Determine if a given *Glish* variable is a **region** `tool`.

**Example**

```
- r1 := drm.box()
- is_region(r1)
T
- r2 := [10,20]
- is_region(r2)
F
```

---

### 1.2.10 coordsys - Tool

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |

Operations on CoordinateSystems
include "coordsys.g"

**Constructors**

| | |
|---|---|
| coordsys | Create a default coordsys tool |

**Functions**

| | |
|---|---|
| axesmap | Find mapping between world and pixel axes |
| axiscoordinatetypes | Return types of coordinates for each axis |
| conversiontype | Get extra reference conversion layer |
| convert | Convert a numeric mixed coordinate |
| convertmany | Convert many numeric mixed coordinates |
| coordinatetype | Return type of specified coordinate |
| copy | Copy this Coordsys tool |
| done | Destroy this Coordsys tool |
| epoch | Return the epoch |
| findaxis | Find specified axis in coordinate system |
| findcoordinate | Find axes of specified coordinate |
| frequencytofrequency | Convert frequency to frequency with an offset |
| frequencytovelocity | Convert frequency to velocity |
| fromrecord | Fill Coordinate System from a record |
| id | Return the fundamental identifier of this tool |
| increment | Recover the increments |

| | |
|---|---|
| lineartransform | Recover the linear transform matrix |
| names | Recover the names for each axis |
| naxes | Recover the number of axes |
| ncoordinates | Recover the number of coordinates in the Coordinate System |
| observer | Return the name of the observer |
| projection | Recover the direction coordinate projection |
| referencecode | Return specified reference code |
| referencepixel | Recover the reference pixel |
| referencevalue | Recover the reference value |
| reorder | Reorder the coordinates |
| replace | Replace a coordinate |
| restfrequency | Recover the rest frequency |
| setconversiontype | Set extra reference conversion layer |
| setdirection | Set direction coordinate values |
| setepoch | Set a new epoch |
| setincrement | Set the increment |
| setlineartransform | Set the linear transform |
| setnames | Set the axis names |
| setobserver | Set a new observer |
| setprojection | Set the direction coordinate projection |
| setreferencecode | Set new reference code |
| setreferencelocation | Set reference pixel and value |
| setreferencepixel | Set the reference pixel |
| setreferencevalue | Set the reference value |
| setrestfrequency | Set the rest frequency |
| setspectral | Set tabular values for the spectral coordinate |
| setstokes | Set the Stokes types |
| settabular | Set tabular values for the tabular coordinate |
| settelescope | Set a new telescope |
| setunits | Set the axis units |
| stokes | Recover the Stokes types |
| summary | Summarize basic information about the Coordinate System |
| telescope | Return the telescope |
| toabs | Convert relative coordinate to absolute |
| toabsmany | Convert many numeric relative coordinates to absolute |
| topixel | Convert from absolute world to pixel coordinate |
| topixelmany | Convert many absolute numeric world coordinates to pixel |
| torecord | Convert Coordinate System to a record |
| torel | Convert absolute coordinate to relative |
| torelmany | Convert many numeric absolute coordinates to relative |
| toworld | Convert from absolute pixel coordinate to world |
| toworldmany | Convert many absolute pixel coordinates to numeric world |
| type | Return the type of this tool |
| units | Recover the units for each axis |
| velocitytofrequency | Convert velocity to frequency |

**Summary**

A Coordsys `tool` is used to store and manipulate a Coordinate System (we will use the term 'Coordinate System' interchangeably with 'Coordsys `tool`'). A Coordinate System is a collection of coordinates, such as a direction coordinate (E.g. RA/DEC), or a spectral coordinate (e.g. an LSRK frequency).

The main job of the Coordsys `tool` is to convert between absolute pixel and world (physical) coordinates. It also supports relative pixel and world coordinates (relative to reference location).

A Coordinate System is generally associated with an image (manipulated via an Image `tool`) but can also exist in its own right. An image is basically just a regular lattice of pixels plus a Coordinate System describing the mapping from pixel coordinate to world (or physical) coordinate.

Each coordinate is associated with a number of axes. For example, a direction coordinate has two coupled axes; a longitude and a latitude. A spectral coordinate has one axis. A linear coordinate can have an arbitrary number of axes, but they are all uncoupled. The Coordinate System actually maintains two kinds of axes; pixel axes and world axes.

As well as the coordinates, there is some extra information stored in the Coordinate System. This consists of the telescope, the epoch (date of observation), and the highly influential observer's name. The telescope (i.e. position on earth) and epoch are important if you want to, say, change a spectral coordinate from LSRK to TOPO.

For general discussion about celestial coordinate systems, see the papers by Mark Calabretta and Eric Greisen. Background on the WCS system and relevant papers (including the papers published in

```
A&A 2002, 1061-1075 and 1077-1122
```

can be found here. Note that the actual system implemented originally in AIPS++ was based on a 1996 draft of these papers. The final papers are being implemented while new version of the defining library become available.

**Coordinate formatting**

Many of the Coordsys `tool` functions use a world coordinate value as an argument. This world value can be formatted in many ways.

Some functions (e.g. toworld) have a function argument called `format` which takes a string. This controls the format in which the coordinate is output and hence possibly input into some other function.

Possibilities for `format` are :

- 'n' - means the world coordinate is given as a numeric vector (actually doubles). The units are implicitly those returned by function units.
- 'q' - means the world coordinate is given as a vector of quantities (value and unit) - see the quanta module. If there is only one axis (e.g. spectral coordinate), you will get a single quantum only.

- 'm' - means the world coordinate is given as a record of measures - see the measures module.

  The record consists of fields named from from `direction`, `spectral`, `stokes`, `linear`, and `tabular`, depending upon which coordinate types are present in the Coordinate System.

  The `direction` field holds a direction measure.

  The `spectral` field holds further subfields `frequency`, `radiovelocity`, `opticalvelocity`, `betavelocity`. The `frequency` subfield holds an frequency measure. The `radiovelocity` subfield holds a doppler measure using the radio velocity definition. The `opticalvelocity` subfield holds a doppler measure using the optical velocity definition. The `betavelocity` subfield holds a doppler measure using the true or beta velocity definition.

  The `stokes` field just holds a string giving the Stokes type (not a real measure).

  The `linear` and `tabular` fields hold a vector of quanta (not a real measure).

- 's' - means the the world coordinate is given as a vector of formatted strings

You can give any combination of the allowed letters when using the `format` argument. If you supply one letter the coordinate is as described above. If you supply more than one letter, the coordinate is given as a record, with possible fields 'numeric', 'quantity', 'measure' and 'string' where each of these fields is given as described above.

There are functions torel and toabs used to inter-convert between absolute and relative world and pixel coordinates. These functions have an argument `isworld` whereby you can specify whether the coordinate is a pixel coordinate or a world coordinate. In general, you should not need to use this argument because any coordinate variable generated by Coordsys `tool` functions 'knows' whether it is absolute or relative, world or pixel (uses a *Glish* attribute). However, you may be inputting a coordinate variable which you have generated in some other way, and then you may need this.

**Stokes Coordinates**
Stokes axes don't fit very well into our Coordinate model since they are not interpolatable. The alternative to having a Stokes Coordinate is having a Stokes pixel type (like double, complex). Both have their good and bad points. We have chosen to use a Stokes Coordinate.
With the Stokes Coordinate, any absolute pixel coordinate value must be in the range of 1 to $nStokes$, where $nStokes$ is the number of Stokes types in the Coordinate.

We define relative world coordinates for a Stokes axis to be the same as absolute world coordinates (it makes no sense to think of a relative value $XY - XX$ say).
You can use the specialized functions stokes and setstokes to recover and set new Stokes values in the Stokes Coordinate.

**World and Pixel axes**
The Coordinate System maintains what it calls pixel and world axes. The pixel axis is associated with, say, the axes of a lattice of pixels. The world axes describe notional world axes, generally in the same order as the pixel axes. However, they may be different. Imagine that a 3-D image is collapsed along one axis. The resultant image has 2 pixel axes. However, we can maintain the world coordinate for the collapsed axis (so we know the coordinate value still). Thus we have three world axes and two pixel axes. It is also possible for the C++ programmer to reorder these pixel and world axes. However, this is strongly discouraged, and you should never actually encounter a situation where the pixel and world axes are in different orders, but you may encounter cases where the number of world and pixela axes is different.
For those of us (AIPS++ programmers) writing robust scripts, we must account for these possibilities, although the user really shouldn't bother. This, the `pixel` and `world` vectors return the pixel and world axes of the found coordinate.
The functions referencevalue, increment, units, and names return their vectors in world axis order. However, function referencepixel returns in pixel axis order (and the world vectors might have more values than the referencepixel vector).

**Overview of Coordsys tool functions**

- **Get/set -** Functions to get and set various items within the Coordinate System are
  - referencepixel - get the reference pixel
  - setreferencepixel - set the reference pixel
  - referencevalue - get the reference value
  - setreferencevalue - set the reference value
  - setreferencelocation - Set reference pixel and value to these values
  - increment - get axis increments
  - setincrement - set axis increments
  - lineartransform - get linear transform
  - setlineartransform - set linear transform
  - names - get axis names
  - setnames - set axis names
  - units - get axis units

- setunits - set axis units
- stokes - get Stokes values
- setdirection - set Direction coordinate values
- setstokes - set Stokes values
- setspectral - set Spectral coordinate tabular values
- settabular - set Tabular coordinate tabular values
- projection - get direction coordinate projection
- setprojection - set direction coordinate projection
- referencecode - get reference codes
- setreferencecode - set reference codes
- restfrequency - get the spectral coordinate rest frequency
- setrestfrequency - set the spectral coordinate rest frequency
- epoch - get the epoch of observation
- setepoch - set the epoch of observation
- telescope - get the telecope of the observation
- settelescope - set the telecope of the observation
- observer - get observer name
- setobserver - set observer name

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

- axesmap - get mapping between pixel and world axes order
- axiscoordinatetypes - get type of coordinate for each axis
- coordinatetype - get type of coordinates
- copy - make a copy of this tool
- done - destroy this `tool`
- findaxis - find specified axis (by number) in coordinate system
- findcoordinate - find specified (by number) coordinate
- fromrecord - set Coordinate System from a *Glish* record
- id - get the fundamental identifier of this `tool`
- naxes - get number of axes
- ncoordinates - get the number of coordinates
- reorder - reorder coordinates
- summary - summarize the Coordinate System
- torecord - Convert a Coordinate SYstem to a *Glish* record

- type - the type of this `tool`

- **Coordinate conversion**
  - convert - Convert one numeric coordinate with mixed input and output formats (abs/rel/world/pixel)
  - toabs - Convert a relative coordinate to an absolute coordinate
  - topixel - Convert from absolute world coordinate to absolute pixel coordinate
  - torel - Convert an absolute coordinate to a relative coordinate
  - toworld - Convert from an absolute pixel coordinate to an absolute world coordinate
  - convertmany - Convert many numeric coordinates with mixed input and output formats (abs/rel/world/pixel)
  - toabsmany - Convert many relative coordinates to absolute coordinates
  - topixelmany - Convert many absolute world coordinates to absolute pixel coordinates
  - torelmany - Convert many absolute coordinates to relative coordinates
  - toworldmany - Convert many absolute pixel coordinates to absolute world coordinates
  - frequencytovelocity - Convert from frequency to velocity
  - frequencytofrequency - Convert from frequency to frequency via a velocity offset
  - velocitytofrequency - Convert from velocity to frequency
  - setconversiontype - Set extra reference frame conversion layer
  - conversiontype - Recover extra reference frame conversion types

- **Tests -**
  - coordsystest - Run test suite for Coordsys `tool`

**Events**
There are no events emitted or acted upon by this `tool`.

**Example**
include 'coordsys.g'

```
cs := coordsys(direction=T, linear=2)
cs.summary()
```

**coordsys - Constructor**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Create a default coordsys tool

**Synopsis**
coordsys(direction, spectral, stokes, linear, tabular)

**Arguments**

| | | |
|---|---|---|
| direction | in | Make a direction coordinate ?<br>Allowed:Bool<br>Default: F |
| spectral | in | Make a spectral coordinate ?<br>Allowed:Bool<br>Default: F |
| stokes | in | Make a Stokes coordinate with these Stokes<br>Allowed:String or vector of Strings<br>Default: "" |
| linear | in | Make a linear coordinate with this many axes<br>Allowed:Integer<br>Default: 0 |
| tabular | in | Make a tabular coordinate<br>Allowed:Bool<br>Default: F |

**Returns**
Coordsys `tool` or fail

**Description**

By default, this constructor makes an empty Coordsys `tool`. You can ask it to include various sorts of coordinates through the arguments. Except for Stokes, you don't have any control over the coordinate contents (e.g. reference value etc.) it does make for you on request. But you can edit the Coordinate System after creation if you wish.
If you wish to make a Stokes coordinate, then you assign `stokes` to a string (or a vector of strings) saying which Stokes you want. AIPS++ allows rather a lot of potential Stokes types.

Probably most useful is some combination of the basic I, Q, U, V, XX, YY, XY, YX, RR, LL, RL, and LR.
However, a more esoteric choice is also possible: RX, RY, LX, LY, XR, XL, YR, YL (these are mixed linear and circular), PP, PQ, QP, QQ (general quasi-orthogonal correlation products) RCircular, LCircular, Linear (single dish polarization types).
You can also specify some polarization 'Stokes' types: Ptotal (Polarized intensity $((Q^2+U^2+V^2)^(1/2))$, Plinear (Linearly Polarized intensity $((Q^2+U^2)^(1/2))$, PFtotal (Polarization Fraction (Ptotal/I)), PFlinear (Linear Polarization Fraction (Plinear/I)), and Pangle (Linear Polarization Angle $(0.5arctan(U/Q)$ in radians)).
Probably you will find the more unusual types aren't fully supported throughout the system.
You can make a LinearCoordinate with as many uncoupled axes as you like. Thus, `linear=2` makes one LinearCoordinate with 2 axes (think of it like a DirectionCoordinate which also has 2 axes [but coupled in this case], a longitude and a latitude).
If you make a TabularCoordinate, it is linear to start with. You can change it to a non-linear one by providing a list of pixel and world values to function settabular.

### Example

```
- cs := coordsys()
- cs.ncoordinates()
0
- cs.done()
- cs
F
#
- cs2 := coordsys(direction=T, stokes='I V');
- cs2.ncoordinates()
2
- cs2.coordinatetype()
Direction Stokes
#
- cs2.summary()
Direction system : J2000
Telescope        : ATCA
Observer         : Karl Jansky
Date observation : 2000/07/03/08:10:57

Axis Type      Name             Proj   Coord value at pixel    Coord incr Units
-------------------------------------------------------------------------------
```

```
1    Direction Right Ascension   SIN  00:00:00.000     1.00 -6.000000e+01 arcsec
2    Direction Declination       SIN +00.00.00.000     1.00  6.000000e+01 arcsec
3    Stokes    Stokes                          IV
```

The second Coordinate System contains a direction coordinate and a Stokes coordinate. This means that there are three 'axes' associated with the 2 coordinates.

---

**coordsys.axesmap - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Find mapping between world and pixel axes

**Synopsis**
axesmap(toworld)

**Arguments**

| | | |
|---|---|---|
| toworld | in | Map from pixel to world ?<br>Allowed:T or F<br>Default: T |

**Returns**
Vector of Int or fail

**Description**

This function returns a vector describing the mapping from pixel to world or world to pixel axes. It is not for general user use.
See the discussion about pixel and world axis ordering. Generally they will be in the same order.

**Example**

```
- cs := coordsys(direction=T, spectral=T)
- cs.axesmap(T);
[1 2 3]
- cs.axesmap(F);
[1 2 3]
```

---

**coordsys.axiscoordinatetypes - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Return types of coordinates for each axis

**Synopsis**
axiscoordinatetypes(world)

**Arguments**

| | | |
|---|---|---|
| world | in | World or pixel axes ?<br>Allowed:T or F<br>Default: T |

**Returns**
Vector of strings or fail

**Description**

This function (short-hand name `act`) returns a vector string giving the coordinate type for each axis (world or pixel) in the Coordinate System. See the discussion about pixel and world axis ordering.

**Example**

```
- cs := coordsys(direction=T, spectral=T)
- cs.axiscoordinatetypes()
Direction Direction Spectral
```

---

**coordsys.conversiontype - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Get extra reference conversion layer

**Synopsis**
conversiontype(type)

**Arguments**

| | | |
|---|---|---|
| type | in | Coordinate type<br>Allowed:String from 'direction', 'spectral' |

**Returns**
String or fail

**Description**

Some coordinates contain a reference code. Examples of reference codes are B1950 and J2000 for direction coordinates, or LSRK and BARY for spectral coordinates. When you do conversions between pixel and world coordinate, the coordinates are in the reference frame corresponding to these codes. Function setconversiontype allows you to specify a different reference frame which is used when converting between world and pixel coordinate.
This function allows you to recover those conversion types. If no extra conversion layer has been set, you get back the native reference types.

**Example**

```
- cs := coordsys(direction=T, spectral=T)
- print cs.conversiontype (type='direction'), ' ', cs.conversiontype (type='spectral'),
J2000 LSRK
-
- cs.setconversiontype (direction='GALACTIC', spectral='BARY');
- print cs.conversiontype (type='direction'), ' ', cs.conversiontype (type='spectral'),
GALACTIC BARY
```

---

---

**coordsys.convert - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Convert a numeric mixed coordinate

**Synopsis**

convert(coordin, absin, dopplerin, unitsin, absout, dopplerout, unitsout, shape)

**Arguments**

| | | |
|---|---|---|
| coordin | in | Input coordinate<br>Allowed:Numeric vector |
| absin | in | Are input coordinate elements absolute ?<br>Allowed:Boolean vector<br>Default: All T |
| dopplerin | in | Input doppler type for velocities<br>Allowed:String<br>Default: 'radio' |
| unitsin | in | Input units<br>Allowed:String vector<br>Default: Native |
| absout | in | Are output coordinate elements absolute ?<br>Allowed:Boolean vector<br>Default: All T |
| dopplerout | in | Output doppler type for velocities<br>Allowed:String<br>Default: 'radio' |
| unitsout | in | Output units<br>Allowed:String vector<br>Default: Native |
| shape | in | Image shape<br>Allowed:Integer vector<br>Default: Unused |

**Returns**

Numeric vector or fail

**Description**

This function converts between mixed pixel/world/abs/rel numeric coordinates. The input and output coordinates are specified via a numeric vector giving coordinate values, a string vector giving units, a boolean vector specifying whether the coordinate is absolute or relative (to the reference pixel) and doppler strings specifying the doppler convention for velocities.
The units string may include `pix` for pixel coordinates and velocity units (i.e. any unit consistent with `m/s`).
The allowed doppler strings and definition are described in function summary.
The `shape` argument is optional. If your Coordinate System from an image, then assign the image shape to this argument. It is used only when making mixed (pixel/world) conversions for Direction Coordinates to resolve ambiguity.
The example will make the use of this function clear.

**Example**

```
- cs := coordsys(direction=T, spectral=T)    # 3 axes
-
- cout := cs.convert (coordin=[10,20,30],
                      absin=[T,T,T],
                      unitsin="pix pix pix",
                      absout=[T,F,T],
                      dopplerout='optical',
                      unitsout="pix arcsec km/s");
-
- print 'cout=', cout
[10 1140.0058 1134.82336]
```

In this example we convert from a vector of absolute pixels to a mixture of pixel/world and abs/rel.

---

**coordsys.convertmany - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Convert many numeric mixed coordinates

**Synopsis**
convertmany(coordin, absin, dopplerin, unitsin, absout, dopplerout, unitsout, shape)

#### Arguments

| | | |
|---|---|---|
| coordin | in | Input coordinate<br>Allowed:Numeric matrix |
| absin | in | Are input coordinate elements absolute ?<br>Allowed:Boolean vector<br>Default: All T |
| dopplerin | in | Input doppler type for velocities<br>Allowed:String<br>Default: 'radio' |
| unitsin | in | Input units<br>Allowed:String vector<br>Default: Native |
| absout | in | Are output coordinate elements absolute ?<br>Allowed:Boolean vector<br>Default: All T |
| dopplerout | in | Output doppler type for velocities<br>Allowed:String<br>Default: 'radio' |
| unitsout | in | Output units<br>Allowed:String vector<br>Default: Native |
| shape | in | Image shape<br>Allowed:Integer vector<br>Default: Unused |

#### Returns

Numeric matrix or fail

#### Description

This function converts between many mixed pixel/world/abs/rel numeric coordinates. See function convert for more information.
The only diffference with that function is that you provide a matrix holding many coordinates to convert and a matrix of many converted coordinates is returned.

#### Example

```
- cs := coordsys(direction=T, spectral=T)    # 3 axes
-
- p := [15,20,30];                           # absolute pixel coordinate
```

```
- cin := array (p, 3, 10);                 # 10 conversions each of length 3
- cin[3,] := 1:10;                         # spectral pixel runs from 1 to 10
- cin
[[1:3,]
    15 15 15 15 15 15 15 15 15 15
    20 20 20 20 20 20 20 20 20 20
    1  2  3  4  5  6  7  8  9  10]
-
- cout := cs.convertmany (coordin=cin,
                          absin=[T,T,T],
                          unitsin="pix pix pix",
                          absout=[T,F,T],
                          dopplerout='optical',
                          unitsout="pix deg km/s");
-
- cout
[[1:3,]
    15          15          15          15          15          15          15          15          15          15
    0.316668279 0.316668279 0.316668279 0.316668279 0.316668279 0.316668279 0.316668279 0.316668279 0.316668279 0.316668279
    1145.30291  1145.09023  1144.87756  1144.66488  1144.4522   1144.23953  1144.02685  1143.81418  1143.6015   1143.38883]
-
- cout[,1]                                   # First coordinate
[15 0.316668279 1145.30291]
-
- cout[,10]                                  # 10th coordinate
[15 0.316668279 1143.38883]
```

---

**coordsys.coordinatetype - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Return type of specified coordinate

**Synopsis**
coordinatetype(which)

**Arguments**

| | | |
|---|---|---|
| which | in | Which coordinate ?<br>Allowed:1-rel integer or unset<br>Default: All |

**Returns**
String or fail

**Description**

This function (short-hand name `ct`) returns a string describing the type of the specified coordinate. If `which=unset` the types for all coordinates are returned. Possible output values are 'Direction', 'Spectral', 'Stokes', 'Linear', and 'Tabular'

**Example**

```
- cs := coordsys(direction=T, spectral=T)
- cs.coordinatetype(1)
Direction
- cs.coordinatetype()
Direction Spectral
```

---

**coordsys.copy - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Copy this Coordsys tool

**Synopsis**
copy()

**Returns**
Coordsys tool or fail

**Description**

*Glish* uses copy-on-write semantics. This sometimes means that it maintains a reference between variables that you would rather it didn't.
This function returns a copy, not a reference, of the Coordsys `tool`. It is your responsibility to call the `done` function on the new `tool`.

### Example

```
- cs := coordsys()
- cs2 := cs            # Reference
- cs.done()
- cs
F
- cs2                  # cs2 gets doned when c2 does
F
-
- cs := coordsys()
- cs2 := cs.copy()      # Copy
- cs.done()
- cs
F
- cs2
ref [busy=<function>, result=<function>,...   # cs2 is still viable
- cs2.done()
- cs2                  # Now it's done
F
```

---

### coordsys.done - Function

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Destroy this Coordsys tool

**Synopsis**
done()

**Returns**
T or fail

### Description

If you no longer need to use a Coordsys `tool`, calling this function will free up its resources. That is, it destroys the `tool`. You can no longer call any functions on the `tool` after it has been `done`.

### Example

```
- cs := coordsys()
- cs.done()
- cs
F
```

---

### coordsys.epoch - Function

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Return the epoch

**Synopsis**
epoch()

**Returns**
Epoch measure or fail

### Description

This function (short-hand name `e`) returns the epoch of the observation as a Measure. You can format it with quanta

### Example

```
- cs := coordsys()
- ep := cs.epoch()
- ep
[type=epoch, refer=UTC, m0=[value=51728.4578, unit=d]]
-
- time := dm.getvalue(ep)           # Extract time with measures
- dq.time(time)                     # Format with quanta
10:59:10.170
```

---

### coordsys.findaxis - Function

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Find specified axis in coordinate system

**Synopsis**

findaxis(coordinate, axisincoordinate, world, axis)

**Arguments**

| | | |
|---|---|---|
| coordinate | out | Coordinate number<br>Allowed:Integer |
| axisincoordinate | out | Axis in the coordinate<br>Allowed:Integer |
| world | in | is axis a world or pixel axis ?<br>Allowed:Bool<br>Default: T |
| axis | in | Axis in coordinate system<br>Allowed:Integer<br>Default: 1 |

**Returns**

T, F or fail

**Description**

This function (short-hand name **fa**) finds the specified axis in the Coordinate System. If the axis does not exist, it returns F (not fail).

**Example**

```
- cs := coordsys(direction=T, linear=2)          # RA/DEC/Lin1/Lin2
- local coord, axisincoord;
- cs.findcoordinate(coord, axisincoord, T, 2)    # DEC
- print coord, axisincoord
1 2
- cs.findcoordinate(coord, axisincoord, T, 3)    # Lin1
- print coord, axisincoord
2 1
```

In these examples, the Coordinate System has 4 axes (both world and pixel the same). The first example finds the DEC axis (coordinate system axis 2) to be the second axis of the Direction Coordinate (coordinate 1). The second example finds the first linear axis (coordinate system axis 3) to be the first axis of the Linear Coordinate (coordinate 2).

---

### coordsys.findcoordinate - Function

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Find axes of specified coordinate

#### Synopsis

findcoordinate(pixel, world, type, which)

#### Arguments

| | | |
|---|---|---|
| pixel | out | Pixel axes<br>Allowed:Vector of integers |
| world | out | World axes<br>Allowed:Vector of integers |
| type | in | Type of coordinate to find<br>Allowed:String from "direction", "stokes", "spectral", "linear", "tabular"<br>Default: "direction" |
| which | in | Which coordinate if more than one<br>Allowed:Integer<br>Default: First (1) |

#### Returns

T, F or fail

#### Description

This function (short-hand name **fc**) finds the axes in the Coordinate System for the specified coordinate (minimum match is active for argument `type`). By default it finds the first coordinate, but if there is more than one (can happen for linear coordinates), you can specify which. If the coordinate does not exist, it returns F (not fail).

See also the function axesmap which returns the mapping between pixel and world axes.

### Example

```
- cs := coordsys(direction=T)
- local pa, wa;
- cs.findcoordinate(pa, wa, 'direction')
- print 'pixel, world axes =', pa, wa
pixel, world axes = [1 2] [1 2]
```

---

### coordsys.frequencytofrequency - Function

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Convert frequency to frequency with an offset

### Synopsis

frequencytofrequency(value, frequnit, velocity, doppler)

### Arguments

| | | |
|---|---|---|
| value | in | Frequency to convert<br>Allowed:Numeric or vector of numeric |
| frequnit | in | Unit of input frequencies<br>Allowed:String<br>Default: Intrinsic units |
| velocity | in | Velocity offset<br>Allowed:Quantity<br>Default: None |
| doppler | in | Velocity doppler definition<br>Allowed:String<br>Default: 'radio' |

### Returns

Double or vector of double or fail

### Description

This function (short-hand name **ftf**) converts frequencies to frequencies by applying (subtracting) a velocity offset.
The input frequencies are specified via a vector of numeric values and a specified unit (`frequnit`). If you don't give a frequency unit, it is assumed that the units are those given by function units for the spectral coordinate.
This function does not make any frame conversions (e.g. LSR to BARY) but you can specifiy the velocity doppler definition for the velocity via the `doppler` argument (see summary for possible values).
This function will return a fail if there is no spectral coordinate in the Coordinate System. See also function frequencytovelocity.
This function presently uses an approximation only valid for velocities less than the speed of light. The correction tends to zero as the velocity appraches the speed of light. You will get NaNs for larger values of the velocity.

### Example

```
- cs := coordsys(spectral=T)
- idx := 1 + 0.1*(1:5)
- rv := cs.referencevalue()
- freq := idx * rv
-
- print 'In = ', freq
[1.5565e+09 1.698e+09 1.8395e+09 1.981e+09 2.1225e+09]
-
- print cs.frequencytofrequency(freq, velocity='100km/s')
[1.55598089e+09 1.6974337e+09 1.83888651e+09 1.98033932e+09 2.12179213e+09]
```

---

## coordsys.frequencytovelocity - Function

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Convert frequency to velocity

### Synopsis

frequencytovelocity(value, frequnit, doppler, velunit)

### Arguments

| | | |
|---|---|---|
| value | in | Frequency to convert<br>Allowed:Numeric or vector of numeric |
| frequnit | in | Unit of input frequencies<br>Allowed:String<br>Default: Intrinsic units |
| doppler | in | Velocity doppler definition<br>Allowed:String<br>Default: 'radio' |
| velunit | in | Unit of output velocities<br>Allowed:String<br>Default: 'km/s' |

### Returns

Double or vector of double or fail

### Description

This function (short-hand name **ftv**) converts frequencies to velocities.
The input frequencies are specified via a vector of numeric values and a specified unit (`frequnit`). If you don't give a frequency unit, it is assumed that the units are those given by function units for the spectral coordinate.
This function does not make any frame conversions (e.g. LSR to BARY) but you can specifiy the velocity doppler definition via the `doppler` argument (see summary for possible values).
The velocities are returned in a vector for which you specify the units (`velunit` - default is km/s).
This function will return a fail if there is no spectral coordinate in the Coordinate System. See also function velocitytofrequency.

### Example

```
- im := image('hcn.cube')
- cs := im.coordsys()
-
- local pa, wa;
- cs.findcoordinate(pa, wa, 'spectral')     # Find spectral axis
- pixel := cs.referencepixel();             # Use reference pixel for non-spectral
-
- nFreq := im.shape()[pa[1]];               # Length of spectral axis
- freq := [];
```

```
- for (i in 1:nFreq) {
    pixel[pa[1]] := i;                      # Assign value for spectral axis of pixel coordinate
    w := cs.toworld(value=pixel, format='n')# Convert pixel to world
    freq[i] := w[wa[1]];                    # Fish out frequency
  }
- vel := cs.frequencytovelocity(value=freq, doppler='optical', velunit='km/s')
- print vel
[137.868718 133.690916 129.513229 125.33566 121.158206 116.980869
 112.803649 108.626544 104.449556 100.272685 96.0959294 91.9192905
  87.7427679 83.5663616 79.3900717 75.213898 71.0378408 66.8618999
  62.6860752 58.5103668 54.3347748 50.159299 45.9839395 41.8086963
  37.6335694 33.4585588 29.2836644 25.1088863 20.9342244 16.7596788
  12.5852495  8.41093638 4.23673952 0.0626588884 -4.11130551 -8.28515368
 -12.4588856 -16.6325014 -20.8060009 -24.9793842 -29.1526513 -33.3258023
 -37.498837 -41.6717555 -45.8445579 -50.0172441 -54.1898141 -58.3622679
 -62.5346056 -66.7068272 -70.8789325 -75.0509218 -79.2227948 -83.3945518
 -87.5661926 -91.7377173 -95.9091259 -100.080418 -104.251595 -108.422655
-112.593599 -116.764427 -120.935139 -125.105735]
```

In this example, we find the optical velocity in km/s of every pixel along the spectral axis of our image. First we obtain the Coordinate System from the image. Then we find which axis of the Coordinate System (image) pertain to the spectral coordinate. Then we loop over each pixel of the spectral axis, and convert a pixel coordinate (one for each axis of the image) to world. We obtain the value for the spectral axis from that world vector, and add it to the vector of frequencies. Then we convert that vector of frequencies to velocity.

---

**coordsys.fromrecord - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Fill Coordinate System from a record

**Synopsis**

fromrecord(record)

**Arguments**

| | | |
|---|---|---|
| record | in | Record containing Coordinate System<br>Allowed:Glish record |

**Returns**
T or fail

**Description**

You can convert a Coordinate System to a record (torecord). This function (fromrecord) allows you to set the contents of an existing Coordinate System from such a record. In doing so, you overwrite its current contents.
These functions are not for general user use.

**Example**

```
- cs := coordsys(direction=T, stokes="I Q")
- cs.ncoordinates()
2
- r := cs.torecord()
-
- cs2 := coordsys()
- cs2.ncoordinates()
0
- cs2.fromrecord(r)
- cs2.ncoordinates()
2
```

---

**coordsys.id - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Return the fundamental identifier of this tool

**Synopsis**
id()

**Description**

This function should be of little interest to users. It returns the fundamental identifier of the `tool` used in the tasking system.

**Returns**
ToolID - a record

**Example**

```
- cs := coordsys()
- cs.id()
[sequence=5, pid=17277, time=962622872, host=phoenix, agentid=3]
```

---

**coordsys.increment - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Recover the increments

**Synopsis**
increment(format, type)

**Arguments**

| | | |
|---|---|---|
| format | in | Format string<br>Allowed: String from combination of "n", "q", "s", "m"<br>Default: 'n' |
| type | in | Coordinate type<br>Allowed: String from "direction", "stokes", "spectral", "linear", "tabular" or unset<br>Default: All |

**Returns**
Vector of doubles, vector of quantities, vector of string, record or fail

**Description**

Each axis associated with the Coordinate System has a reference value, reference pixel and an increment (per pixel). These are used in the mapping from pixel to world coordinate.

This function (short-hand name `i`) returns the increment (in world axis order). You can recover the increments either for all coordinates (leave `type` unset) or for a specific coordinate type (mimumum match of the allowed types will do). If you ask for a non-existent coordinate an exception is generated.
See the discussion regarding the formatting possibilities available via argument `format`.
You can set the increment with function setincrement.

**Example**

```
- cs := coordsys(direction=T, spectral=T)
- cs.increment(format='q')
[__*0=[value=-1, unit='], __*1=[value=1, unit='], __*2=[value=1000, unit=Hz]]
-
- cs.increment(format='n')
[-1 1 1000]
-
- cs.increment(format='n', type='spectral')
1000
```

---

**coordsys.lineartransform - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Recover the linear transform matrix

**Synopsis**
lineartransform(type)

**Arguments**

| | | |
|---|---|---|
| type | in | Coordinate type<br>Allowed:String from "direction", "stokes", "spectral", "linear", "tabular" |

**Returns**
Vector of double or fail

**Description**

Recover the linear transform component for the specified coordinate type. You can set the linear transform with function setlineartransform.

**Example**

```
- cs := coordsys(direction=T, linear=3)
- cs.lineartransform('dir')                  # 2 x 2
[[1:2,]
    1 0
    0 1]
- cs.lineartransform('lin')                  # 3 x 3
[[1:3,]
    1 0 0
    0 1 0
    0 0 1]
```

---

**coordsys.names - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Recover the names for each axis

**Synopsis**
names(type)

**Arguments**

| | | |
|---|---|---|
| type | in | Coordinate type<br>Allowed: String from "direction", "stokes", "spectral", "linear", "tabular" or unset<br>Default: All |

**Returns**
Vector of strings or fail

**Description**

Each axis associated with the Coordinate System has a name (they don't mean anything fundamental). This function returns those names in world axis order. You can recover the names either for all coordinates (leave `type` unset) or for a specific coordinate type (mimumum match of the allowed types will do). If you ask for a non-existent coordinate an exception is generated.
You can set the names with function setnames.

**Example**

```
- cs := coordsys(direction=T, spectral=T)
- n := cs.names(T)
- n[1]
Right Ascension
- n[2]
Declination
- n[3]
Frequency
-
- cs.names('spec')
Frequency
```

---

**coordsys.naxes - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Recover the number of axes

**Synopsis**
naxes(world)

**Arguments**

| | | |
|---|---|---|
| world | in | Find number of world or pixel axes ?<br>Allowed:T or F<br>Default: T |

**Returns**
Int or fail

**Description**

Find the number of axes in the Coordinate System.
You may find the number of world or pixel axes; these are generally the same and general users can ignore the distinction. See the discussion about pixel and world axis ordering.

**Example**

```
- cs := coordsys(direction=T, spectral=T)
- n := cs.naxes(T)
3                          # 2 direction axes, 1 spectral
- n := cs.naxes(F)
3
```

---

**coordsys.ncoordinates - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Recover the number of coordinates in the Coordinate System

**Synopsis**
ncoordinates()

**Returns**
Integer or fail

**Description**

This function (short-hand name `nc`) recovers the number of coordinates in the Coordinate System.

**Example**

```
- cs := coordsys(direction=T, spectral=T)
- cs.ncoordinates()
2
- cs2 := coordsys(linear=4)
- cs2.ncoordinates()
1
```

---

---

**coordsys.observer - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Return the name of the observer

**Synopsis**
observer()

**Returns**
String or fail

**Description**

This function returns the name of the observer. You can set it with the function setobserver.

**Example**

```
- cs := coordsys()
- cs.observer()
Karl Jansky
```

---

**coordsys.projection - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Recover the direction coordinate projection

**Synopsis**
projection(type)

**Arguments**

| | | |
|---|---|---|
| type | in | Type of projection<br>Allowed:String<br>Default: Current |

**Returns**
Record, Vector string, Int or fail

**Description**

If the Coordinate System contains a direction coordinate, this function (short-hand name `p`) can be used to recover information about the projection. For discussion about celestial coordinate systems, including projections, see the papers by Mark Calabretta and Eric Greisen. The initial draft from 1996 (implemented in AIPS++Background information can be found here.
What this function returns depends upon the value you assign to `type`.

- `type=unset`. In this case (the default), the actual projection type and projection parameters are returned in a *Glish* record with fields `type` and `parameters`, respectively.
- `type='all'`. In this case, a vector of strings containing all of the possible projection codes is returned.
- `type=code`. If you specify a valid projection type code (see list by setting `type='all'`) then what is returned is the number of parameters required to describe that projection (useful in function setprojection).

You can change the projection with setprojection.
If the Coordinate System does not contain a direction coordinate, an exception is generated.

**Example**

```
- cs := coordsys(direction=T)
- cs.projection()
[type=SIN, parameters=[0 0] ]
-
- cs.projection('all')
AZP TAN SIN STG ARC ZPN ZEA AIR CYP CAR MER CEA COP
COD COE COO BON PCO GLS PAR AIT MOL CSC QSC TSC
-
- cs.projection('ZPN')
10
```

We first recover the projection type and parameters from the direction coordinate. Then we find the list of all possible projection types. FInally, we recover the number of parameters required to describe the 'ZPN' projection.

---

**coordsys.referencecode - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Return specified reference code

**Synopsis**
referencecode(type, list)

**Arguments**

| | | |
|---|---|---|
| type | in | Which coordinate type ?<br>Allowed:String from "direction", "stokes", "spectral", "linear", "tabular" or unset<br>Default: All |
| list | in | List all possibilities ?<br>Allowed:Bool<br>Default: F |

**Returns**
Vector of strings, string or fail

**Description**

This function (short-hand name `rp`) returns the reference code for all, or the specified coordinate type. Examples of the reference code are B1950 and J2000 for direction coordinates, or LSRK and BARY for spectral coordinates.
If `type` is left unset, then a vector of strings is returned, one code for each coordinate type in the Coordinate System.
If you specify `type` then select from 'direction', 'spectral', 'stokes', and 'linear' (the first two letters will do). However, only the first two coordinate types will return a non-empty string. If the Coordinate System does not contain a coordinate of the type you specify, an exception is generated.
The argument `list` is ignored unless you specify a specific `type`. If `list=T`, then this function returns the list of all possible reference codes for the specified coordinate type. Otherwise, it just returns the actual code current set in the Coordinate System.
The list of all possible types is returned as a *Glish* record (it is actually generated by the listcodes function in the measures system). This record has two fields. These are called 'normal' (containing all normal codes) and 'extra' (maybe empty, with all extra codes like planets).
You can set the reference code with setreferencecode.

### Example

```
- cs := coordsys(direction=T)
- list := cs.referencecode('dir', T)
- list.normal[1:10]
J2000 JMEAN JTRUE APP B1950 BMEAN BTRUE GALACTIC HADEC AZEL
-
- cs.referencecode('dir')
J2000
```

In this example we first get the list of all possible reference codes for a direction coordinate (and print some of the normal ones). Then we get the actual reference code for the direction coordinate in our Coordinate System.

---

### coordsys.referencepixel - Function

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Recover the reference pixel

### Synopsis

referencepixel(type)

### Arguments

| | | |
|---|---|---|
| type | in | Coordinate type<br>Allowed: String from "direction", "stokes", "spectral", "linear", "tabular" or unset<br>Default: All |

### Returns

Vector of double or fail

### Description

Each axis associated with the Coordinate System has a reference value, reference pixel and an increment (per pixel). These are used in the mapping from pixel to world coordinate.
This function (short-hand name `rp`) returns the reference pixel (in pixel axis order). You can recover the reference pixel either for all coordinates (leave `type` unset) or for a specific coordinate type (mimumum match of the allowed types will do). If you ask for a non-existent coordinate an exception is generated.
You can set the reference pixel with function setreferencepixel.

### Example

```
- cs := coordsys(spectral=T, linear=2)
- cs.setreferencepixel([1.0, 2.0, 3.0])
- cs.referencepixel()
[1 2 3]
-
- cs.referencepixel('lin')
[2 3]
```

---

**coordsys.referencevalue - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Recover the reference value

**Synopsis**
referencevalue(format, type)

**Arguments**

| | | |
|---|---|---|
| format | in | Format string<br>Allowed:String from combination of ”n”, ”q”, ”s”, ”m”<br>Default: ’n’ |
| type | in | Coordinate type<br>Allowed:String from ”direction”, ”stokes”, ”spectral”, ”linear”, ”tabular” or unset<br>Default: All |

**Returns**
Vector of doubles, vector of quantities, vector of string, record or fail

**Description**

Each axis associated with the Coordinate System has a reference value, reference pixel and an increment (per pixel). These are used in the mapping from pixel to world coordinate.
This function (short-hand name `rv`) returns the reference value (in world axis order). You can recover the reference value either for all coordinates (leave `type` unset) or for a specific coordinate type (mimumum match of the allowed types will do). If you ask for a non-existent coordinate an exception is generated.
See the discussion regarding the formatting possibilities available via argument `format`.
You can set the reference value with function setreferencevalue.

**Example**

```
- cs := coordsys(direction=T, spectral=T)
- cs.referencevalue(format='q')
[__*0=[value=0, unit='], __*1=[value=0, unit='], __*2=[value=1.415e+09, unit=Hz]]
-
- cs.referencevalue(format='n')
[0 0 1.415e+09]
-
```

```
- cs.referencevalue(format='n', type='spec')
1.415e+09
```

---

**coordsys.reorder - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Reorder the coordinates

**Synopsis**
reorder(order)

**Arguments**

| | | |
|---|---|---|
| order | in | New coordinate order<br>Allowed:Vector of integers |

**Returns**
Vector of quantities or doubles, or fail

**Description**

This function reorders the coordinates in the Coordinate System. You specify the new order of the coordinates in terms of their old order.

**Example**

```
- cs := coordsys(direction=T, spectral=T, linear=2)
- cs.coordinatetype()
Direction Spectral Linear
-
- cs.reorder([2,3,1]);
- cs.coordinatetype()
Spectral Linear Direction
```

---

**coordsys.replace - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Replace a coordinate

**Synopsis**
replace(csys, whichin, whichout)

**Arguments**

| | | |
|---|---|---|
| csys | in | Coordinate System to replace from<br>Allowed:Coordsys tool |
| whichin | in | Index of input coordinate<br>Allowed:Integer |
| whichout | in | Index of output coordinate<br>Allowed:Integer |

**Returns**
T or fail

**Description**

This function replaces on coordinate in the current Coordinate System by one coordinate in the given Coordinate System. The specified coordinates must have the same number of axes.

**Example**

```
- cs := coordsys(direction=T, linear=1)
- cs.coordinatetype()
Direction Linear
- cs2 := coordsys(spectral=T)
- cs2.replace (cs2,2,1)
- cs.coordinatetype()
Direction Spectral
```

---

**coordsys.restfrequency - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Recover the rest frequency

**Synopsis**
restfrequency()

**Returns**
Quantity or fail

**Description**

If the Coordinate System contains a spectral coordinate, then it has a rest frequency. In fact, the spectral coordinate can hold several rest frequencies (to handle for example, an observation where the band covers many lines), although only one is active (for velocity conversions) at a time.
This function (short-hand name `rf`) recovers the rest frequencies as a quantity vector. The first frequency is the active one.
You can change the rest frequencies with setrestfrequency.
If the Coordinate System does not contain a frequency coordinate, an exception is generated.

**Example**

```
- cs := coordsys(spectral=T)
- cs.restfrequency()
[value=1.42040575e+09, unit=Hz]
-
- cs.setrestfrequency (value=[1.2e9. 1.3e9], which=2, append=F)
- cs.restfrequency()
[value=[1.3e+09 1.2e+09] , unit=Hz]
```

In the example, the initial spectral coordinate has 1 rest frequency. Then we set it with two, nominating the second as the active rest frequency, and recover them.

---

**coordsys.setconversiontype - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Set extra reference conversion layer

**Synopsis**
setconversiontype(direction, spectral)

**Arguments**

| | | |
|---|---|---|
| direction | in | Reference code<br>Allowed:String or unset<br>Default: None |
| spectral | in | Reference code<br>Allowed:String or unset<br>Default: None |

**Returns**
T, F or fail

**Description**

Some coordinates contain a reference code. Examples of reference codes are B1950 and J2000 for direction coordinates, or LSRK and BARY for spectral coordinates. When you do conversions between pixel and world coordinate, the coordinates are in the reference frame corresponding to these codes.
This function allows you to specify a different reference frame which is used when converting between world and pixel coordinate (see function conversiontype to recover the conversion types). If it returns F, it means that although the conversion machines were successfully created, a trial conversion failed. This usually means the REST frame was involved which requires a radial velocity (not yet implemented). If this happens, the conversion type will left as it was. You will get a fail if more blatant things are wrong like a missing coordinate, or an incorrect reference code.
The list of possible reference codes can be obtained via function referencecode.
With this function, you specify the desired reference code. Then, when a conversion between pixel and world is requested, an extra conversion is done to (`toWorld`) or from (`toPixel`) the specified reference frame.

The summary function shows the extra conversion reference system to the right of the native reference system (if it is different) and in parentheses. Note that to convert between different spectral reference frames, you need a position, epoch and direction. The position (telescope) and epoch (date of observation), if not in your coordinate system can be set with functions settelescope and setepoch. The direction is the reference direction of the *required* direction coordinate in the coordinate system.
As an example, let us say you are working with a spectral coordinate which was constructed with the LSRK reference frame. You want to convert some pixel coordinates to barycentric velocities (reference code BARY).

```
- cs := coordsys(direction=T, spectral=T); # Create coordinate system
- local pa, wa;
- cs.findcoordinate(pa, wa, 'spectral')    # Find spectral coordinate
- u := cs.units()[wa[1]];                  # Spectral unit
- cs.referencecode(type='spectral')        # Which is  in LSRK reference frame
LSRK
-
- p := [10,20,30]
- w := cs.toworld(p, format='n')           # Convert a pixel to LSRK world
- print 'pixel, world = ', p, w
pixel, world =  [10 20 30]  [21590.9999 19.0000967 1.415029e+09]
-
- p2 := cs.topixel(w)                      # and back to pixel
- print 'world, pixel = ', w, p2
world, pixel =  [21590.9999 19.0000967 1.415029e+09]  [10 20 30]
-
- v := cs.frequencytovelocity(value=w[wa[1]],     # Convert LSRK frequency to LSRK velocity
                              frequnit=u,
                              doppler='RADIO',
                              velunit='m/s');
- print 'pixel, frequency, velocity = ', p[pa[1]], w[wa[1]], v
pixel, frequency, velocity =  30 1.415029e+09 1134823.36
-
-
- cs.setconversiontype(spectral='BARY')    # Specify BARY reference code
- w := cs.toworld(p, format='n')           # Convert a pixel to BARY world
print 'pixel, world = ', p, w
pixel, world =  [10 20 30]  [21590.9999 19.0000967 1.41498179e+09]
-
- p2 := cs.topixel(w)                      # and back to pixel
- print 'world, pixel = ', w, p2
world, pixel =  [21590.9999 19.0000967 1.41498179e+09]  [10 20 29.9952772]
-
- v := cs.frequencytovelocity(value=w[wa[1]],   # Convert BARY frequency to BARY velocity
                              frequnit=u,
```

```
                              doppler='RADIO',
                              velunit='m/s');
- print 'pixel, frequency, velocity = ', p[pa[1]], w[wa[1]], v
pixel, frequency, velocity =  30 1.41498179e+09 1144787.26
```

You must also be aware of when this extra layer is active and when it is not. It's a bit nasty.

- \- Whenever you use `toWorld`, `toPixel` `toWorldMany`, or `toPixelMany` the layer is active.
- \- Whenever you use `convert` or `convertMany` the layer *may* be active. Here are the rules !

  It is only relevant to spectral and direction coordinates.

  For the direction coordinate part of your conversion, if you request a pure world or pixel conversion it is active. Any pixel/world mix will not invoke it (because it is ill defined).

  For the spectral coordinate part it is always active (only one axis so must be pixel or world).
- \- This layer is irrelevant to all functions converting between frequency and velocity, and absolute and relative. The values are in whatever frame you are working with.

The

summary function lists the reference frame for direction and spectral coordinates. If you have also set a conversion reference code it also lists that (to the right in parentheses).

---

**coordsys.setdirection - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Set direction coordinate values

**Synopsis**
setdirection(refcode, proj, projpar, refpix, refval, incr, xform, poles)

**Arguments**

| | | |
|---|---|---|
| refcode | in | Reference code<br>Allowed:String<br>Default: No change |
| proj | in | Projection type<br>Allowed:String<br>Default: No change |
| projpar | in | Projection parameters<br>Allowed:Numeric vector<br>Default: No change |
| refpix | in | Reference pixel<br>Allowed:Numeric vector<br>Default: No change |
| refval | in | Reference value<br>Allowed:Vector of doubles, vector of quantities, quantity of vector of doubles, vector of string, record<br>Default: No change |
| incr | in | Increment<br>Allowed:Vector of doubles, vector of quantities, quantity of vector of doubles, vector of string, record<br>Default: No change |
| xform | in | Linear transform<br>Allowed:2-D numeric array<br>Default: No change |
| poles | in | Native poles<br>Allowed:Vector of doubles, vector of quantities, quantity of vector of doubles, vector of string, record<br>Default: No change |

**Returns**
T or fail

**Description**

When you construct a Coordsys `tool`, if you include a Direction Coordinate, it will have some default parameters. This function simply allows you to replace the values of the Direction Coordinate.
You can also change almost all of those parameters (such as projection, reference value etc.) via the individual functions setreferencecode, setprojection, setreferencepixel, setreferencevalue, setincrement, and setlineartransform provided by the Coordsys `tool`. See those functions for

more details about the formatting of the above function arguments.
Bear in mind, that if your Coordinate System came from a real image, then the reference pixel is special and you should not change it.

### Example

```
- cs := coordsys(direction=T);
- cs.setdirection (refcode='GALACTIC', proj='SIN', projpar=[0,0],
                   refpix=[-10,20], refval="10deg -20deg");
-
- cs.projection()
[type=SIN, parameters=[0 0] ]
- cs.referencepixel()
[-10 20]
- cs.referencevalue(format='s')
10.000000 deg -20.000000 deg
```

---

## coordsys.setepoch - Function

| | |
|---:|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Set a new epoch

### Synopsis
setepoch(value)

### Arguments

| | | |
|---|---|---|
| value | in | New epoch<br>Allowed:Epoch measure |

### Returns
T or fail

### Description

This function (short-hand name `se`) sets a new epoch (supplied as an epoch measure) of the observation. You can get the current epoch with function epoch.

**Example**

```
- cs := coordsys()
- ep := cs.epoch()
- ep
[type=epoch, refer=UTC, m0=[value=51728.4578, unit=d]]
-
- ep := dm.epoch('UTC', 'today')
- cs.setepoch(ep)
- cs.epoch()
[type=epoch, refer=UTC, m0=[value=51729.1679, unit=d]]
```

---

**coordsys.setincrement - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Set the increment

**Synopsis**
setincrement(value, type)

**Arguments**

| | | |
|---|---|---|
| value | in | Increments<br>Allowed: Vector of doubles, vector of quantities, quantity of vector of doubles, vector of string, record |
| type | in | Which coordinate type ?<br>Allowed: String from "direction", "stokes", "spectral", "linear", "tabular" or unset<br>Default: All |

**Returns**
T or fail

**Description**

Each axis associated with the Coordinate System has a reference value, reference pixel and an increment (per pixel). These are used in the mapping from pixel to world coordinate.
This function (short-hand name `si`) allows you to set a new increment. You should not do this on "stokes" axes unless you are an adept or a big risk taker.
You can set the increments either for all axes (`type=unset`) or for just the axes associated with a particular coordinate type.
You may supply the increments in all of the formats described in the formatting discussion.
In addition, you can also supply the increments as a quantity of vector of doubles. For example `dq.quantity([-1,2],'arcsec')`.
You can recover the current increments with function increment.

**Example**

```
- cs := coordsys(direction=T, spectral=T)
- rv := cs.increment(format='q')
- rv
[__*0=[value=-1, unit='], __*1=[value=1, unit='], __*2=[value=1000, unit=Hz]]
-
- rv2 := dq.quantity('4kHz');
- cs.setincrement(value=rv2, type='spec')
- cs.increment(type='spec', format='q')
[value=4000, unit=Hz]
-
- cs.increment(format='q')
[__*0=[value=-1, unit='], __*1=[value=1, unit='], __*2=[value=4000, unit=Hz]]
-
- cs.setincrement (value="-2' 2' 2e4Hz")
- cs.increment(format='q')
[__*0=[value=-2, unit='], __*1=[value=2, unit='], __*2=[value=20000, unit=Hz]]
```

In the example we first recover the increments as a vector of quantities. We then create a quantity for a new value for the spectral coordinate increment. Note we use units of kHz whereas the spectral coordinate is currently expressed in units of Hz. We then set the increment for the spectral coordinate. We then recover the increment again; you can see 4kHz has been converted to 4000Hz. We also show how to set the increment using a string interface.

---

**coordsys.setlineartransform - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Set the linear transform

**Synopsis**
setlineartransform(type, value)

**Arguments**

| | | |
|---|---|---|
| type | in | Which coordinate type ?<br>Allowed:String from "direction", "stokes", "spectral", "linear", "tabular"<br>Default: All |
| value | in | Linear transform<br>Allowed:Array of double |

**Returns**
T or fail

**Description**

This function set the linear transform component. For Stokes Coordinates this function will return T but do nothing.
You can recover the current linear transform with function lineartransform.

**Example**

```
- cs := coordsys(spectral=T, linear=3)
- xf := cs.lineartransform('lin')
- xf
[[1:3,]
    1 0 0
    0 1 0
    0 0 1]
- xf[1,2] := 0.01
- cs.slt(xf,'lin')
- cs.lt('lin')
```

```
[[1:3,]
    1 0.01 0
    0 1    0
    0 0    1]
```

---

**coordsys.setnames - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Set the axis names

**Synopsis**
setnames(value, type)

**Arguments**

| | | |
|---|---|---|
| value | in | Names<br>Allowed:Vector of strings |
| type | in | Which coordinate type ?<br>Allowed:String from "direction", "stokes", "spectral", "linear", "tabular" or unset<br>Default: All |

**Returns**
T or fail

**Description**

Each axis associated with the Coordinate System has a name. It isn't used in any fundamental way.
This function (short-hand name `sn`) allows you to set new axis names.
You can set the names either for all axes (`type=unset`) or for just the axes associated with a particular coordinate type.
You can recover the current axis names with function names.

**Example**

```
- cs := coordsys(spectral=T, linear=2)
- cs.setnames(value="a b c")
- cs.names()
a b c
-
- cs.setnames('lin', "flying fish")
- cs.names()
a flying fish
```

---

**coordsys.setobserver - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Set a new observer

**Synopsis**

setobserver(value)

**Arguments**

| | | |
|---|---|---|
| value | in | New observer<br>Allowed:String |

**Returns**

T or fail

**Description**

If you want grab all the glory, or transfer the blame, this function (short-hand name `so`) sets a new observer of the observation. You can get the current observer with function observer. The observer's name is not fundamental to the Coordinate System !

**Example**

```
- cs := coordsys()
- cs.observer()
Karl Jansky
- cs.setobserver('Ronald Biggs')
- cs.observer()
Ronald Biggs
```

---

**coordsys.setprojection - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Set the direction coordinate projection

**Synopsis**
setprojection(type, parameters)

**Arguments**

| | | |
|---|---|---|
| type | in | Type of projection<br>Allowed:String |
| parameters | in | Projection parameters<br>Allowed:Vector of double<br>Default: [] |

**Returns**
T or fail

**Description**

If the Coordinate System contains a direction coordinate, this function (short-hand name `sp`) can be used to set the projection. For discussion about celestial coordinate systems, including projections, see the papers by Mark Calabretta and Eric Greisen. The initial draft from 1996 (implemented in AIPS++) can be found here.
You can use the function projection to find out all the possible types of projection. You can also use it to find out how many parameters you need to describe a particular projection. See Calabretta and Greisen for details about

those parameters (see section 4 of their paper); in FITS terms these parameters are what are labelled as PROJP.
Some brief help here on the more common projections in astronomy.

- SIN has either 0 parameters or 2. For coplanar arrays like East-West arrays, one can use what is widely termed the NCP projection. This is actually a SIN projection where the parameters are 0 and $1/tan(\delta_0)$ where $\delta_0$ is the reference declination. Images made from the ATNF's Compact Array with AIPS++ will have such a projection. Otherwise, the SIN projection requires no parameters (but you can give it two each of which is zero if you wish).
- TAN is used widely in optical astronomy. It requires 0 parameters.
- ZEA (zenithal equal area) is used widely in survey work. It requires 0 parameters.

If the Coordinate System does not contain a direction coordinate, an exception is generated.

**Example**

```
- im := image('cena')
- cs := im.coordsys()
- cs.projection()
[type=SIN, parameters=[0 0] ]
-
- cs.projection('ZEA')
0
- cs.setprojection('ZEA')
- im2 := im.regrid('cena.zea', csys=cs)
```

We change the projection of an image from SIN to ZEA (which requires no parameters).

---

**coordsys.setreferencecode - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Set new reference code

**Synopsis**
setreferencecode(value, type, adjust)

**Arguments**

| | | |
|---|---|---|
| value | in | Reference code<br>Allowed:String |
| type | in | Which coordinate type ?<br>Allowed:String from "direction" or "spectral" |
| adjust | in | Adjust reference value ?<br>Allowed:Bool<br>Default: T |

**Returns**
T or fail

**Description**

This function (short-hand name `src`) sets the reference code for the specified coordinate type. Examples of reference codes are B1950 and J2000 for direction coordinates, or LSRK and BARY for spectral coordinates.
You must specify `type`, selecting from 'direction', or 'spectral' (the first two letters will do). If the Coordinate System does not contain a coordinate of the type you specify, an exception is generated.
Specify the new code with argument `value`. To see the list of possible codes, use the function referencecode (see example).
If `adjust` is T, then the reference value is recomputed. This is invariably the correct thing to do. If `adjust` is F, then the reference code is simply overwritten; do this very carefully.

**Example**

```
- cs := coordsys(direction=T)
- list := cs.referencecode('dir', T)      # See possibilities
- list.normal[1:10]
J2000 JMEAN JTRUE APP B1950 BMEAN BTRUE GALACTIC HADEC AZEL
-
- cs.referencecode('dir')
J2000
- cs.setreferencecode('B1950', 'dir', T)
```

In this example we first get the list of all possible reference codes for a direction coordinate (and print some of the normal ones). Then we set the actual reference code for the direction coordinate in our Coordinate System.

**Example**

```
- cs := coordsys(direction=T)
- list := cs.referencecode('dir', T)      # See possibilities
- list.normal[1:10]
J2000 JMEAN JTRUE APP B1950 BMEAN BTRUE GALACTIC HADEC AZEL
-
- cs.referencecode('dir')
J2000
- cs.setreferencecode('B1950', 'dir', T)
```

In this example we first get the list of all possible reference codes for a direction coordinate (and print some of the normal ones). Then we set the actual reference code for the direction coordinate in our Coordinate System.

**Example**

```
- im := image('myimage.j2000')                        # Open image
- cs := im.coordsys()               # Get Coordinate System
- cs.referencecode('dir', F)
J2000
- cs.setreferencecode('B1950', 'dir', T)              # Set new direction system
- im2 := im.regrid(outfile='myimage.b1950', csys=cs)  # Regrid and make new image
```

In this example we show how to regrid an image from J2000 to B1950. First we recover the Coordinate System into the Coordsys `tool` called **cs**. We then set a new direction reference code, making sure we recompute the reference value. Then the new Coordinate System is supplied in the regridding process (done with an Image `tool`).

---

**coordsys.setreferencelocation - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Set reference pixel and value

**Synopsis**
setreferencelocation(pixel, world, mask)

**Arguments**

| | | |
|---|---|---|
| pixel | in | New reference pixel<br>Allowed: Vector of integer<br>Default: Old reference pixel |
| world | in | New reference value<br>Allowed: Vector of doubles, vector of quantities, quantity of vector of doubles, vector of string, record<br>Default: Old reference value |
| mask | in | Indicates which axes to center<br>Allowed: Vector of boolean<br>Default: All |

**Returns**
T or fail

**Description**

This function (short-hand name `srl`) sets the reference pixel and reference value to the specified values. The world coordinate can be specified in any of the formats that the output world coordinate is returned in by the toworld function.
You can specify a mask (argument `mask`) indicating which pixel axes are set (T) and which are left unchanged (F). This function will refuse to change the reference location of a Stokes axis (gets you into trouble otherwise).
This function can be rather useful when regridding images. It allows you to keep easily a particular feature centered in the regridded image.

**Example**

```
- cs := coordsys(linear=2)
- cs.referencepixel()
[1 1]
- cs.referencevalue()
```

```
[0 0]
-
- w := cs.toworld([20,20], format='n')
- shp := [129,129]
- p := (shp-1)/2.0 + 1;
- cs.setreferencelocation (pixel=p, world=w)
- cs.referencepixel()
[65 65]
- cs.referencevalue()
[19 19]
```

---

**coordsys.setreferencepixel - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Set the reference pixel

**Synopsis**
setreferencepixel(type, value)

**Arguments**

| | | |
|---|---|---|
| type | in | Which coordinate type ?<br>Allowed:String from "direction", "stokes", "spectral", "linear", "tabular" or unset<br>Default: All |
| value | in | Reference pixel<br>Allowed:Vector of double |

**Returns**
T or fail

**Description**

Each axis associated with the Coordinate System has a reference value, reference pixel and an increment (per pixel). These are used in the mapping from pixel to world coordinate.

This function (short-hand name `srp`) allows you to set a new reference pixel. You should not do this on "stokes" axes unless you are an adept or a big risk taker.
You can set the reference pixel either for all axes (`type=unset`) or for just the axes associated with a particular coordinate type.
Bear in mind, that if your Coordinate System came from a real image, then the reference pixel is special and you should not change it for Direction Coordinates.
You can recover the current reference pixel with function referencepixel.

**Example**

```
- cs := coordsys(spectral=T, linear=2)
- cs.setreferencepixel(value=[1.0, 2.0, 3.0])
- cs.referencepixel()
[1 2 3]
-
- cs.setreferencepixel('lin', [-1, -1])
- cs.referencepixel()
[1 -1 -1]
```

---

**coordsys.setreferencevalue - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Set the reference value

**Synopsis**
setreferencevalue(value, type)

**Arguments**

| | | |
|---|---|---|
| value | in | Reference value<br>Allowed: Vector of doubles, vector of quantities, quantity of vector of doubles, vector of string, record |
| type | in | Which coordinate type ?<br>Allowed: String from "direction", "stokes", "spectral", "linear", "tabular" or unset<br>Default: All |

**Returns**
T or fail

**Description**

Each axis associated with the Coordinate System has a reference value, reference pixel and an increment (per pixel). These are used in the mapping from pixel to world coordinate.
This function (short-hand name `srv`) allows you to set a new reference value. You should not do this on "stokes" axes unless you are an adept or a big risk taker.
You may supply the reference value in all of the formats described in the formatting discussion.
You can recover the current reference value with function referencevalue.
Note that the value argument should be one of the specified possibilitioes. Especially a `measure` will be accepted, but will have a null effect, due to the interpretation as a generic record.

**Example**

```
- cs := coordsys(direction=T, spectral=T)
- rv := cs.referencevalue(format='q')
- rv
[__*0=[value=0, unit='], __*1=[value=0, unit='], __*2=[value=1.415e+09, unit=Hz]]
-
- rv2 := rv[3];
- rv2.value := 2.0e9;
- cs.setreferencevalue(type='spec', value=rv2)
- cs.referencevalue(format='n')
[0 0 2e+09]
```

To set a new direction reference value, the easiest way, given a direction measure `dr` would be:

```
- cs.setreferencevalue(type='direction', value=dm.getvalue(dr))
```

In the example we first recover the reference value as a vector of quantities. We then select out the value for the spectral coordinate, change it, and set it back in the Coordinate System. We then recover the reference value again, but this time just as a vector of doubles. In the case of the dircetion, we set the value from a known direction measure.

---

**coordsys.setrestfrequency - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Set the rest frequency

**Synopsis**
setrestfrequency(value, which, append)

**Arguments**

| | | |
|---|---|---|
| value | in | New rest frequencies<br>Allowed:Quantity, double or string vector |
| which | in | Which is the active rest frequency<br>Allowed:Integer<br>Default: 1 |
| append | in | Append this list or overwrite ?<br>Allowed:Bool<br>Default: F |

**Returns**
T or fail

**Description**

If the Coordinate System contains a spectral coordinate, then it has a rest frequency. In fact, the spectral coordinate can hold several rest frequencies (to handle for example, an observation where the band covers many lines), although only one is active (for velocity conversions) at a time.
This function (short-hand name `srf`) allows you to set new rest frequencies. You can provide the rest frequency as a quantity, a quantity string, or a double (units of current rest frequency assumed).

You specify whether the list of frequencies will be appended to the current list or whether it will replace that list. You must select which of the frequencies will become the active one. By default its the first in the list. The index refers to the final list (either appended or replaced).
You can recover the current rest frequencies with restfrequency.
If the Coordinate System does not contain a frequency coordinate, an exception is generated.

**Example**

```
- cs := coordsys(spectral=T)
- cs.restfrequency()
[value=1.42040575e+09, unit=Hz]
-
- cs.setrestfrequency(dq.quantity('1.4GHz'))
- cs.restfrequency()
[value=1.4e+09, unit=Hz]
-
- cs.setrestfrequency(1.3e9)
- cs.restfrequency()
[value=1.3e+09, unit=Hz]
-
- cs.setrestfrequency (value=[1.2e9, 1.3e9], which=2)
- cs.restfrequency()
[value=[1.3e+09 1.2e+09] , unit=Hz]
-
- cs.setrestfrequency (dq.quantity([1,2],'GHz'), which=4, append=T)
- cs.restfrequency()
[value=[2e+09 1.2e+09 1.3e+09 1e+09] , unit=Hz]
-
- cs.setrestfrequency ("1.4E9Hz 1667MHz")
- cs.restfrequency()
[value=[1.4e+09 1.667e+09] , unit=Hz]
```

---

**coordsys.setspectral - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Set tabular values for the spectral coordinate

**Synopsis**

setspectral(refcode, restfreq, frequencies, doppler, velocities)

**Arguments**

| | | |
|---|---|---|
| refcode | in | Reference code<br>Allowed:String<br>Default: No change |
| restfreq | in | Rest frequency<br>Allowed:Quantum or double<br>Default: No change |
| frequencies | in | Vector of frequencies<br>Allowed:Quantum vector<br>Default: No change |
| doppler | in | Doppler type<br>Allowed:String<br>Default: No change |
| velocities | in | Vector of velocitiestypes<br>Allowed:Quantum vector<br>Default: No change |

**Returns**

T or fail

**Description**

When you construct a Coordsys `tool`, if you include a Spectral Coordinate, it will be linear in frequency. This function allows you to replace the Spectral Coordinate by a finite table of values. Coordinate conversions between pixel and world are then done by interpolation.

You may specify either a vector of frequencies or velocities. If you specify frequencies, you can optionally specify a (new) reference code (see function setreferencecode for more details) and rest frequency (else the existing ones will be used).

If you specify velocities, you can optionally specify a (new) reference code and rest frequency (else the existing ones will be used). You must also give the doppler type (see function summary for more details). The velocities are then converted to frequency for creation of the Spectral Coordinate (which is fundamentally described by frequency).

You may specify the rest frequency as a Quantum or a double (native units of Spectral Coordinate used).

**Example**

```
- cs := coordsys(spectral=T);
-
- f1 := [1,1.01,1.03,1.4]
- fq := dq.quantity(f1, 'GHz')
- cs.setspectral(frequencies=fq)
- v := cs.frequencytovelocity(f1, 'GHz', 'radio', 'km/s')
- print 'v=', v
[88731.3175 86620.7061 82399.4832 4306.86125]
- vq := dq.quantity(v, 'km/s')
- cs.setspectral(velocities=vq, doppler='radio')
- f2 := cs.velocitytofrequency(v, 'GHz', 'radio', 'km/s')
-
- print 'f1 = ', f1
[1 1.01 1.03 1.4]
- print 'f2 = ', f2
[1 1.01 1.03 1.4]
```

We make a linear Spectral Coordinate. Then overwrite it with a list of frequenices. Convert those values to velocity, then overwrite the coordinate starting with a list of velocities. Then convert the velocities to frequency and show we get the original result.

---

**coordsys.setstokes - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Set the Stokes types

**Synopsis**
setstokes(stokes)

**Arguments**

| | | |
|---|---|---|
| stokes | in | Stokes types<br>Allowed:Vector of strings |

**Returns**
T or fail

### Description

If the Coordinate System contains a Stokes Coordinate, this function allows you to change the Stokes types defining it. If there is no Stokes Coordinate, an exception is generated.
See the coordsys constructor to see the possible Stokes types you can set.
You can set the Stokes types with function setstokes.

### Example

```
- cs := coordsys(stokes='I V')
- cs.stokes()
I V
- cs.setstokes("XX RL")
- cs.stokes()
XX RL
```

---

### coordsys.settabular - Function

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Set tabular values for the tabular coordinate

### Synopsis

settabular(pixel, world, which)

### Arguments

| | | |
|---|---|---|
| pixel | in | Vector of (1-rel) pixel values<br>Allowed:Double vector<br>Default: No change |
| world | in | Vector of world values<br>Allowed:Double vector<br>Default: No change |
| which | in | Which Tabular coordinate<br>Allowed:Integer<br>Default: 1 |

**Returns**
T or fail

**Description**

When you construct a Coordsys `tool`, if you include a Tabular Coordinate, it will be linear. This function allows you to replace the Tabular Coordinate by a finite table of values. Coordinate conversions between pixel and world are then done by interpolation (or extrapolation beyond the end). The table of values must be at least of length 2 or an exception will occur.
You may specify a vector of pixel and world values (in the current units of the Tabular Coordinate). These vectors must be the same length. If you leave one of them unset, then the old values are used, but again, ultimately, the pixel and world vectors must be the same length.
The new reference pixel will be the first pixel value. The new reference value will be the first world value.
Presently, there is no way for you to recover the lookup table once you have set it.
If you have more than one Tabular Coordinate, use argument `which` to specify which one you want to modify.

**Example**

```
- cs := coordsys(tabular=T);
- cs.settabular (pixel=[1,10,15,20,100], world=[10,20,50,100,500])
```

We make a linear Tabular Coordinate. Then overwrite it with a non-linear list of pixel and world values.

---

**coordsys.settelescope - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Set a new telescope

**Synopsis**
settelescope(value)

**Arguments**

| | | |
|---|---|---|
| value | in | New telescope<br>Allowed:String |

**Returns**
T or fail

**Description**

This function (short-hand name **st**) sets a new telescope of the observation. The telescope position may be needed for reference code conversions; this is why it is maintained in the Coordinate System. So it is fundamental to the Coordinate System and should be correct.
You can find a list of the observatory names know to AIPS++ with the Measures obslist function.
You can get the current telescope with function telescope.

**Example**

```
- cs := coordsys()
- cs.telescope()
ATCA
- cs.settelescope('VLA')
- cs.telescope()
VLA
-
- cs.settelescope('The One In My Backyard')
WARN: This telescope is not known to the AIPS++ system
You can request that it be added
-
- dm.obslist()
ALMA ATCA BIMA CLRO DRAO DWL GB GBT JCMT MOPRA NRAO12M PKS VLA WSRT
```

---

**coordsys.setunits - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Set the axis units

**Synopsis**
setunits(value, type, overwrite, which)

**Arguments**

| | | |
|---|---|---|
| value | in | Units<br>Allowed:Vector of strings |
| type | in | Which coordinate type ?<br>Allowed:String from "direction", "stokes", "spectral", "linear", "tabular" or unset<br>Default: All |
| overwrite | in | Overwrite linear or tabular coordinate units ?<br>Allowed:Bool<br>Default: F |
| which | in | Which coordinate if more than one of same type<br>Allowed:Integer<br>Default: First |

**Returns**
T or fail

**Description**

Each axis associated with the Coordinate System has a unit. This function (short-hand name `su`) allows you to set new axis units.
You can set the units either for all axes (`type=unset`) or for just the axes associated with a particular coordinate type.
In general, the units must be consistent with the old units. When you change the units, the increment and reference value will be adjusted appropriately. However, for a linear or tabular coordinate, and only when you specify `type='linear'` or `type='tabular'` (i.e. you supply units only for the specified linear of tabular coordinate), and if you set `overwrite=T`, you can just overwrite the units with no further adjustments. Otherwise, the `overwrite` argument will be silently ignored. Use argument `which` to specify which coordinate if you have more than one of the specified type.
You can recover the current axis units with function units.

**Example**

```
- cs := coordsys(direction=T, spectral=T)
- cs.summary()
-
Direction reference : J2000
Spectral  reference : LSRK
Velocity  type      : RADIO
Rest frequency      : 1.42041e+09 Hz

Telescope        : ATCA
Observer         : Karl Jansky
Date observation : 2000/07/05/06:58:46

Axis Type      Name             Proj   Coord value at pixel    Coord incr Units
--------------------------------------------------------------------------------
1    Direction Right Ascension   SIN  00:00:00.000     1.00 -6.000000e+01 arcsec
2    Direction Declination      SIN +00.00.00.000     1.00  6.000000e+01 arcsec
3    Spectral  Frequency             1.415000e+09     1.00  1.000000e+03 Hz
               Velocity              1.140944e+03     1.00 -2.110611e-01 km/s
-
- cs.setunits(value="deg rad mHz");
-
-
Direction reference : J2000
Spectral  reference : LSRK
Velocity  type      : RADIO
Rest frequency      : 1.42041e+12 mHz

Telescope        : ATCA
Observer         : Karl Jansky
Date observation : 2000/07/05/06:58:46

Axis Type      Name             Proj   Coord value at pixel    Coord incr Units
--------------------------------------------------------------------------------
1    Direction Right Ascension   SIN  00:00:00.000     1.00 -6.000000e+01 arcsec
2    Direction Declination      SIN +00.00.00.000     1.00  6.000000e+01 arcsec
3    Spectral  Frequency             1.415000e+12     1.00  1.000000e+06 mHz
               Velocity              1.140944e+03     1.00 -2.110611e-01 km/s
```

---

**coordsys.stokes - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Recover the Stokes types

**Synopsis**
stokes()

**Returns**
Vector of strings or fail

**Description**

If the Coordinate System contains a Stokes Coordinate, this function recovers the Stokes types defining it. If there is no Stokes Coordinate, an exception is generated.
You can set the Stokes types with function setstokes.

**Example**

```
- cs := coordsys(stokes='I V')
- cs.stokes()
I V
```

---

**coordsys.summary - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Summarize basic information about the Coordinate System

**Synopsis**
summary(doppler, list)

**Arguments**

| | | |
|---|---|---|
| doppler | in | List velocity information with this doppler definition<br>Allowed:String<br>Default: RADIO |
| list | in | List to global logger<br>Allowed:Boolean<br>Default: T |

**Returns**
T, fail, or Vector of strings

**Description**

This function (short-hand name `s`) summarizes the information contained in the Coordinate System.
For spectral coordinates, the information is listed as a velocity as well as a frequency. The argument `doppler` allows you to specify what doppler convention it is listed in. You can choose from `radio, optical` and `beta`. Alternative names are `z` for `optical`, and `relativistic` for `beta`. The default is `radio`. The definitions are

- radio: $1 - F$
- optical: $-1 + 1/F$
- beta: $(1 - F^2)/(1 + F^2)$

where $F = \nu/\nu_0$ and $\nu_0$ is the rest frequency. If the rest frequency has not been set in your image, you can set it with the function setrestfrequency.
These velocity definitions are provided by the measures system via the Doppler measure (see example).
If you set `list=F`, then the summary will not be written to the global logger. However, the return value will be a vector of strings holding the summary information, one string per line of the summary.
For direction and spectral coordinates, the reference frame (e.g. J2000 or LSRK) is also listed. Along side this, in parentheses, will be the conversion reference frame as well (if it is different from the native reference frame). See function setconversion to see what this means.

**Example**

```
- d := dm.doppler('beta')
- dm.listcodes(d)
[normal=RADIO Z RATIO BETA GAMMA OPTICAL TRUE RELATIVISTIC, extra=]
```

```
-
- cs := coordsys(direction=T, spectral=T)
- cs.summary()
-
Direction reference : J2000
Spectral  reference : LSRK (-> BARY)
Velocity  type      : RADIO
Rest frequency      : 1.42041e+09 Hz

Telescope        : ATCA
Observer         : Karl Jansky
Date observation : 2000/07/04/10:01:11

Axis Type      Name             Proj   Coord value at pixel    Coord incr Units
--------------------------------------------------------------------------------
1    Direction Right Ascension   SIN  00:00:00.000     1.00 -6.000000e+01 arcsec
2    Direction Declination       SIN +00.00.00.000     1.00  6.000000e+01 arcsec
3    Spectral  Frequency              1.415000e+09     1.00  1.000000e+03 Hz
               Velocity               1.140944e+03     1.00 -2.110611e-01 km/s
```

---

**coordsys.telescope - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Return the telescope

**Synopsis**
telescope(measure)

**Arguments**

| | | |
|---|---|---|
| measure | in | Return as position measure<br>Allowed:T or F<br>Default: F |

**Returns**
String or position measure or fail

### Description

This function (short-hand name `t`) returns the telescope contained in the Coordinate System. It can be returned as a simple string (`measure=F`) or as a position measure (`measure=T`).
The telescope position may be needed for reference code conversions; this is why it is maintained in the Coordinate System.
The conversion from string to position is done with observatory. The example shows you what is really going on.

### Example

```
- cs := coordsys()
- cs.telescope()
ATCA
- cs.telescope(T)
[type=position, refer=ITRF, m2=[value=6372960.26, unit=m],
m1=[unit=rad, value=-0.52613792], m0=[unit=rad, value=2.61014232]]
-
- dm.observatory(cs.telescope())
[type=position, refer=ITRF, m2=[value=6372960.26, unit=m],
m1=[unit=rad, value=-0.52613792], m0=[unit=rad, value=2.61014232]]
```

We get the telescope as a string and as a position measure. We also show how the Measures system was used to convert from the simple name to a position Measure.

---

### coordsys.toabs - Function

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Convert relative coordinate to absolute

### Synopsis

toabs(value, isworld)

### Arguments

| | | |
|---|---|---|
| value | in | Relative coordinate<br>Allowed:Numeric vector, vector string, vector quantum, record |
| isworld | in | Is coordinate world or pixel ?<br>Allowed:T, F, unset<br>Default: unset |

**Returns**

Same as input or fail

**Description**

This function converts a relative coordinate to an absolute coordinate. The coordinate may be a pixel coordinate or a world coordinate.
If the coordinate is a pixel coordinate, it is supplied as a numeric vector. If the coordinate is a world coordinate, you may give it in all of the formats described in the formatting discussion.
If the coordinate value is supplied by a Coordsys `tool` function (e.g. toworld) then the coordinate 'knows' whether it is world or pixel (and absolute or relative). However, you might supply the value from some other source as a numeric vector (which could be world or pixel) in which case you must specify whether it is a world or pixel coordinate via the `isworld` argument.

**Example**

```
- cs := coordsys(direction=T, spectral=T)
-
- aw := cs.toworld([100,100,24], 's')
- rw := cs.torel(aw)
- aw2 := cs.toabs(rw)
-
- aw
23:53:23.781 +01.39.00.821 1.415023e+09 Hz
- rw
5.943285e+03 arcsec 5.940821e+03 arcsec 2.300000e+04 Hz
- aw2
23:53:23.781 +01.39.00.821 1.415023e+09 Hz
```

This example uses world coordinates.

**Example**

```
- cs := coordsys(direction=T, spectral=T)
-
- ap := cs.topixel()          # Reference value
- rp := cs.torel(ap)
- ap2 := cs.toabs(rp)
-
- ap
[1 1 1]
- rp
[0 0 0]
- ap2
[1 1 1]
```

This example uses pixel coordinates.

---

**coordsys.toabsmany - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Convert many numeric relative coordinates to absolute

**Synopsis**
toabsmany(value, isworld)

**Arguments**

| | | |
|---|---|---|
| value | in | Relative coordinates<br>Allowed:Numeric matrix |
| isworld | in | Is coordinate world or pixel ?<br>Allowed:T, F, unset<br>Default: unset |

**Returns**
Numeric matrix or fail

**Description**

This function converts many relative coordinates to absolute. It exists so you can efficiently make many conversions (which would be rather slow if you did

them all with toabs). Because speed is the object, the interface is purely in terms of numeric matrices, rather than being able to accept strings and quanta etc. like toabs can.
When dealing with world coordinates, the units of the numeric values must be the native units, given by function units.

#### Example

```
- cs := coordsys(direction=T, spectral=T)    # 3 axes
-
- rv := cs.referencevalue();                 # reference value
- w := cs.torel(rv)                          # make relative
- inc := cs.increment();                     # increment
- idx := 1:100;
- off := inc[3] * idx;                       # offset for third axis
- wrel := array (w, 3, 100);                 # 100 conversions each of length 3
- wrel[3,] +:= off;                          # Make spectral axis values change
-
- wabs  := cs.toabsmany (rel, T)             # Convert
- wabs[,1]                                   # First absolute coordinate
[0 0 1.415001e+09]
- wabs[,100]                                 # 100th absolute coordinate
[0 0 1.415002e+09]
```

This example uses world coordinates.

### coordsys.topixel - Function

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Convert from absolute world to pixel coordinate

#### Synopsis

topixel(value)

#### Arguments

| | | | |
|---|---|---|---|
| value | in | Absolute world coordinate | |
| | | Allowed: | Numeric vector, vector of quantities, vector of strings or record |
| | | Default: | Reference value |

**Returns**
Vector of double or fail

**Description**

This function converts between world (physical) coordinate and absolute pixel coordinate (1-rel).
The world coordinate can be provided in one of four formats via the argument `world`. These match the output formats of function toworld.
If you supply fewer world values than there are axes in the Coordinate System, your coordinate vector will be padded out with the reference value for the missing axes. Excess values will be silently ignored.
You may supply the world coordinate in all of the formats described in the formatting discussion.

**Example**

```
- cs := coordsys(direction=T, spectral=T, stokes="I V", linear=2)
-
- w := cs.toworld([-2,2,1,2,23,24], 'n')
- cs.topixel(w)
[-2 2 1 2 23 24]
-
- w := cs.toworld([-2,2,1,2,23,24], 'q')
- cs.topixel(w)
[-2 2 1 2 23 24]
-
- w := cs.toworld([-2,2,1,2,23,24], 'm')
- cs.topixel(w)
[-2 2 1 2 23 24]
-
- w := cs.toworld([-2,2,1,2,23,24], 's')
- cs.topixel(w)
[-2 2 1 2 23 24]
-
- w := cs.toworld([-2,2,1,2,23,24], 'mnq')
- cs.topixel(w)
```

```
[-2 2 1 2 23 24]
```

**Example**

```
- cs := coordsys (stokes="I V", linear=2)
- cs.toworld([1,2,3], 's')
I 1.000000e+00 2.000000e+00
-
- cs.toworld([1,2,3], 'm')
[stokes=I, linear=[*1=[value=1, unit=km], *2=[value=2, unit=km]]]
-
- cs.toworld([1,2,3], 'q')
[*1=[value=1, unit=], *2=[value=1, unit=km], *3=[value=2, unit=km]]
```

**Example**

```
- cs := coordsys (spectral=T, linear=1)
- cs.toworld([1,2,3], 'q')
[*1=[value=1.415e+09, unit=Hz], *2=[value=1, unit=km]]
```

---

**coordsys.topixelmany - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Convert many absolute numeric world coordinates to pixel

**Synopsis**
topixelmany(value)

**Arguments**

| | | |
|---|---|---|
| value | in | Absolute world coordinates<br>Allowed:Numeric matrix |

**Returns**
Numeric matrix or fail

**Description**

This function converts many absolute world coordinates to pixel coordinates. It exists so you can efficiently make many conversions (which would be rather slow if you did them all with topixel). Because speed is the object, the interface is purely in terms of numeric matrices, rather than being able to accept strings and quanta etc. like topixel can.
The units of the numeric values must be the native units, given by function units.

**Example**

```
- cs := coordsys(direction=T, spectral=T)    # 3 axes
-
- rv := cs.referencevalue();                 # reference value
- inc := cs.increment();                     # increment
- idx := 0:99;
- off := inc[3] * idx;                       # offset for third axis
-
- wabs := array (rv, 3, 100);                # 100 conversions each of length 3
- wabs[3,] +:= off;                          # Make spectral axis values change
-
- pabs  := cs.topixelmany (wabs);            # Convert
- pabs[,1]                                   # First absolute pixel coordinate
[1 1 1]
- pabs[,100]                                 # 100th absolute pixel coordinate
[1 1 100]
```

**coordsys.torecord - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Convert Coordinate System to a record

**Synopsis**
torecord()

**Returns**
Glish record or fail

**Description**

You can convert a Coordinate System to a record with this function. There is also fromrecord to set a Coordinate System from a record.
These functions are not for general user use; they are used for sending Coordsys `tools` about the *Glish* bus.

**Example**

```
- cs := coordsys(direction=T, stokes="I Q")
- rec := cs.torecord();
- cs2 := coordsys();
- cs2.fromrecord(rec);
-
- print cs.ncoordinates(), cs2.ncoordinates()
2 2
```

---

**coordsys.torel - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Convert absolute coordinate to relative

**Synopsis**
torel(value, isworld)

**Arguments**

| | | |
|---|---|---|
| value | in | Absolute coordinate<br>Allowed:Numeric vector, vector string, vector quantum, record |
| isworld | in | Is coordinate world or pixel ?<br>Allowed:T, F, unset<br>Default: unset |

**Returns**
Same as input or fail

**Description**

This function converts an absolute coordinate to a relative coordinate. The coordinate may be a pixel coordinate or a world coordinate.
Relative coordinates are relative to the reference pixel (pixel coordinates) or the reference value (world coordinates) in the sense
$relative = absolute - reference$.
If the coordinate is a pixel coordinate, it is supplied as a numeric vector. If the coordinate is a world coordinate, you may give it in all of the formats described in the formatting discussion.
If the coordinate value is supplied by a Coordsys `tool` function (e.g. toworld) then the coordinate 'knows' whether it is world or pixel (and absolute or relative). However, you might supply the value from some other source as a numeric vector (which could be world or pixel) in which case you must specify whether it is a world or pixel coordinate via the `isworld` argument.

**Example**

```
- cs := coordsys(direction=T, spectral=T)
-
- aw := cs.toworld([100,100,24], 's')
- rw := cs.torel(aw)
- aw2 := cs.toabs(rw)
-
- aw
23:53:23.781 +01.39.00.821 1.415023e+09 Hz
- rw
5.943285e+03 arcsec 5.940821e+03 arcsec 2.300000e+04 Hz
- aw2
23:53:23.781 +01.39.00.821 1.415023e+09 Hz
```

This example uses world coordinates.

**Example**

```
- cs := coordsys(direction=T, spectral=T)
-
- ap := cs.topixel()           # Reference value
```

```
- rp := cs.torel(ap)
- ap2 := cs.toabs(rp)
-
- ap
[1 1 1]
- rp
[0 0 0]
- ap2
[1 1 1]
```

This example uses pixel coordinates.

---

**coordsys.torelmany - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Convert many numeric absolute coordinates to relative

**Synopsis**
torelmany(value, isworld)

**Arguments**

| | | |
|---|---|---|
| value | in | Absolute coordinates<br>Allowed:Numeric matrix |
| isworld | in | Is coordinate world or pixel ?<br>Allowed:T, F, unset<br>Default: unset |

**Returns**
Numeric matrix or fail

**Description**

This function converts many absolute coordinates to relative. It exists so you can efficiently make many conversions (which would be rather slow if you did them all with torel). Because speed is the object, the interface is purely in terms of numeric matrices, rather than being able to accept strings and quanta etc. like torel can.

When dealing with world coordinates, the units of the numeric values must be the native units, given by function units.

**Example**

```
- cs := coordsys(direction=T, spectral=T)    # 3 axes
-
- w := cs.referencevalue();                  # reference value
- inc := cs.increment();                     # increment
- idx := 0:99;
- off := inc[3] * idx;                       # offset for third axis
- wabs := array (w, 3, 100);                 # 100 conversions each of length 3
- wabs[3,] +:= off;                          # Make spectral axis values change
-
- wrel  := cs.torelmany (wabs, T)            # Convert
- wrel[,1]                                   # First relative coordinate
[0 0 0]
- wrel[,100]                                 # 100th relative coordinate
[0 0 99000]
```

This example uses world coordinates.

**coordsys.toworld - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Convert from absolute pixel coordinate to world

**Synopsis**
toworld(value, format)

**Arguments**

| | | |
|---|---|---|
| value | in | Absolute pixel coordinate<br>Allowed:Numeric vector<br>Default: Reference pixel |
| format | in | Format string<br>Allowed:String from combination of "n", "q", "s", "m"<br>Default: 'n' |

**Returns**
Vector numeric, vector quantity, vector string, record, or fail

**Description**

This function converts between absolute pixel coordinate (1-rel) and absolute world (physical coordinate).
If you supply fewer pixel values than there are axes in the Coordinate System, your coordinate vector will be padded out with the reference pixel for the missing axes. Excess values will be silently ignored.
You may ask for the world coordinate in all of the formats described in the discussion regarding the formatting possibilities available via argument `format`.

**Example**

```
- cs := coordsys(direction=T, spectral=T)
-
- cs.toworld([-2,2,2], 'n')
[3.00000051 1.00000001 1.415001e+09]
-
- cs.toworld([-2,2,2], 'q')
[*1=[value=3.00000051, unit='],
 *2=[value=1.00000001, unit='],
 *3=[value=1.415001e+09, unit=Hz]]
-
- cs.toworld([-2,2,2], 'm')
[direction=[type=direction, refer=J2000, m1=[value=0.000290888213, unit=rad],
                                         m0=[unit=rad, value=0.000872664774]],
 spectral=[frequency=[type=frequency, refer=LSRK, m0=[value=1.415001e+09, unit=Hz]],
           radiovelocity=[value=1140.73308, unit=km/s],
           opticalvelocity=[value=1145.09023, unit=km/s],
           betavelocity=[value=1142.90335, unit=km/s]]]
-
- cs.toworld([-2,2,2], 's')
00:00:12.000 +00.01.00.000 1.415001e+09
```

**Example**

```
- cs := coordsys (stokes="I V", linear=2)
- cs.toworld([1,2,3], 's')
I 1.000000e+00 2.000000e+00
```

```
-
- cs.toworld([1,2,3], 'm')
[stokes=I, linear=[*1=[value=1, unit=km], *2=[value=2, unit=km]]]
-
- cs.toworld([1,2,3], 'q')
[*1=[value=1, unit=], *2=[value=1, unit=km], *3=[value=2, unit=km]]
```

**Example**

```
- cs := coordsys (spectral=T, linear=1)
- cs.toworld([1,2,3], 'q')
[*1=[value=1.415e+09, unit=Hz], *2=[value=1, unit=km]]
```

---

**coordsys.toworldmany - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Convert many absolute pixel coordinates to numeric world

**Synopsis**
toworldmany(value)

**Arguments**

| | | |
|---|---|---|
| value | in | Absolute pixel coordinates<br>Allowed:Numeric matrix |

**Returns**
Numeric matrix or fail

**Description**

This function converts many absolute pixel coordinates to world coordinates. It exists so you can efficiently make many conversions (which would be rather slow if you did them all with toworld). Because speed is the object, the

interface is purely in terms of numeric matrices, rather than being able to produce strings and quanta etc. like toworld can.
The units of the output world values are the native units given by function units.

#### Example

```
- cs := coordsys(direction=T, spectral=T)    # 3 axes
-
- rp := cs.referencepixel();                 # reference pixel
- off := 0:99;                               # offset for third axis
-
- pabs := array (rp, 3, 100);                # 100 conversions each of length 3
- pabs[3,] +:= off;                          # Make spectral axis values change
-
- wabs  := cs.toworldmany (pabs);            # Convert
- wabs[,1]                                   # First absolute pixel coordinate
[0 0 1.415e+09]
- wabs[,100]                                 # 100th absolute pixel coordinate
[0 0 1.415099e+09]
```

### coordsys.type - Function

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Return the type of this tool

#### Synopsis
type()

#### Returns
String

#### Description

This function returns the string 'coordsys'.

---

### coordsys.units - Function

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Recover the units for each axis

**Synopsis**
units(type)

**Arguments**

| | | |
|---|---|---|
| type | in | Coordinate type<br>Allowed: String from 'direction', 'stokes', 'spectral', 'linear' or unset<br>Default: All |

**Returns**
Vector of strings or fail

**Description**

Each axis associated with the Coordinate System has a unit. This function (short-hand name `u`) returns those units (in world axis order).
You can recover the units either for all coordinates (leave `type` unset) or for a specific coordinate type (mimumum match of the allowed types will do). If you ask for a non-existent coordinate an exception is generated.
You can set the units with function setunits.

**Example**

```
- cs := coordsys(direction=T, spectral=T)
- cs.units(T)
' ' Hz
-
- cs.units('spec')
Hz
```

---

**coordsys.velocitytofrequency - Function**

| | |
|---|---|
| **Package** | **general** |
| **Module** | **images** |
| **Tool** | **coordsys** |

Convert velocity to frequency

**Synopsis**

velocitytofrequency(value, frequnit, doppler, velunit)

**Arguments**

| | | |
|---|---|---|
| value | in | Velocity to convert<br>Allowed:Numeric or vector of numeric |
| frequnit | in | Unit of output frequencies<br>Allowed:String<br>Default: Intrinsic units |
| doppler | in | Velocity doppler definition<br>Allowed:String<br>Default: 'radio' |
| velunit | in | Unit of input velocities<br>Allowed:String<br>Default: 'km/s' |

**Returns**

Double or vector of double or fail

**Description**

This function (short-hand name **vtf**) converts velocities to frequencies.
The input velocities are specified via a vector of numeric values, a specified unit (`velunit`), and a velocity doppler definition (`doppler`).
The frequencies are returned in a vector for which you specify the units (`frequnit`). If you don't give the unit, it is assumed that the units are those given by function units for the spectral coordinate.
This function will return a fail if there is no spectral coordinate in the Coordinate System. See also the function frequencytovelocity.

**Example**

```
- im := image('hcn.cube')
- cs := im.coordsys()
-
- local pa, wa;
- cs.findcoordinate(pa, wa, 'spectral')     # Find spectral axis
- pixel := cs.referencepixel();             # Use reference pixel for non-spectral
-
- nFreq := im.shape()[pa[1]];               # Length of spectral axis
- freq := [];
- for (i in 1:nFreq) {
    pixel[pa[1]] := i;                      # Assign value for spectral axis of pixel coordinate
    w := cs.toworld(value=pixel, format='n')# Convert pixel to world
    freq[i] := w[wa[1]];                    # Fish out frequency
  }
- vel := cs.frequencytovelocity(value=freq, doppler='optical', velunit='km/s')
- freq2 := cs.velocitytofrequency(value=vel, doppler='optical', velunit='km/s')
```

In this example, we find the optical velocity in km/s of every pixel along the spectral axis of our image. First we obtain the Coordinate System from the image. Then we find which axis of the Coordinate System (image) pertain to the spectral coordinate. Then we loop over each pixel of the spectral axis, and convert a pixel coordinate (one for each axis of the image) to world. We obtain the value for the spectral axis from that world vector, and add it to the vector of frequencies. Then we convert that vector of frequencies to velocity. Then we convert it back to frequency. They better agree.

---

---

### 1.2.11 is_coordsys - Function

**Package** general
**Module** images

Is this variable a Coordsys tool ?

**Synopsis**
is_coordsys(thing)

**Arguments**

| | | |
|---|---|---|
| thing | in | Variable to test<br>Allowed:Glish variable<br>Default: None |

**Returns**
T or F

**Description**

Determine if this variable is a Coordsys tool. This is very useful in scripts; for robustness, you check that the `tool` is a Coordsys `tool` before you start applying Coordsys `tool functions`.

**Example**

```
cs := coordsys();
is_coordsys(cs)
T
cs2 := 'doggies'
is_coordsys(cs2)
F
```

---

### 1.2.12 coordsystools - Function

**Package** general
**Module** images

Find the list of current global Coordsys tools

**Synopsis**
coordsystools()

**Returns**
Vector of Strings

**Description**
This function retuns a list of the names of the Coordsys `tools` currently in Global name space.

**Example**

```
cs1 := coordsys();
im := shapetoimage(shape=[10,10])
cs2 := im.coordsys();
coordsystools()
cs1 cs2
```

---

### 1.2.13 coordsystest - Function

**Package** general
**Module** images

Run test suite for Coordsys tool

**Synopsis**
coordsystest(which)

**Arguments**

| | | |
|---|---|---|
| which | in | Which test to run<br>Allowed:Integer or vector of integers<br>Default: All tests |

**Returns**
T or F

**Description**

Self-test of the Coordsys `tool`. This function is quite extensive and includes forced errors. As long as the function finally returns with a `T`, it has succeeded (regardless of what error messages you might see). Note that one of the tests is if the created coordinate system tools match the actual list of tools. If a coordinate system has been created outside the test, the tests will fail. The `coordsystools` function can be used to check for pre-existing tools. They can be removed by re-assigning them. E.g.:

```
cs := T
```

**Example**