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.