with a vector value and unit of m-1 Default: none
kN2ODer	out	derivatives of absorption coefficients for N2O Allowed:Quantum with a vector value and unit of m-1 Default: none

Example

```
myatm.getAbsCoeffDer( kH2OLinesDer=kH2OLinesDer,
                      kH2OContDer=kH2OContDer,kO2Der=kO2Der,
                      kDryContDer=kDryContDer, kO3Der=kO3Der,
                      kCODer=kCODer, kN2ODer=kN2ODer )
print spaste( '(kH2OLinesDer.value[1],kH2OLinesDer.unit) = ',
                kH2OLinesDer.value[1], kH2OLinesDer.unit )

(kH2OLinesDer.value[1],kH2OLinesDer.unit) =  -9.73882945e-08m-1
```

---

atmosphere.getPhaseFactor - Function

Package	**utility**
Module	**benchmark**
Tool	**atmosphere**

get the dipersive and non-dispersive phase delay factor

Synopsis
getPhaseFactor(dispPhase, nonDispPhase)

Description

Get the dipersive and non-dispersive phase depay factor.

Arguments

dispPhase	out	dispersive phase delay factor Allowed:Quantum with a vector value and unit of m-1 Default: none
nonDispPhase	out	non-dispersive phase delay factor Allowed:Quantum with a vector value and unit of m-1 Default: none

Example

```
myatm.getPhaseFactor( dispPhase=dispPhase, nonDispPhase=nonDispPhase );
print spaste( 'dispPhase = ' , dispPhase );
print spaste( 'nonDispPhase = ', nonDispPhase );

dispPhase = [value=[740.282096 740.282096] , unit=deg.m-1]
nonDispPhase = [value=[1.60255787 1.60255787] , unit=deg.m-1]
```

atmosphere.computeSkyBrightness - Function

Package	**utility**
Module	**benchmark**
Tool	**atmosphere**

compute the sky brightness by integrating the transfer equation

Synopsis
computeSkyBrightness(airMass, tbgr, precWater)

Description

Compute the sky brightness by integrating the transfer equation.

Arguments

airMass	in	air mass Allowed:Double, dimensionless Default: 1.51
tbgr	in	temperature of cosmic background Allowed:Quantity with unit of temperature, K Default: ’2.73K
precWater	in	precipitable water content Allowed:Quantity with dimension of length, m Default: ’4.05e-3m’

Example

```
myatm.computeSkyBrightness(airMass=1.51, tbgr='2.73K',
                           precWater='4.05e-3m')
```

atmosphere.getSkyBrightness - Function

Package	**utility**
Module	**benchmark**
Tool	**atmosphere**

get sky brightness computed by computeSkyBrightness()

Synopsis
getSkyBrightness(iopt)

Description

Get sky brightness computed by computeSkyBrightness().

Arguments

iopt	in	temperature type Allowed:Int 1 or 2, standing for blackbody temperature or Rayleigh Jeans temperature Default: 1

Returns
Quantum with a vector value and dimension of temperature, K, which is the sky brightness for the given temperature type.

Example

```
tBand1 := myatm.getSkyBrightness(iopt=1);
tBand2 := myatm.getSkyBrightness(iopt=2);
print spaste("SkyBrightness for Blackbody = ",tBand1 );
print spaste("SkyBrightness for Rayleigh Jeans = ",tBand2 )

SkyBrightness for Blackbody =[value=[19.9391734 19.9391734] , unit=K]
SkyBrightness for Rayleigh Jeans =[value=[17.9019838 17.9019838] , unit=K]
```

---

atmosphere.getSkyBrightnessSpec - Function

Package	**utility**
Module	**benchmark**
Tool	**atmosphere**

get sky brightness by integrating the transfer equation for each channel

Synopsis
getSkyBrightnessSpec(iopt)

Description

Get sky brightness by integrating the transfer equation for each channel.

Arguments

iopt	in	temperature type Allowed:Int 1 or 2, standing for blackbody temperature or Rayleigh Jeans temperature Default: 1

Returns
Quantum with a Array value and dimension of temperature, K, which is the sky brightness for the given temperature type.

Example

```
tBandSpec1 := myatm.getSkyBrightnessSpec(iopt=1)
tBandSpec2 := test_atm.getSkyBrightnessSpec(iopt=2)
print spaste('For blackbody, SkyBrightness = ', tBandSpec1)
print spaste('For Rayleigh Jeans,  SkyBrightness = ', tBandSpec2)

For Blackbody, SkyBrightness = [value=[[1:2,]
    19.9541615 19.9439307 19.9339303 19.9246711
    19.9541615 19.9439307 19.9339303 19.9246711], unit=K]
For Rayleigh Jeans, SkyBrightness = [value=[[1:2,]
    17.9210985 17.9081171 17.895366 17.8833538
    17.9210985 17.9081171 17.895366 17.8833538], unit=K]
```

---

atmosphere.setSkyCoupling - Function

Package	**utility**
Module	**benchmark**
Tool	**atmosphere**

set sky coupling

Synopsis
setSkyCoupling(c)

Description

Set sky coupling.

Arguments

c	in	sky coupling Allowed:Float, standing for the sky coupling Default: 1.0

Example

```
myatm.setSkyCoupling( c=1.0 )
```

---

atmosphere.getSkyCoupling - Function

Package	**utility**
Module	**benchmark**
Tool	**atmosphere**

return the sky coupling

Synopsis
getSkyCoupling()

Description

Return the sky coupling.

Returns
Float, the sky coupling.

Example

```
sc_out := myatm.getSkyCoupling();
print spaste( 'SkyCoupling = ', sc_out );

SkyCoupling = 1.0
```