$$
\begin{aligned}
U &= U_0 \left(\frac{\nu}{\nu_0}\right)^\alpha \\
V &= V_0 \left(\frac{\nu}{\nu_0}\right)^\alpha
\end{aligned}
$$

where I_0, Q_0, U_0, V_0 is the flux at the reference frequency (ν_0) and ν is the specified frequency.

Current Capabilities The component models module can currently do the following operations, all of them through the componentlist tool:

1. Create an empty list of components and add or remove user specified components from the list.
2. Create a list of components from an ascii file.
3. Save the list to disk and read it back.
4. Return the flux of the list in any user specified direction.
5. Sort the components using a number of criteria.

In addition there is a component editor that allows the user to view and edit the parameters of a component through a graphical user interface.

See Also

Concepts from the measures module are used throughout this module.

Tools

componenteditor	A graphical user interface for displaying & editing a component.
componentlist	A tool for the manipulation of groups of components

Functions

is_componentlist	Is the argument a componentlist tool?

1.3.1 componentlist - Tool

Package synthesis
Module componentmodels

A tool for the manipulation of groups of components
include "componentlist.g"

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 emptycomponentlist constructor. This creates a componentlist tool that does not contain any components and is not associated with any file.
The alternative is to use either the componentlist or asciitocomponentlist constructors. These create a componentlist tool by reading the data from an AIPS++ table or a properly formatted ascii file.
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.

Constructors

asciitocomponentlist	Create a componentlist from an ascii file
componentlist	Construct an componentlist from the data in an AIPS++ table
emptycomponentlist	Construct an empty componentlist

Functions

add	Add a component to the list.
addcomponent	Add a component to the list
close	Save the componentlist to disk and reset its state.
component	Extract a component from the list.
concatenate	Append components from another componentlist.
convertfluxpol	Change (convert) the polarization representation of the specified components
convertfluxunit	Change (convert) the flux units of the specified components
convertfrequnit	Convert the reference frequency to a new unit
convertrefdir	Convert the reference direction to a new frame
convertshape	Change the units of the shape parameters
deselect	Unmark components in the list
done	Delete the componentlist tool
edit	Start up the component editor gui
getfluxerror	Get the error in the flux of the specified component
getfluxpol	Get the polarization representation for the flux of the specified component
getfluxunit	Get the flux unit of the specified component

getfluxvalue	Get the flux value of the specified component
getfreq	Get the reference frequency
getfreqframe	Get the reference frequency frame
getfrequnit	Get the reference frequency unit
getfreqvalue	Get the reference frequency value
getlabel	Get the label of the specified component
getrefdir	Return the reference direction
getrefdirdec	Get the declination of the reference direction.
getrefdirframe	Get the reference frame of the reference direction.
getrefdirra	Get the RA of the reference direction.
getshape	Return the shape parameters the component
getspectrum	Return the spectral parameters the component
indices	Return a vector of indices.
is_physical	Check if a component is physically plausible
length	Find the number of components in the list.
print	Summarize the specified component to the logger
purge	Permanently delete removed components.
recover	Obtain removed components.
remove	Remove a component from the list.
rename	Give the list a name so it can save itself.
replace	Replace components in the list.
sample	Sample the flux of the list in a specified direction.
select	Mark components in the list
selected	Determine which components are selected
setflux	Set the flux of the specified components
setfreq	Set the reference frequency
setfreqframe	Set the reference frame for the frequency
setlabel	Set the label of the specified components
setrefdir	Set the reference direction
setrefdirframe	Set the reference frame for the direction
setshape	Change the shape of the component
setspectrum	Change the spectrum of the component
shapetype	Returns the shape type of the component
simulate	Add some simulated components to the list
sort	Sort the components in a list
spectrumtype	Returns the spectral shape of the component

emptycomponentlist - Constructor

Package	synthesis
Module	componentmodels
Tool	componentlist

Construct an empty componentlist

Synopsis
emptycomponentlist(log)

Description

Use this constructor to construct a componentlist tool that does not contain any components. Components can be appended to the list using the simulate function and the list can be stored in an AIPS++ table by giving it a name ie., using the rename function.

Arguments

log	Send a message to the logger Allowed Boolean Default T	

Returns
A componentlist tool

Example

```
include 'componentlist.g'
cl := emptycomponentlist();
```

---

componentlist - Constructor

Package	synthesis
Module	componentmodels
Tool	componentlist

Construct an componentlist from the data in an AIPS++ table

Synopsis
componentlist(filename, readonly, log)

Description

Use this constructor to construct a componentlist tool by reading the data from an AIPS++ 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[3].
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

filename	The filename of the table Allowed Any valid componentlist table Default no default
readonly	Should the table be opened read only Allowed T or F Default F
log	Send a message to the logger Allowed Boolean Default T

Returns
A componentlist tool

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
```

---

asciitocomponentlist - Constructor

Package	synthesis
Module	componentmodels
Tool	componentlist

Create a componentlist from an ascii file

[3]To be more precise the table should have been created using the functions in the ComponentList class (C++).

Synopsis

asciitocomponentlist(filename, asciifile, refer, format, direction, spectrum, flux, log)

Description

This constructor allows conversion of a number of ascii-file-based formats to componentlists. The formats currently supported are the AIPS star file format, the Caltech model format, and the WENSS, FIRST, and NVSS surveys.

AIPS ST file The AIPS star file describes positions of ''stars'' that one might wish to plot overlaid on some other display. It does not contain source strength information. In the AIPS help file for STARS, the format of the star file is described as follows:

```
    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             10: Five pointed star
   1: Ellipse           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: 11    pointed star
  7: Septagon         17: 12    pointed star
  8: Octagon          18: 13    pointed star
  9: Nine-gon         19: 14    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, AIPS++ 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 AIPS++, 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 declination. A detailed description appears in the 1998 May issue of The Astronomical Journal (Condon, J. J., Cotton, W. D., Greisen, E. W., Yin, Q. F., Perley, R. A., Taylor, G. B., & Broderick, J. J. 1998, AJ, 115, 1693). Go to the NVSS catalog search page, search for the sources that you want, cut out only the lines that include the sources (you can leave in the alternating lines with error information), put in a file and then convert.

Caveats:

- The catalog contains upper limits on some scale sizes. We have chosen to represent these as actual sizes.

Arguments

filename	Name of output component list table Allowed String Default None
asciifile	Name of input ascii file Allowed String Default None
refer	Input reference frame Allowed Any valid direction reference frame Default J2000
format	Name of format (only ST supported) Allowed String Default ST
direction	Direction measure (for relative coordinates) Allowed Direction measure Default None
spectrum	Default spectrum field Allowed Any valid spectrum field Default [type="Constant", frequency=[type="frequency" , refer="LSR" , m0=[unit="GHz" , value=1.0]]
flux	Default flux field Allowed Any valid flux field Default [value=[0.0, 0.0, 0.0, 0.0], unit='Jy', polarization="Stokes"]
log	Send a message to the logger Allowed Boolean Default T

Returns
Number of components found

Example

```
mycl := asciitocomponentlist('sgra-stars.cl', 'sgra-stars.stfile', refer='j2000',
format='ST')
```

This takes an AIPS stars file in J2000 positions, and converts the positions into Galactic. The output is a component list on disk and available in Glish as mycl.

---

componentlist.concatenate - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Append components from another componentlist.

Synopsis
concatenate(list, which, log)

Description

The concatenate function copies the specified component(s), from the specified to 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

list	list to copy from Alloweda componentlist tool Defaultno default
which	which components to copy Alloweda integer or integer vector Defaultunset
log	Send a message to the logger AllowedBoolean DefaultT

Returns
nothing

Example

```
include 'componentlist.g'
cl1 := componentlist('crux.cl');
```

```
cl2 := emptycomponentlist();
cl2.concatenate(cl1, [1,3]);
cl1.done()
cl2.rename('crux-copy.cl');
cl2.done()
```

This example reads a componentlist from an AIPS++ 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.remove - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Remove a component from the list.

Synopsis
remove(which, log)

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

which	indices of which component(s) to remove Allowed: a vector containing unique integers between 1 and the length of the list Default: no default	
log	Send a message to the logger Allowed: Boolean Default: T	

Returns
nothing

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
cl.remove(1);
```

This example reads a componentlist from an AIPS++ table and removes the first element from the list.

---

componentlist.purge - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Permanently delete removed components.

Synopsis
purge()

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.

Returns
nothing

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
cl.remove(1);
cl.purge();
```

This example reads a componentlist from a table and removes the first element from the list and then frees up the memory associated with the removed component.

---

componentlist.recover - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Obtain removed components.

Synopsis
recover(log)

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

log	Send a message to the logger AllowedBoolean DefaultT

Returns
nothing

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
cl.remove(1);
cl.recover();
```

This example reads a componentlist from a table and removes the first element from the list. The recover function is then called to append the component to the end of the list. Overall the list is unchanged although the first component has been moved to the end.

---

componentlist.length - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Find the number of components in the list.

Synopsis
length()

Description
The length function returns a non-negative integer that indicates how many components the list currently contains.

Returns
Non-negative integer

Example

```
include 'componentlist.g'
cl := emptycomponentlist();
n := cl.length();
```

This example creates an empty componentlist. Hence the variable n will be zero after the length function is executed.

---

componentlist.indices - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Return a vector of indices.

Synopsis
indices()

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.

Returns
vector of non-negative integers

Example

```
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);
}
```

This example reads a componentlist from disk and then proceeds to sum the flux of all the components. The units for all components are initially converted to Jansky's and the polarization representations set to Stokes.

---

componentlist.sort - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Sort the components in a list

Synopsis
sort(criteria, log)

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 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 (ra,dec) $= (0, 0)$, are at the beginning of the list.

Polarization Sorts the list so that components with the largest fractional polarization, $\frac{\sqrt{Q^2+U^2+V^2}}{I}$, are at the front of the list. Components where $I = 0$ are placed at the end of the list.

The parsing of the string containg the sorting criteria is case insensitive. You cannot sort a list that has been opened read only.

Arguments

criteria	a string containg the criteria to use to sort the list Allowed:'Flux', 'Position' or 'Polarization' Default:Flux
log	Send a message to the logger Allowed:Boolean Default:T

Returns
nothing

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
cl.sort('Polarization');
```

This example reads a componentlist from disk and then proceeds to sort the components so that the most polarized components are at the front of the list.

---

componentlist.is_physical - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Check if a component is physically plausible

Synopsis
is_physical(which)

Description
The is_physical 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:

- $I >= \sqrt{Q^2 + U^2 + V^2}$
- 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 indicies of all the components. The returned value will only be True if all the specified components are physical.

Arguments

which	A vector of indices	
	Allowed:	Indices must be between 1 and the list length
	Default:	no default

Returns
nothing

Example

```
include 'componentlist.g'
cl := emptycomponentlist();
cl.simulate(2);
cl.setflux(1, [10, 1+3i, 1-4i, 0]print cl.is_physical(1:2), polarization='linear');
print cl.is_physical(1:2)
```

This example creates a non-physical component hence the print statement will return F.

---

componentlist.sample - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Sample the flux of the list in a specified direction.

Synopsis
sample(direction, pixellatsize, pixellongsize, frequency)

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

direction	The direction to sample Allowedany valid direction measure. Defaultno default
pixellatsize	the x-size of the in pixels to use when sampling Allowedany quantity that has angular units. Defaultno default
pixellongsize	the y-size of the in pixels to use when sampling Allowedany quantity that has angular units. Defaultno default
frequency	The frequency to sample at AllowedAny frequency measure Defaultno default

Returns
A vector with four elements containing the flux in Janskys/pixel in the I, Q, U, & V polarizations

Example

```
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);
```

In this example the componentlist is sampled at the position of α-Cru. The flux will be a vector with four elements.

---

componentlist.rename - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Give the list a name so it can save itself.

Synopsis

rename(filename, log)

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 AIPS++ 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.
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.
If a table with the specified name already exists then the user is prompted, using a popup dialog window, to see whether it should be overwritten. In a non-windows environment the file is not overwritten.

Arguments

filename	The filename of the table Allowed: Any valid filename Default: no default
log	Send a message to the logger Allowed: Boolean Default: T

Returns
nothing

Example

```
include 'componentlist.g'
cl := emptycomponentlist();
cl.simulate(1);
cl.setshape(1, 'gaussian', '35mas', '27mas', '-10d')
cl.setflux(1, [1.0, 0.2, 0.1, 0.01]);
cl.rename('smallblob.cl');
cl.close();
```

```
n := cl.length();
```

This example creates and 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 AIPS++ 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 n contains zero at the end of this example.

---

componentlist.simulate - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Add some simulated components to the list

Synopsis
simulate(howmany, log)

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 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

howmany	How many components to simulate Allowed A positive integer Default 1
log	Send a message to the logger Allowed Boolean Default T

Returns
nothing

Example

```
include 'componentlist.g'
cl := emptycomponentlist();
cl.simulate(2)
cl.edit(1);
cl.setflux(2, [2.78, 0, 0, 0]);
cl.rename('test.cl');
cl.close();
```

This example creates a componentlist with two components. The component editor gui is then called on the first component to set the parameters to the desired ones, and the setflux function is used to modify the second one. 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 - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Add a component to the list

Synopsis
addcomponent(flux, fluxunit, polarization, ra, raunit, dec, decunit, dirframe, shape, majoraxis, minoraxis, positionangle, freq, freqframe, spectrumtype, index, label)

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.

spectrumtype, index See the setspectrum function.

label See the setlabel function.

Arguments

flux	The flux value Allowed: A vector with four real or complex numbers Default: [1,0,0,0]
fluxunit	The units of the flux Allowed: Any string with the same dimensions as the Jansky Default: Jy
polarization	The polarization of the value field Allowed: ‘‘Stokes’’, ‘‘linear’’ or ‘‘circular’’ Default: Stokes
ra	The RA of the new direction Allowed: A formatted string or a number Default: 00:00:00.00
raunit	The units of the RA Allowed: Any string containing an angular unit or ’angle’ or ’time’ Default: ’time’
dec	The declination of the new direction Allowed: A formatted string or a number Default: ’90.00.00.00’
decunit	The units of the declination Allowed: Any string containing an angular unit or ’angle’ or ’time’ Default: ’time’
dirframe	The new reference frame Allowed: A string like ’B1950’, ’J2000’ or ’galactic’ Default: j2000
shape	The new shape type Allowed: A string that is either ’point’, ’Gaussian’ or ’disk’ Default: point
majoraxis	The width of the larger axis Allowed: A quantity with angular units Default: 2arcmin
minoraxis	The width of the smaller axis Allowed: A quantity with angular units Default: 1arcmin
positionangle	The rotation of the axes with respect to the reference frame Allowed: A quantity with angular units Default: 0deg
freq	The reference frequency Allowed: A quantity with units equivalent to the ’Hz’ Default: 1.415GHz
freqframe	The new reference frame Allowed: A string like ’LSR’, ’GEO’ or ’TOPO Default: LSR
spectrumtype	The new spectrum type Allowed: A string that is either ’constant’ or ’spectral index’ Default: constant
index	The spectral indices Allowed: A 4-element vector or real numbers Default: [1,0,0,0]
label	The label for the component Allowed: A string Default: The default label

Returns
nothing

---

componentlist.close - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Save the componentlist to disk and reset its state.

Synopsis
close(log)

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 is the same state as a componentlist that has been newly created using the emptycomponentlist constructor. 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

log	Send a message to the logger AllowedBoolean DefaultT

Returns
nothing

Example
See the example for the rename function.

---

componentlist.edit - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Start up the component editor gui

Synopsis
edit(which, log)

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 editting a component. It is also suggested you only edit one component at a time.

Arguments

which	An index specifying which component Allowed An integer between 1 and the list length Default no default
log	Send a message to the logger Allowed Boolean Default T

Returns
nothing

Example
See the example for the simulate function.

---

componentlist.done - Function

Package synthesis
Module componentmodels
Tool componentlist

Delete the componentlist tool

Synopsis
done()

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.

Returns
T

Example
See the example for the rename function.

---

componentlist.select - Function

Package synthesis
Module componentmodels
Tool componentlist

Mark components in the list

Synopsis
select(which)

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 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

which	A vector of indices	
	Allowed:	Indices must be between 1 and the list length
	Default:	no default

Returns
nothing

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
cl.select([1,3]);
```

This example reads a list from disk and then marks the first and third component as selected.

---

componentlist.deselect - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Unmark components in the list

Synopsis
deselect(which)

Description

The deselect function is used to remove the ‘‘selected’’ mark from specified components in the list. This function wiil 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

which	A vector of indices	
	Allowed:	Indices must be between 1 and the list length
	Default:	no default

Returns
nothing

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
cl.select([1,3]);
cl.deselect([2,3]);
```

This example reads a list from disk. After the deselect operation only the first component is still selected.

---

componentlist.selected - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Determine which components are selected

Synopsis

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.

Returns
A vector of integers between 1 and the list length

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
cl.select([1,3]);
cl.deselect([2,3]);
cl.selected()
```

This example reads a list from disk. After the deselect operation only the first component is still selected and the selected function will return [1]

---

componentlist.getlabel - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Get the label of the specified component

Synopsis
getlabel(which)

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

which	An index specifying which component	
	Allowed	An integer between 1 and the list length
	Default	no default

Returns
A string

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
print cl.getlabel(1);
```

This example prints the label associated with the first component. It returns 'Alpha Crux'.

---

componentlist.setlabel - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Set the label of the specified components

Synopsis
setlabel(which, value, log)

Description
The setlabel function is used to reassign the label (an arbitrary text string) of the specified components to a new value.

Arguments

which	A vector of indices specifying the components to modify Allowed A vector with indices between 1 and the list length Default no default	
value	The label for the specified components Allowed A string Default no default	
log	Send a message to the logger Allowed Boolean Default T	

Returns
nothing

Example

```
include 'componentlist.g'
cl := componentlist('centarusA.cl');
cl.setlabel(1, 'Core')
cl.setlabel(2:10, 'Left Lobe')
cl.setlabel(11:23, 'Right Lobe')
```

This example reads a model of Centarus A which contains at least 23 components. It then marks the first one as the core, the next nine as the left lobe, and the next thirteen as the right lobe. The lobes may, for example, be modelled using a collection of Gaussian shaped components.

---

componentlist.getfluxvalue - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Get the flux value of the specified component

Synopsis
getfluxvalue(which)

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

which	An index specifying which component	
	Allowed	An integer between 1 and the list length
	Default	no default

Returns
A vector of length 4 with the flux values in the current units and polarization representation.

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
flux := cl.getfluxvalue(1);
unit := cl.getfluxunit(1);
pol := cl.getfluxpol(1);
err := cl.getfluxerror(1);
```

This example returns the values, units, polarization and error of the first component in the list.

---

componentlist.getfluxunit - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Get the flux unit of the specified component

Synopsis
getfluxunit(which)

Description
The getfluxunit function returns the units of the flux of the specified component. The actual values are obtained using the getfluxvalue function.

Arguments

which		An index specifying which component
	Allowed	An integer between 1 and the list length
	Default	no default

Returns
A string with the units of the flux for the specified component

Example
See the example for the getfluxvalue function.

---

componentlist.getfluxpol - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Get the polarization representation for the flux of the specified component

Synopsis
getfluxpol(which)

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

which	An index specifying which component	
	Allowed	An integer between 1 and the list length
	Default	no default

Returns
A string containing either ''Stokes'', ''linear'' or ''circular''. If ''Stokes'' is returned, then the flux values are the [I,Q,U,V] components of the polarization and are always real. For ''linear and ''circular'' the values returned are complex. If the value is ''linear'' then the values returned are the [XX,XY,YX,YY] components of the polarization. The parrallactic angle is assumed to be zero in this case. If the value is ''circular'' then the values returned are the [RR,RL,LR,LL] components of the polarization.
See the example for the getfluxvalue function.

---

componentlist.getfluxerror - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Get the error in the flux of the specified component

Synopsis
getfluxerror(which)

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 retreived and if any of the parameters of the flux change the user is responsible for updating the errors.

Arguments

which	An index specifying which component	
	Allowed	An integer between 1 and the list length
	Default	no default

Returns
A vector of length 4 with the errors in the flux values in the current units and polarization representation.
See the example for the getfluxvalue function.

---

componentlist.setflux - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Set the flux of the specified components

Synopsis
setflux(which, value, unit, polarization, error, log)

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.

Arguments

which	A vector of indices specifying the components to modify Allowed: A vector with indices between 1 and the list length Default: no default
value	The flux values for the specified components Allowed: A vector with four real or complex numbers Default: no default
unit	The units of the flux Allowed: Any string with the same dimensions as the Jansky Default: Jy
polarization	The polarization of the value field Allowed: ''Stokes'', ''linear'' or ''circular'' Default: Stokes
error	The error in the value field Allowed: A complex vector of length four. Default: [0,0,0,0]
log	Send a message to the logger Allowed: Boolean Default: T

Returns
nothing

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
cl.setflux(cl.indices(), [1,0,0,0], unit='jy',
   polarization='stokes', error=[.3, 0, 0, 0])
```

This example sets all the components in the list to a flux of 1 Jy in I and zero in the Q, U, & V polarizations.

---

componentlist.convertfluxunit - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Change (convert) the flux units of the specified components

Synopsis
convertfluxunit(which, unit)

Description
The convertfluxunit function is used to convert the flux to another unit. The units *must* have the same dimensions as the Jansky.

Arguments

which	A vector of indices specifying the components to modify Allowed A vector with indices between 1 and the list length Default no default
unit	The units of the flux Allowed Any string with the same dimensions as the Jansky Default Jy

Returns
nothing

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
print cl.getfluxvalue(1);
cl.convertflux(cl.indices(), 'WU');
print cl.getfluxvalue(1);
```

This example converts the flux of all components to ''Westerbork Units''. Assuming the units were originally Jansky's then the numbers printed before and after the call to the convertfluxunit function will differ by a factor of 200.

componentlist.convertfluxpol - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Change (convert) the polarization representation of the specified components

Synopsis
convertfluxpol(which, polarization)

Description
The convertfluxpol function is used to convert the flux to another polarization representation. There are are three representations used, namely , 'Stokes', 'linear' & 'circular'

Arguments

which	A vector of indices specifying the components to modify Allowed: A vector with indices between 1 and the list length Default: no default
polarization	The new polarization representation Allowed: ''Stokes'', ''linear'' or ''circular'' Default: Stokes

Returns
nothing

Example

```
include 'componentlist.g'
cl := componentlist('centarusA.cl');
print cl.getfluxvalue(1);
cl.convertpol(1, 'linear');
print cl.getfluxvalue(1);
```

This example converts the flux of the first component to a linear representation. If the flux was originally [1,0.1,0,0.01] (in the Stokes representation) then the resultant flux will be [1.1, 0.01i, -0.01i, 0.9]. Note that the flux is returned as a complex vector when the polarization representation is not Stokes.

---

componentlist.getrefdir - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Return the reference direction

Synopsis
getrefdir(which)

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

which		An index specifying which component
	Allowed	An integer between 1 and the list length
	Default	no default

Returns
A direction measure

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
dir := cl.getrefdir(1);
```

This example returns the direction of the centre of the first component in the componentlist. The returned value is a direction measure that can be used as an argument to many functions within AIPS++.

---

componentlist.getrefdirra - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Get the RA of the reference direction.

Synopsis
getrefdirra(which, unit, precision)

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, degrees,minutes or degrees, minutes & 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, hours, minutes or hours, minutes & seconds are returned.
All directions are stored internally in double precision.

Arguments

which	An index specifying which component Allowed An integer between 1 and the list length Default no default	
unit	The angular unit of the returned value. Allowed Any string containing an angular unit or 'angle' or 'time' Default deg	
precision	The number of digits in the returned string. Allowed Numbers between 1 and 16 make the most sense Default 6	

Returns
A formatted string containing the RA of the reference direction.

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
print 'The first component is at RA: ', cl.getrefdirra(1, 'time'),
      ' Dec: ', cl.getrefdirdec(1, 'angle'),
      ' (', cl.getrefdirframe(1), ')'
```

This example prints the following:

```
The first component is at RA:  12:26:36  Dec:  -063.05.55  ( J2000 )
```

which is the direction of the first component.

---

componentlist.getrefdirdec - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Get the declination of the reference direction.

Synopsis
getrefdirdec(which, unit, precision)

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

which	An index specifying which component Allowed An integer between 1 and the list length Default no default
unit	The angular unit of the returned value. Allowed Any string containing an angular unit or 'angle' or 'time' Default deg
precision	The number of digits in the returned string. Allowed Numbers between 1 and 16 make the most sense Default 6

Returns
A formatted string containing the declination of the reference direction.

Example
See the example for the getrefdirra function.

---

componentlist.getrefdirframe - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Get the reference frame of the reference direction.

Synopsis
getrefdirframe(which)

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

which	An index specifying which component	
	Allowed	An integer between 1 and the list length
	Default	no default

Returns
A formatted string containing the reference frame of the reference direction.

Example
See the example for the getrefdirra function.

---

componentlist.setrefdir - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Set the reference direction

Synopsis
setrefdir(which, ra, raunit, dec, decunit, log)

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 (if raunit argument is not 'time' or 'angle').
The raunit argument is a string that indicates the units for the ra. It can be any angular unit eg., 'deg', 'rad', 'arcsec', 'mas', or it can be 'time' or 'angle'. If the raunit is 'time'

then the ra argument must be a string in the format
'HH:MM:SS.sss' eg., '19:34:63.8'. If the raunit is 'angle' then the ra argument must be a string in the format '+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

which	A vector of indices specifying the components to modify Allowed A vector with indices between 1 and the list length Default no default
ra	The RA of the new direction Allowed A formatted string or a number Default no default
raunit	The units of the RA Allowed Any string containing an angular unit or 'angle' or 'time' Default no default
dec	The declination of the new direction Allowed A formatted string or a number Default no default
decunit	The units of the declination Allowed Any string containing an angular unit or 'angle' or 'time' Default no default
log	Send a message to the logger Allowed Boolean Default T

Returns
nothing

Example

```
include 'componentlist.g'
cl := emptycomponentlist('crux.cl');
cl.simulate(3);
cl.setrefdir(1, '12:26:35.9', 'time', '-63.5.56', 'angle')
```

```
cl.setrefdir(2, '12h26m35.9', 'time', '-63d5m56', 'angle')
cl.setrefdir(3, -173.35, 'deg', '-1.10128', 'rad')
```

This example sets the reference direction for three newly created components to the same value.

---

componentlist.setrefdirframe - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Set the reference frame for the direction

Synopsis
setrefdirframe(which, frame, log)

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 are 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

which	A vector of indices specifying the components to modify Allowed A vector with indices between 1 and the list length Default no default
frame	The new reference frame Allowed A string like 'B1950', 'J2000' or 'galactic' Default no default
log	Send a message to the logger Allowed Boolean Default T

Returns
nothing

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
cl.setrefdirframe(cl.indices(), 'B1950');
```

This example sets the reference frame for all the components in the list to the B1950 reference frame. You would use this example if you had entered the B1950 coordinates for the direction of all the components in the list but had forgotten that the default reference frame was J2000 for newly created components.

---

componentlist.convertrefdir - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Convert the reference direction to a new frame

Synopsis
convertrefdir(which, 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

which	A vector of indices specifying the components to modify AllowedA vector with indices between 1 and the list length Defaultno default
frame	The new reference frame AllowedA string like 'B1950', 'J2000' or 'galactic' Defaultno default

Returns
nothing

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
cl.convertrefdirframe(cl.indices(), 'J2000');
```

This example illustrates what you would do if you had a list where all the directions were in the B1950 reference frame but you needed the coordinates in the J2000 reference frame. Unlike the setrefdirframe function this one adjusts the right ascension and declination values.

---

componentlist.shapetype - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Returns the shape type of the component

Synopsis
shapetype(which)

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', and 'Disk'. This function returns one of these three strings.

Arguments

which	An index specifying which component	
	Allowed	An integer between 1 and the list length
	Default	no default

Returns
A string specifying the current component shape

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
print 'The first component has a', cl.shapetype(1), ' shape';
```

This example prints the following:

```
The first component has a Point  shape
```

---

componentlist.getshape - Function

Package synthesis
Module componentmodels
Tool componentlist

Return the shape parameters the component

Synopsis
getshape(which)

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

which	An index specifying which component	
	Allowed:	An integer between 1 and the list length
	Default:	no default

Returns
A record with fields that contain the parameters of the component shape.

Example
See the examples for the setshape & convertshape functions.

---

componentlist.setshape - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Change the shape of the component

Synopsis
setshape(which, type, majoraxis, minoraxis, positionangle, log)

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' or 'disk'. The parsing of this string is case insensitive.

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' or 'disk', 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

which	A vector of indices specifying the components to modify Allowed A vector with indices between 1 and the list length Default no default
type	The new shape type Allowed A string that is either 'point', 'Gaussian' or 'disk' Default Point
majoraxis	The width of the larger axis Allowed A quantity with angular units Default 1arcmin
minoraxis	The width of the smaller axis Allowed A quantity with angular units Default 1arcmin
positionangle	The rotation of the axes with respect to the reference frame Allowed A quantity with angular units Default 0deg
log	Send a message to the logger Allowed Boolean Default T

Returns
nothing

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl', readonly=F);
cl.setshape(3, 'disk',  '45mas', '45mas');
print cl.getshape(3).majoraxis;
```

This example revises the model for Gamma-Crux changing its shape from a point to a circular disk of diameter 45mas. The output from the print statement is [value=45, unit=mas].

---

componentlist.convertshape - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Change the units of the shape parameters

Synopsis
convertshape(which, majoraxis, minoraxis, positionangle)

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

which	A vector of indices specifying the components to modify Allowed A vector with indices between 1 and the list length Default no default
majoraxis	The units to use on the larger axis Allowed A string with angular units Default arcmin
minoraxis	The units to use on the smaller axis Allowed A string with angular units Default arcmin
positionangle	The units to use for the rotation of these axes Allowed A string with angular units Default deg

Returns
nothing

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
cl.convertshape(3, 'arcsec', 'arcsec');
print cl.getshape(3).minoraxis;
```

This example converts the units for the majoraxis and minor axis widths to arc-seconds. Assuming they where originally set to 45mas, as is done in the example for the setshape function, the value printed would be [value=0.045, unit=arcsec].

---

componentlist.spectrumtype - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Returns the spectral shape of the component

Synopsis
spectrumtype(which)

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

which		An index specifying which component
	Allowed	An integer between 1 and the list length
	Default	no default

Returns
A string specifying the spectral shape of the specified component

Example

```
include 'componentlist.g'
cl := componentlist('crux.cl');
print 'The first component has a', cl.spectrumtype(1), ' spectrum';
```

This example prints the following:

```
The first component has a Constant  spectrum
```

---

componentlist.getspectrum - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Return the spectral parameters the component

Synopsis
getspectrum(which)

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 indicies for the I, Q, U, & V components of the flux.

Arguments

which	An index specifying which component	
	Allowed	An integer between 1 and the list length
	Default	no default

Returns
A record with fields that contain the parameters of the spectral type.

Example
See the examples for the setspectrum & getspectrum functions.

---

componentlist.setspectrum - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Change the spectrum of the component

Synopsis
setspectrum(which, type, index, log)

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. The index argument is a 4-element vector that contains the spectral index of the I, Q, U, & V polarizations. 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

which	A vector of indices specifying the components to modify Allowed: A vector with indices between 1 and the list length Default: no default
type	The new spectrum type Allowed: A string that is either 'constant or 'spectral index' Default: Constant
index	The spectral indices Allowed: A 4-element vector or real numbers Default: [1,0,0,0]
log	Send a message to the logger Allowed: Boolean Default: T

Returns
nothing

Example

```
include 'componentlist.g'
cl := componentlist('centarusA.cl');
cl.setspectrum(2:10, 'spectral index', [-0.5, 0, 0, 0]);
print cl.getshape(2).index;
```

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]

---

componentlist.getfreq - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Get the reference frequency

Synopsis
getfreq(which)

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

which		An index specifying which component
	Allowed	An integer between 1 and the list length
	Default	no default

Returns
A frequency measure (a record)

Example

```
include 'componentlist.g'
cl := componentlist('centarusA.cl');
f := cl.getfreq(2);
```

This example returns the reference frequency of the first component in the componentlist. The returned value is a frequency measure that can be used as an argument to many functions within AIPS++.

---

componentlist.getfreqvalue - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Get the reference frequency value

Synopsis
getfreqvalue(which)

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 getfrequnit function

Arguments

which	An index specifying which component	
	Allowed	An integer between 1 and the list length
	Default	no default

Returns
A number containing the value of the reference frequency.

Example

```
include 'componentlist.g'
cl := componentlist('centarusA.cl');
print 'The reference frequency is ', cl.getfreqvalue(1), ' ',
      cl.getfrequnit(1), '(', cl.getfreqframe(1), ')'
```

This example prints the following:

```
The reference frequency is  1   GHz ( LSR )
```

which is the reference frequency of the first component.

---

componentlist.getfrequnit - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Get the reference frequency unit

Synopsis
getfrequnit(which)

Description

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

which	An index specifying which component	
	Allowed	An integer between 1 and the list length
	Default	no default

Returns
A string containing the unit of the reference frequency.

Example
See the example for the getfreqvalue function.

---

componentlist.getfreqframe - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Get the reference frequency frame

Synopsis
getfreqframe(which)

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

which		An index specifying which component
	Allowed	An integer between 1 and the list length
	Default	no default

Returns
A string containing the reference frame of the reference frequency.

Example
See the example for the getfreqvalue function.

---

componentlist.setfreq - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Set the reference frequency

Synopsis
setfreq(which, value, unit, log)

Description

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

which	A vector of indices specifying the components to modify Allowed: A vector with indices between 1 and the list length Default: no default	
value	The new frequency value Allowed: A number Default: no default	
unit	The units of the frequency Allowed: Any string with the same dimensions as the 'Hz' Default: 'GHz'	
log	Send a message to the logger Allowed: Boolean Default: T	

Returns
nothing

Example

```
include 'componentlist.g'
cl := componentlist('centarusA.cl');
cl.setfreq(1, 1.415, 'GHz');
```

This example sets the reference frequency of the first component to 1.415 GHz. If the spectral type is 'spectral index' and the indicies are non-zero then this will change the flux of the component at all frequencies.

---

componentlist.setfreqframe - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Set the reference frame for the frequency

Synopsis
setfreqframe(which, frame, log)

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

which	A vector of indices specifying the components to modify Allowed A vector with indices between 1 and the list length Default no default
frame	The new reference frame Allowed A string like 'LSR', 'GEO' or 'TOPO Default no default
log	Send a message to the logger Allowed Boolean Default T

Returns
nothing

Example

```
include 'componentlist.g'
cl := componentlist('centarusA.cl');
cl.setfreqframe(1:23, 'LSR')
```

This example sets the reference frequency frame for the first component to LSR (local standard of rest).

---

componentlist.convertfrequnit - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Convert the reference frequency to a new unit

Synopsis
convertfrequnit(which, 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

which	A vector of indices specifying the components to modify Allowed: A vector with indices between 1 and the list length Default: no default
unit	The new frequency unit Allowed: Any string with the same dimensions as the 'Hz' Default: 'GHz'

Returns
nothing

Example

```
include 'componentlist.g'
cl := componentlist('centarusA.cl');
cl.convertfrequnit(1, 'kHz');
print 'The reference frequency is ', cl.getfreqvalue(1), ' ',
      cl.getfrequnit(1), '(', cl.getfreqframe(1), ')'
```

This example prints, when used in conjunction with example from the setfreq function the following:

```
The reference frequency is  1.415e+06   kHz ( LSR )
```

---

componentlist.component - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Extract a component from the list.

Synopsis
component(which, iknow)

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

which	index of which component to extract Allowed integers between 1 and the length of the list Default no default
iknow	Suppress the warning message Allowed Boolean Default F

Returns
A record representing a component.

---

componentlist.add - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Add a component to the list.

Synopsis
add(component, iknow)

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. This function will be removed in a future release of aips++.

Arguments

component	A record that represents a component Allowed any record that contains the required fields Default no default
iknow	Suppress the warning message Allowed Boolean Default F

Returns
nothing

---

componentlist.replace - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Replace components in the list.

Synopsis
replace(which, list, whichones)

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

which	A vector of indices specifying the components to replace Allowed A vector with indices between 1 and the list length Default no default
list	The list containing the components to copy Allowed A componentlist tool Default no default
whichones	A vector of indices specifying the components to copy Allowed A vector with indices between 1 and the length of the list in the second argument Default F

Returns
nothing

Example

```
include 'componentlist.g'
newcl := componentlist('core', readonly=F);
othercl := componentlist('centarusA.cl', readonly=T);
newcl.replace(1:2, othercl, [10,13]);
```

This example replaces components 1 and 2 in the newcl list with components 10 & 13 from the othercl list.

componentlist.print - Function

Package	synthesis
Module	componentmodels
Tool	componentlist

Summarize the specified component to the logger

Synopsis
print(which)

Description
The print function summarizes the contents of the specified components to the logger.

Arguments

which		An index specifying which component
	Allowed	Unset or an integer between 1 and the list length
	Default	All components

Returns
T

1.3.2 is_componentlist - Function

Package	synthesis
Module	componentmodels

Is the argument a componentlist tool?

Synopsis
is_componentlist(tool)

Description
This global function can be used to determine if the supplied