RUTHERFORD APPLETON LABORATORY
Particle Physics & Astronomy Research Council
Gemini Project
Gemini TCS Note (PTW) 8.4 (TN_PS_G0045)

TCS/PTW/8.4 (TN_PS_G0045)

P. T. Wallace[1]

25th September 1997

World Coordinates

Contents

[1]with acknowledgements to David Terrett, who advised on the EPICS parts of Section 3 and wrote the `astlib` functions `astGetctx`, `astGetpo` and `astSetctx`.

1 INTRODUCTION

This document describes World Coordinate System (WCS) facilities on Gemini. "World Coordinates" are celestial coordinates as opposed to instrumental coordinates. The facilities described here help to address such problems as:

- What is the J2000 $[\alpha, \delta]$ of this pixel?
- Where on my detector is the image of this B1950 $[\alpha, \delta]$ star?
- Which direction on my picture is north?
- What headers should I use in my FITS files?

In addition, there is a general-purpose celestial coordinate transformation facility that can answer such questions as:

- What are the J2000 coordinates of this equinox B1950 epoch 1978.2 star?
- To what $[Az, El\,]$ should my telescope mount be set in order to image that J2000 $[\alpha, \delta]$ on this pixel of my detector?

New readers may find it easiest to start by looking through the *Programming Examples* section at the end, which hints at the problems being addressed and the available techniques for dealing with them. Previewing the section on *The Telescope Context* is also worthwhile, as this explains where the basic information comes from that ties down the WCS. By that stage, the *General Principles* section can be tackled, and finally the introduction to the *Astrometry Software* section. The remaining material is intended mainly for reference.

2 GENERAL PRINCIPLES

2.1 The Basics

The Gemini WCS facilities are based on the assumption that, for astrometric purposes, users of the telescope are concerned with only two classes of coordinates:

- Positions in the focal plane.
- Directions in the sky.

The WCS facilities provide for transformations between these coordinates. All intermediate calculations, for example involving the position of M2, the orientation of the instrument rotator or the amount of atmospheric dispersion, are encapsulated in the transformations which the TCS/WCS facilities provide.

The Gemini focal-plane coordinate systems are shown in Figure 1. The $[x, y]$ coordinate system is fixed to the instrument mount and rotates with the instrument, about the origin. The sky coordinates supported are three sorts of $[\alpha, \delta]$ (FK4, FK5 and geocentric apparent place) plus topocentric $[Az, El\,]$.

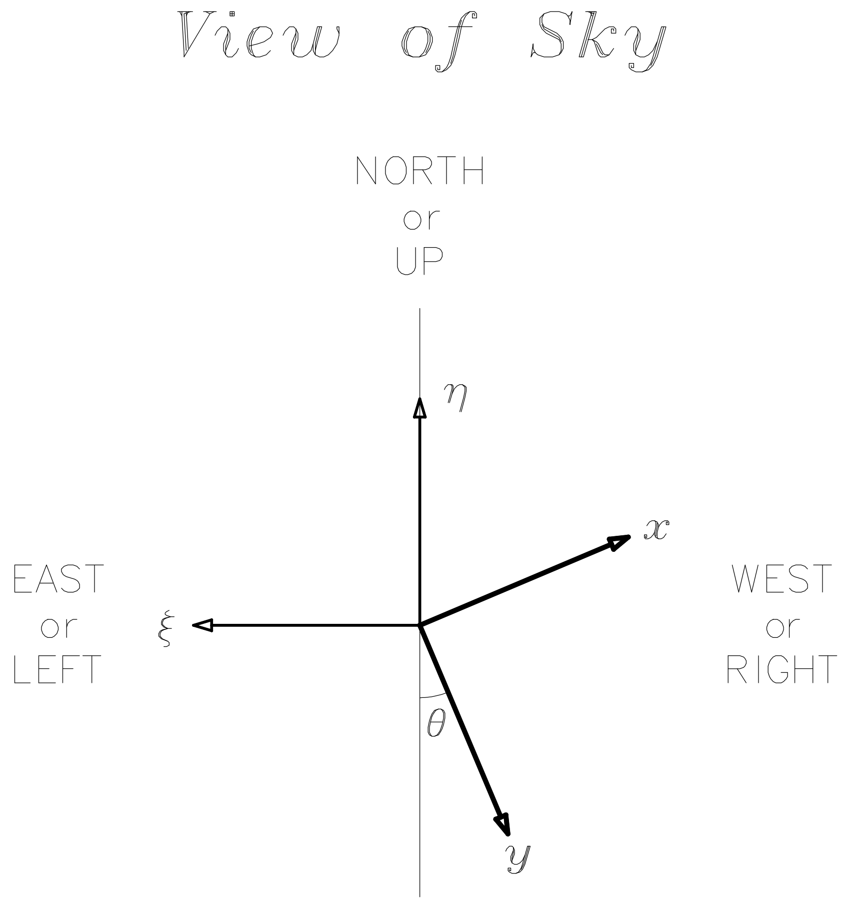


Figure 1: **Gemini Focal-Plane Coordinate Systems.**

This is a view looking up at the sky; the view looking at the back of the instrument rotator from outside the telescope is the same apart from the field being upside-down.

- θ is the user-specified rotator position-angle. It can be in terms of any of the supported sky coordinate systems, for example with respect to J2000 $[\,\alpha, \delta\,]$. This *user-specified* position-angle is essentially the same as the *mechanical* position-angle if the nominated coordinate system is $[\,Az, El\,]$.
- x and y are the focal-plane coordinates used by the WCS facilities. They rotate with the instrument mount.
- ξ and η are the so-called "standard coordinates"; they are related to right ascension and declination (or azimuth and elevation) through standard formulae.

The WCS facilities described in this paper provide for transformations between $[\,x, y\,]$ and $[\,\xi, \eta\,]$. There is also support for using local instrumental coordinates (*e.g.* pixels) instead of $[\,x, y\,]$.

2.2 Sky Coordinates

The relationships between different celestial coordinate systems, though well-documented, are recondite and subtle. For example the relationship between B1950 $[\alpha, \delta]$ and J2000 $[\alpha, \delta]$ is much more than the 50 years of precession that makes up the bulk of the transformation. The coordinate rotation itself comes from several different sources, involving two distinct precession models and an intermediate epoch. Furthermore, the two systems are in relative motion, so that distant objects such as QSOs have proper motions when expressed in B1950 coordinates. Yet another complication is that the B1950 system involves an overall distortion on top of the pure rotation, due to the E-terms of aberration.

A more obvious departure from any simple-minded scheme – for example one based only on (i) where the optical axis of the telescope is pointing, (ii) the image scale and (iii) rotator angle – is the effect of refraction. Quite apart from dispersion effects (an imaginary white J2000 *réseau* on the celestial sphere would appear in the focal plane fringed with false colour), the whole picture is compressed in the vertical direction.

The effect of all of these complications – subtle in the case of B1950 versus J2000, gross in the case of refraction – are properly handled by the TCS, and the WCS transformations that the TCS supports include their effects. All the user has to do is to nominate which celestial coordinate system to use (and to specify the working wavelength), whereupon efficient transformations between those coordinates and $[x, y]$, in either direction, are available through easy-to-use function calls. However, a fundamental difficulty with any scheme that tries to achieve this is the inescapable trade-off between simplicity and efficiency on the one hand and accuracy on the other. At one extreme lies the "simple-minded" scheme mentioned earlier, where the basic formulae of spherical astronomy are applied directly. The other extreme is to carry out an exhaustive and rigorous transformation for every desired point.

2.3 The WCS Model

The approach taken for the Gemini WCS service maximizes ease of use and efficiency without significantly compromising accuracy. The scheme is based on the "affine transformation" used in conventional plate reduction. This models the pointing in the focal plane by expressing the relationship between a gnomonic (*i.e.* tangent plane) projection of the celestial coordinates in the field of view to the $[x, y]$ coordinates in the focal plane (which can either be in the rotating frame of the instrument mount or can be in coordinates local to the instrument). The model consists of eight numbers: the two spherical coordinates of the centre of the projection (θ_0, ϕ_0) and the six coefficients of the affine transformation (a, b, c, d, e, f).

2.4 Where the WCS Model Comes From

The WCS software obtains the model by sampling the focal plane in five places (the centre and the corners of a square extending about halfway to the edge of the field) and carrying out a rigorous pointing calculation for each. A least-squares fit is then performed to obtain the affine transformation which expresses the predicted standard coordinates $[\xi, \eta]$ as a function of the sample $[x, y]$ coordinates. The six coefficients $a \ldots f$ encapsulate the overall scale and orientation, plus any squash and skew.[2] Higher-order distortions, in particular the fact that refraction makes the bottom of the field more squashed than the top, are not modelled; however, the resulting errors are always acceptable for Gemini, a few milliarcseconds at most.

2.5 Mathematical Details

Normally, application code will use functions from the Gemini `astlib` library to transform positions using a WCS transformation structure. However, on some occasions it may be appropriate to operate with a WCS transformation directly, either using inline code or calls to SLALIB functions. The techniques are as follows.

2.5.1 Focal plane to sky

To transform rotator $[x, y]$ coordinates into celestial coordinates $[\theta, \phi]$, where θ stands for right ascension or azimuth and ϕ stands for declination or elevation, first apply the affine transformation to the $[x, y]$ and generate standard coordinates $[\xi, \eta]$:

$$\xi = a + bx + cy$$

$$\eta = d + ex + fy$$

The $[\xi, \eta]$ can then be de-projected into spherical coordinates $[\theta, \phi]$:

$$d = \cos\phi_\circ - \eta \sin\phi_\circ$$

$$\theta = \tan^{-1}\frac{\xi}{d} + \theta_\circ$$

$$\phi = \tan^{-1}\frac{\sin\phi_\circ + \eta\cos\phi_\circ}{\sqrt{\xi^2 + d^2}}$$

The two stages correspond to the SLALIB routines `slaXy2xy` and `slaDtp2s`.

2.5.2 Sky to focal plane

To transform celestial coordinates $[\theta, \phi]$ into rotator coordinates $[x, y]$, first invert the affine transformation:

$$\Delta = bf - ce$$

$$A = (cd - af)/\Delta$$

[2]Because the origin of $[x, y]$ is the tangent point, two of the coefficients, a and d in the expressions given later, are close to zero (not exactly so because of the changing refraction across the field). Coefficients c and `e` are usually almost the same as each other, and so are b and `f` apart from a sign change. The six coefficients can be decomposed into scales, orientation and non-perpendicularity by means of the SLALIB routine `slaDcmpf`.

$$B = f/\Delta$$

$$C = -c/\Delta$$

$$D = (ae - bd)/\Delta$$

$$E = -e/\Delta$$

$$F = b/\Delta$$

Then transform the spherical coordinates into standard coordinates:

$$\Delta\theta = \theta - \theta_\circ$$

$$d = \sin\phi \sin\phi_\circ + \cos\phi \cos\phi_\circ \cos\Delta\theta$$

$$\xi = \frac{\cos\phi \sin\Delta\theta}{d}$$

$$\eta = \frac{\sin\phi \cos\phi_\circ - \cos\phi \sin\phi_\circ \cos\Delta\theta}{d}$$

Finally transform the standard coordinates into rotator $[\,x, y\,]$:

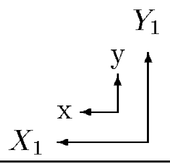
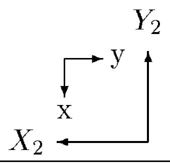
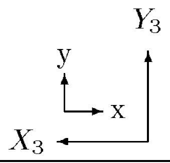
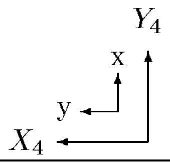
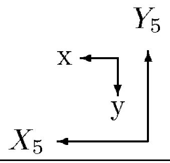
$$x = A + B\xi + C\eta$$

$$y = D + E\xi + F\eta$$

The three stages correspond to the SLALIB routines `slaInvf`, `slaDs2tp` and `slaXy2xy`.

2.6 Nominal Focal-Plane Coordinates at the Instrument Ports

The $[\,x, y\,]$ coordinate system for each of the four instrument ports is such that with the telescope vertical and viewed from the outside the y-axis points up and the x-axis points to the left. In a mechanical sense, this matches that of the Cassegrain port in handedness, but in terms of the region of sky being viewed there is a rotation of the field, different for each port, plus a mirror-reversal. For WCS focal-plane coordinates x and y and instrument-port coordinates X_n and Y_n, where n is the port number:

$X_1 = +x$ $Y_1 = +y$	
$X_2 = -y$ $Y_2 = -x$	
$X_3 = -x$ $Y_3 = +y$	
$X_4 = -y$ $Y_4 = +x$	
$X_5 = +x$ $Y_5 = -y$	

In addition to the gross transformations pictured above, there may be small rotational or zero-point errors arising from optical misalignments.

The WCS facilities described in this paper work always in terms of the Cassegrain port (Port #1), and are therefore independent of which port is in use; the transformation between the coordinate system of the port in question and the standard $[x, y]$ system are the responsibility of the applications which use the TCS. However, because the coordinate systems at these ports are not realized (no encoders or local rotator axes) they bypass the TCS's calculations and cancel out of any calibration procedures.

2.7 Calibration

Careful mechanical measurements of the instrument with respect to the mounting arrangements will enable a preliminary assessment of the WCS to be made: what values of $[x, y]$ to use for the pointing origins, which way up the picture will be, and so on. With the instrument mounted on the telescope, these measurements must, in preparation for the observing run, be refined by observations using stars.

By adjusting the telescope until a star lies on a standard pointing-origin (the centre of the acquisition camera for instance), and then adjusting the instrument pointing origin until the image lies in the desired position on the detector, an $[x, y]$ can be logged which not only will be of the required accuracy but will also cancel out various calibration errors along the way. Second and third such determinations, for a different place on the detector, will allow (with the aid of a routine to be described later) the orientation and handedness of the instrument coordinate system to be determined. Again, any calibration errors will automatically cancel out.

The most important result of these pre-observing calibrations is the ability to place stars on the instrument accurately and to do so for all attitudes of the telescope and rotator. A further important consequence is that the WCS transformations supplied by the TCS will be correct, limited only by the telescope's absolute pointing accuracy. For the highest precision, the pointing can be locally zeroed on a nearby star of known position prior to the WCS transformation being captured.

3 THE TELESCOPE CONTEXT

3.1 The Context Information

In order to predict the mapping between sky coordinates and the rotator $[x, y]$ coordinate system in the focal plane, the WCS software requires the following context information:

- The telescope orientation: specifically, the post-collimation, pre-flexure $[\,Az, El\,]$.
- The telescope parameters: focal length, rotator position-angle, and the coefficients for the geometrical part of the pointing model.
- The parameters involved in transforming from apparent place to observed place: site location, refraction information, sidereal time *etc.*
- M2 scan tip/tilt for the three chop states.
- A timestamp, to mark the time at which the $[\,Az, El\,]$ was or will be correct.

In the case where the WCS facilities are being used in real time to predict what the telescope is actually seeing, all of this information can be directly interrogated from the TCS. In cases where the WCS facilities are being used in an offline application, or online but for another time or field, the context must somehow be simulated. A function is provided to accomplish this, called `timeSimctx`; it requires a number of additional items, concerning the time, the observatory location and the target. The section on "Programming Examples", later, demonstrates what needs to be done.

There are two subtle but important points to make here.

1. Whereas to predict the telescope *pointing*, accurate values for all of the above information must be supplied, the *focal-plane projection* is insensitive to many of the items, provided that the set presented is internally consistent. This is of little relevance to the online/real-time case, where correct values are automatically available. But for the non-real-time case it will make very little difference what values of the pointing coefficients and meteorological parameters (for example) are supplied when generating the simulated context. Even the precise time may not matter, in sidereal tracking cases, where the projection of $[\,\alpha, \delta]$ into the focal plane is, to first order, constant; in such cases there is no point in worrying what value of UT1−UTC to use, for example. When using simulated contexts, the watchwords are consistency and common sense.

2. The WCS facilities use as the low-level telescope position the "post-collimation" or (same thing) "pre-flexure" $[\,Az, El\,]$. The present intention is to implement all the additional pointing terms that will turn out to be needed as a group, applied between the post-collimation level and the MCS demand $[\,Az, El\,]$ level, rather than by intermingling the new terms with the basic 5-term "geometrical" model (AN, AW, NPAE, CA, CE). The MCS demands will then be the "post-collimation" $[\,Az, El\,]$ plus tube flexure and other semi-empirical terms, plus index errors. It is extremely unlikely that these extra terms will significantly affect the orientation of the image, and hence they can safely be neglected for WCS purposes. This has the major benefit that the WCS software will not need to change as the pointing models are developed. The one disadvantage is that applications which wish to predict the positions of star images in the focal plane *starting from raw encoder readings* must themselves calculate and apply the correction for tube flexure and the other additional terms.

3.2 Real-Time Aspects

TCS makes the necessary context information available as an EPICS process variable that is updated at 20 Hz. The EPICS database in the instrument IOC must arrange to copy this information at appropriate times and update the copy in `astlib` global memory. This should be done with a `gensub` record with the following properties set on one of its input links (INPA in this example):

NOA	`39`
FTA	`DOUBLE`
def(INPA)	`TCS:ak:astCtx.VALA`

and with an associated subroutine similar to:

```
long updateAstCtx(struct genSubRecord *pgsub)
{
   (void) astSetctx( pgsub->a );

   return 0;
}
```

(The correct value to use for the **NOA** property is defined in the header file `astLib.h` as macro `AST_CTXA_SIZE`.)

Having copied the telescope context from the TCS to the local IOC in this way, it is necessary to extract the required numbers and turn them into a WCS transformation. The functions which do this are described in the next section.

It should be borne in mind that a new context can only be brought to the IOC through EPICS, and that the functions which the application calls merely access this local copy. This raises the question of how frequently the EPICS transfer should be triggered – maintaining a copy of the context at the full 20 Hz may not be feasible.

As a rule, a Gemini telescope will track the same $[\alpha, \delta]$ for long periods, with the field stabilized using the instrument rotator. Under these circumstances, the WCS transformation (*i.e.* between focal-plane $[x, y]$ and some sort of $[\alpha, \delta]$) will stay approximately constant. Therefore no great demands will be placed on the instrument control system for precise timing, and there will be no need for frequent EPICS activity. Although the transformation will change slowly due to differential refraction, an update to the WCS transformation every few minutes will be good enough for many purposes.

However, where the telescope or the instrument rotator is moving with respect to the nominated coordinate system – trailing, scanning, offsetting *etc.* – the instrumentation system will need to take time into account and will need more frequent EPICS activity. This ever-changing information is rendered manageable by the WCS functions returning a timestamp along with the transformation. The timestamp is the Gemini raw time which corresponds to the other telescope context information which the TCS makes available at 20 Hz and which applies equally precisely to the WCS transformation itself.

4 THE ASTROMETRY SOFTWARE

Developers of Gemini instrumentation applications are provided with two libraries of astrometry-related C functions. One is a set of Gemini-specific routines called `astlib`; the other is the general-purpose positional-astronomy library `slalib`.

4.1 Functions Specific to Gemini

4.1.1 Background

All Gemini subsystems are expected to obtain WCS information through the small library of functions which is described below. Further transformations may then be carried out using `slalib`.

The routines fall into three categories:

1. "Get WCS transformation" – needed as frequently as the transformation changes, perhaps every five minutes for normal applications involving sidereal tracking. WCS transformations are obtained from the TCS in the form of the "context", a set of parameters about the telescope – where it is pointed, the current values of time-dependent positional-astronomy quantities and so on. Using a context obtained from the TCS via EPICS, the subsystem can, without causing network activity or incurring any other interprocessor delays, generate one or more WCS transformations and do multiple WCS transformations. The functions under this heading can be further subdivided into (a) routines only runnable in an IOC and (b) routines that can also run on a Unix host.

2. "Express focal-plane coordinates as celestial coordinates or *vice versa*" – used whenever the system wants to relate the instrument to the sky.

3. "Transform sky coordinates from one reference frame to another" – useful, for example, when an IOC has the B1950 position of a source and wishes to log it also in J2000 coordinates.

Some functions from category 1, and all functions in categories 2 and 3 can be run in a Unix host as well as an IOC. They can also be used offline, in conjunction with stored or artificial WCS transformation structures. Furthermore, the functions in category 3 are quite general and can be used to carry out transformations unconnected with the Gemini telescopes.

Offline use requires certain context information to be supplied which is normally obtained (a good deal more conveniently) directly from the TCS. The context involves time as well as the telescope.

4.1.2 Classified List of Routines

A classified list of the Astrometry Library functions is given below. Detailed specifications for each function (in alphabetical order) are given in the next section.

1. **Get WCS transformation**

   IOC-runnable only:

   - `astGetctx` – obtain the full context information.
   - `astGetpo` – obtain the current pointing origins.
   - `astSetctx` – set the context information.
   - `astGettr` – obtain a WCS transformation.

   Runnable also on a Unix host or offline:

   - `astCtx2tr` – use context to generate a WCS transformation ($[x, y]$ to sky).
   - `astFitij` – fit instrumental coordinates to focal plane coordinates.
   - `astFITS` – express a WCS transformation as FITS headers.
   - `astInvtr` – invert a WCS transformation.
   - `astRot` – predict the rotator orientation.
   - `astSimctx` – generate a simulated context.
   - `astXtndr` – extend a WCS transformation to work from pixel instead of rotator coordinates.

2. **Transform between focal-plane and sky coordinates**

   - `astS2xy` – sky to rotator $[x, y]$.
   - `astS2xyq` – sky to $[x, y]$ using an existing WCS transformation.
   - `astXy2s` – rotator $[x, y]$ to sky.
   - `astXy2sq` – $[x, y]$ to sky using an existing WCS transformation.

3. **Transform between two sorts of sky coordinates**

   - `astCoco` – transform position between celestial frames.
   - `astCocoR` – transform position and rate between celestial frames.

Important Note:

The WCS functions support two forms of transformation: $[x, y]$-to-sky and sky-to-$[x, y]$. These occupy identical data structures; in fact two of the numbers (the spherical coordinates of the rotator axis) are unaffacted as one form is transformed into the other. The inversion operation is most conveniently carried out by the `astInvtr` function, and this works in either direction: two successive calls with the output of the first call fed to the input of the second will reproduce the original transformation.

This symmetry introduces the danger of using the wrong transformation – for example attempting to convert an $[x, y]$ into an $[\alpha, \delta]$ with a WCS structure that is for the sky-to-$[x, y]$ direction. Under these circumstances erroneous numbers will be generated, and no error will be reported. This danger is the price paid for flexibility. Applications may wish to generate their own transformations, perhaps using entirely different units and with the coordinate system reckoned from an entirely different origin. The supplied functions are designed to support this type of use.

In short, it is up to the instrumentation application to keep track of what its various WCS structures are for.

4.1.3 Routine Specifications (alphabetical)

Details of all the `astlib` routines follow.

Notes:

- Applications which use the library must contain an `#include astLib.h` statement. There is an `astsys.h` include file as well, but this is for internal use only. For clarity, the definitions for the various enumerations and structures, extracted from the `#include` files, are reproduced in a later section.
- The coordinate-system argument, `frame`, is an *enumerated type*, defined in `astLib.h`. The indicated codes are used exactly as indicated (no quote marks). For example, to transformation a focal-plane $[x, y]$ into J2000 $[\alpha, \delta]$ (for a 1μ wavelength and chop state A) you could write:

  ```
  j = astXy2s ( x, y, FK5, 2000.0, 1.0, 0, &ra, &dec )
  ```

  Similar remarks apply to the symbol `TT`, which stands for the timescale Terrestrial Time.

---

astCoco Transform Position astCoco

ACTION :

Transform a source position from one coordinate system to another.

CALL :

```
status = astCoco (a1, b1, pmotion,
                  cosys1, equinox1, epoch1,
                  cosys2, equinox2, date, aoprms, tel,
                  &a2, &b2 )
```

GIVEN :

Source

a1	**double**	right ascension or azimuth (radians)
b1	**double**	declination or elevation (radians)
pmotion	**struct PMPXRV**	proper motion *etc.* (see notes 2 and 13)

Starting coordinate system

cosys1	**FRAMETYPE**	original coordinate system
equinox1	**struct EPOCH**	catalogue equinox (see note 13)
epoch1	**struct EPOCH**	catalogue epoch (see note 13)

Final coordinate system

cosys2	**FRAMETYPE**	final coordinate system
equinox2	**struct EPOCH**	equinox of final frame (see note 13)

Circumstances

date	**double**	epoch of observation (TT MJD; see note 1)
aoprms	**double[15]**	apparent-to-observed parameters (see notes 5 and 13)
tel	**struct TELP**	telescope-dependent parameters (see note 13)

RETURNED :

a2	**double***	final right ascension or azimuth (radians)
b2	**double***	final declination or elevation (radians)

RETURNED (function value) :

int	status:	$+1 =$	unreachable position
		$0 =$	OK
		$-1 =$	invalid original frame
		$-2 =$	invalid final frame

NOTES :

1. When using this routine, a potential source of confusion is that there are two sorts of "now":

`date`	epoch of observation
`aoprms[13]`	sidereal time

   In principle, the date and the sidereal time ought to be consistent. However, in practice this is unimportant: the date is used for calculating proper motion, precession *etc.*, while the sidereal time is used for converting between an $[\alpha, \delta]$ and an $[Az, El]$. Note, however, that the sidereal time must be accurate: any vagueness will appear as corresponding uncertainties in the rapidly-changing $[\alpha, \delta]$ to/from $[Az, El]$ transformation.

2. The `pmotion.pm` flag distinguishes between the cases where a proper motion in the (rotating) FK4 frame is being supplied and the case where there is assumed to be no proper motion in a (non-rotating) inertial frame. False (0) means zero proper motion in an inertial frame and is the correct value to use for extragalactic sources. True ($\neq 0$) is the correct choice where there is a known proper motion, *e.g.* from a catalogue. Note that in FK4 coordinates a distant object exhibits a fictitious proper motion of up to about $0''.5$ per century. Conversely, a star that happened to have a zero proper motion in the FK4 system would in fact have a proper motion of up to about $0''.5$ per century with respect to an inertial frame. Setting `pmotion.pm` to false has no effect for non-FK4 frames except that the parallax and radial velocity are assumed zero.

3. The treatment of proper motion, parallax and radial velocity is, strictly speaking, approximate in the case of FK4/FK5 frames of non-standard equinox (*i.e.* not B1950/J2000). The treatment here assumes that the proper motions are constant in spherical coordinates and the radial velocity and parallax are correct at B1950/J2000.

4. Input mean $[\alpha,\delta]$ frames are barycentric (*i.e.* the coordinates are unaffected by annual parallax). Output mean $[\alpha,\delta]$ frames are geocentric (*i.e.* are subject to variations due to annual parallax).

5. The `aoprms` array contains the star-independent apparent-to-observed parameters:

`[0]`	geodetic latitude (radians)
`[1,2]`	sine and cosine of geodetic latitude
`[3]`	magnitude of diurnal aberration vector
`[4]`	height (metres)
`[5]`	ambient temperature (°K)
`[6]`	pressure (mB)
`[7]`	relative humidity (0–1)
`[8]`	effective wavelength (microns)
`[9]`	tropospheric lapse rate (°K/metre)
`[10,11]`	refraction constants A and B (radians)
`[12]`	longitude + eqn of equinoxes + sidereal ΔUT (radians)
`[13]`	local apparent sidereal time (radians)
`[14]`	polar-motion adjustment to azimuth (radians)

   It can be constructed by calling the SLALIB routine `slaAoppa` to generate elements 0-13 and then `slaPolmo` to generate the final element. The sidereal time (element 13) can be selectively updated by calling `slaAoppat`. Adjustment of the refraction constants can be accomplished most efficiently by using `slaAtmdsp` to provide new values for elements 8, 10 and 11.

   Element 12 is not used by the present routine. It is included for compatibility with `slaAoppat`.

6. Where the given position is one of the $[Az, El]$s, it is assumed to refer to the sidereal time supplied in the `aoprms` array. This means that the function cannot convert an $[Az, El]$ from one epoch to another.

7. There is no provision for different telescope parameters applying to the original and final coordinate systems. However, transformations involving different telescope parameters (for example specifying two different pointing- origins) can be carried out

by calling the function twice, first to transform into apparent place (say) and then to transform back again with the new telescope parameters.

8. The transformations between topocentric and mount coordinates assume tangent-plane geometry in the focal plane.
9. Azimuths are with respect to terrestrial rather than celestial north.
10. The "unreachable position" status occurs when predicting mount $[Az, El]$ for a point near the zenith that cannot be reached because of collimation error. A reachable position nearby is returned under these circumstances.
11. For transformations into mount $[Az, El]$, there may be cases near the zenith where there are two solutions. In such cases, the function returns only the solution where the star and the telescope are on the same side of the zenith.
12. Refraction corrections are disabled below the Gemini elevation limit to reduce computation. See the `astCocoR` routine for further details.
13. Not all the arguments are referred to in all cases. The following rules apply:

argument	*referred to if and only if*
`equinox1`	`cosys1` is `FK4` or `FK5`
`epoch1`	`cosys1` is `FK4` or `FK5`
`pmotion`	`cosys1` is `FK4` or `FK5`
`equinox2`	`cosys2` is `FK4` or `FK5`
`epoch2`	`cosys2` is `FK4` or `FK5`
`tel`	`cosys1` and/or `cosys2` is `AZEL_MNT`
`aoprms`	both `cosys1` and `cosys2` are `APPT`, `AZEL_TOPO` or `AZEL_MNT`

astCocoR Transform Position and Drift-Rate astCocoR

ACTION :

Transform a source position and a drift-rate from one coordinate system to another.

CALL :

```
status = astCocoR (a1, b1, pmotion, adot1, bdot1,
                   cosys1, equinox1, epoch1,
                   cosys2, equinox2, date, aoprms, tel,
                   &a2, &b2, &adot2, &bdot2 )
```

GIVEN :

Source

a1	**double**	right ascension or azimuth (radians)
b1	**double**	declination or elevation (radians)
pmotion	**struct PMPXRV**	proper motion *etc.* (see notes 2 and 13)

Drift-rate

adot1	**double**	drift-rate in right ascension or azimuth (radians per SI second)
bdot1	**double**	drift-rate in declination or elevation (radians per SI second)

Starting coordinate system

cosys1	**FRAMETYPE**	original coordinate system
equinox1	**struct EPOCH**	catalogue equinox (see note 13)
epoch1	**struct EPOCH**	catalogue epoch (see note 13)

Final coordinate system

cosys2	**FRAMETYPE**	final coordinate system
equinox2	**struct EPOCH**	equinox of final frame (see note 13)

Circumstances

date	**double**	epoch of observation (TT MJD; see note 1)
aoprms	**double[15]**	apparent-to-observed parameters (see notes 5 and 13
tel	**struct TELP**	telescope-dependent parameters (see note 13)

RETURNED :

a2	**double***	final right ascension or azimuth (radians)
b2	**double***	final declination or elevation (radians)
adot2	**double***	final drift-rate in right ascension or azimuth (radians per SI sec)
bdot2	**double***	final drift-rate in declination or elevation (radians per SI sec)

RETURNED (function value) :

int	status:	$+1 =$	unreachable position
		$0 =$	OK
		$-1 =$	invalid original frame
		$-2 =$	invalid final frame

NOTES :

1. When using this routine, a potential source of confusion is that there are two sorts of "now":

`date`	epoch of observation
`aoprms[13]`	sidereal time

   In principle, the date and the sidereal time ought to be consistent. However, in practice this is unimportant: the date is used for calculating proper motion, precession *etc.*, while the sidereal time is used for converting between an $[\alpha, \delta]$ and an $[Az, El]$. Note, however, that the sidereal time must be accurate: any vagueness will appear as corresponding uncertainties in the rapidly-changing $[\alpha, \delta]$ to/from $[Az, El]$ transformation.

2. The `pmotion.pm` flag distinguishes between the cases where a proper motion in the (rotating) FK4 frame is being supplied and the case where there is assumed to be no proper motion in a (non-rotating) inertial frame. False (0) means zero proper motion in an inertial frame and is the correct value to use for extragalactic sources.

True ($\neq 0$) is the correct choice where there is a known proper motion, *e.g.* from a catalogue. Note that in FK4 coordinates a distant object exhibits a fictitious proper motion of up to about $0\farcs5$ per century. Conversely, a star that happened to have a zero proper motion in the FK4 system would in fact have a proper motion of up to about $0\farcs5$ per century with respect to an inertial frame. Setting `pmotion.pm` to false has no effect for non-FK4 frames except that the parallax and radial velocity are assumed zero.

3. The treatment of proper motion, parallax and radial velocity is, strictly speaking, approximate in the case of FK4/FK5 frames of non-standard equinox (*i.e.* not B1950/J2000). The treatment here assumes that the proper motions are constant in spherical coordinates and the radial velocity and parallax are correct at B1950/J2000.
4. Input mean $[\,\alpha, \delta\,]$ frames are barycentric (*i.e.* the coordinates are unaffected by annual parallax). Output mean $[\,\alpha, \delta\,]$ frames are geocentric (*i.e.* are subject to variations due to annual parallax).
5. The `aoprms` array contains the star-independent apparent-to-observed parameters:

`[0]`	geodetic latitude (radians)
`[1,2]`	sine and cosine of geodetic latitude
`[3]`	magnitude of diurnal aberration vector
`[4]`	height (metres)
`[5]`	ambient temperature (°K)
`[6]`	pressure (mB)
`[7]`	relative humidity (0–1)
`[8]`	effective wavelength (microns)
`[9]`	tropospheric lapse rate (°K/metre)
`[10,11]`	refraction constants A and B (radians)
`[12]`	longitude + eqn of equinoxes + sidereal ΔUT (radians)
`[13]`	local apparent sidereal time (radians)
`[14]`	polar-motion adjustment to azimuth (radians)

It can be constructed by calling the SLALIB routine `slaAoppa` to generate elements 0-13 and then `slaPolmo` to generate the final element. The sidereal time (element 13) can be selectively updated by calling `slaAoppat`. Adjustment of the refraction constants can be accomplished most efficiently by using `slaAtmdsp` to provide new values for elements 8, 10 and 11.

Element 12 is not used by the present routine. It is included for compatibility with `slaAoppat`.

6. Where the given position is one of the $[\,Az, El\,]$s, it is assumed to refer to the sidereal time supplied in the `aoprms` array. This means that the function cannot convert an $[\,Az, El\,]$ from one epoch to another.
7. There is no provision for different telescope parameters applying to the original and final coordinate systems. However, transformations involving different telescope parameters (for example specifying two different pointing- origins) can be carried out by calling the function twice, first to transform into apparent place (say) and then to transform back again with the new telescope parameters.
8. The transformations between topocentric and mount coordinates assume tangent-plane geometry in the focal plane.

9. Azimuths are with respect to terrestrial rather than celestial north.
10. The "unreachable position" status occurs when predicting mount $[Az, El]$ for a point near the zenith that cannot be reached because of collimation error. A reachable position nearby is returned under these circumstances.
11. For transformations into mount $[Az, El]$, there may be cases near the zenith where there are two solutions. In such cases, the function returns only the solution where the star and the telescope are on the same side of the zenith.
12. The routine includes special precautions to avoid expensive and pointless refraction calculations for cases below the Gemini 15° elevation limit. These precautions consist of disabling the refraction corrections when transforming to or from `AZEL_MNT` positions at elevations less than about $14^{\circ}\!.5$.
13. Not all the arguments are referred to in all cases. The following rules apply:

argument	*referred to if and only if*
`equinox1`	`cosys1` is `FK4` or `FK5`
`epoch1`	`cosys1` is `FK4` or `FK5`
`pmotion`	`cosys1` is `FK4` or `FK5`
`equinox2`	`cosys2` is `FK4` or `FK5`
`epoch2`	`cosys2` is `FK4` or `FK5`
`tel`	`cosys1` and/or `cosys2` is `AZEL_MNT`
`aoprms`	both `cosys1` and `cosys2` are `APPT`, `AZEL_TOPO` or `AZEL_MNT`

14. All combinations of target and tracking frames produce drift-rates. These are generated by differencing the prediction for the given date and a few seconds in the future.

astCtx2tr Generate WCS Transformation astCtx2tr

ACTION :

Generate a world coordinate system transformation.

CALL :

```
status = astCtx2tr ( ctx, frame, equinox, wavel, ichop, &wcsp, &time )
```

GIVEN :

ctx	**struct WCS_CTX**	WCS context
frame	**FRAMETYPE**	type of coordinate system
equinox	**struct EPOCH**	equinox (applied only to mean $[\alpha, \delta]$ cases)
wavel	**double**	wavelength (μ)
ichop	**int**	chop state (0=A, 1=B, 2=C)

RETURNED :

wcsp	**struct WCS***	WCS transformation ($[x, y]$ to sky)
time	**double***	Gemini raw time at which the transformation was correct

RETURNED (function value) :

int	status:	$0 =$	OK
		$-1 =$	illegal frame
		$-2 =$	illegal ichop value
		$-3 =$	error from `timeThenD`
		$-4 =$	error from `astCoco`
		$-5 =$	error from `slaDs2tp`
		$-6 =$	error from `slaFitxy`

NOTE :

The frame cannot be mount $[Az, El]$ because it refers to the telescope rather than the sky.

astFitij Fit Instrumental Coordinates to Focal-Plane astFitij

ACTION :

Fit instrumental $[i, j]$ coordinates to focal-plane $[x, y]$ coordinates.

CALL :

```
status = astFitij (n, fpxy, pixij, cij,
                   &pixis, &pixjs, &perp, &orient )
```

GIVEN :

n	**int**	number of sample points (at least 3)
fpxy	**double[n][2]**	sets of $[x, y]$ coordinates
pixij	**double[n][2]**	corresponding $[i, j]$ coordinates

RETURNED :

cij	**double***	affine transformation, $[i, j]$ to $[x, y]$
pixis	**double***	i scale, x units per i unit
pixjs	**double***	j scale, x units per j unit
perp	**double***	i/j non-perpendicularity (radians)
orient	**double***	orientation of $[i, j]$ with respect to $[x, y]$

RETURNED (function value) :

int	status:	$0 =$	OK
		$-1 =$	bug
		$-2 =$	not enough points
		$-3 =$	singular

NOTES :

1. The coefficients `cij` describe the affine transformation between instrumental coordinates $[\,i, j\,]$ (for example pixel coordinates in the case of a CCD) and focal-plane coordinates $[\,x, y\,]$. Writing `cij[0-5]` as a, b, c, d, e, f:

   $$x = a + bi + cj$$

   $$y = d + ei + fj$$

2. The coefficients can be used by the `astXtndtr` function to extend the WCS transformation so that it starts from instrumental $[\,i, j\,]$ instead of focal-plane $[\,x, y\,]$. This transformation then relates instrumental coordinates (*e.g.* pixels) directly to sky coordinates (*e.g.* J2000 $[\,\alpha, \delta\,]$).
3. For $[\,i, j\,]$ in pixels and $[\,x, y\,]$ in mm, the scales `pixis` and `pixjs` are in units of mm per pixel. For a detector with square pixels, these two numbers should be equal.
4. The nonperpendicularity, `perp`, should be zero for the usual case where the instrumental $[\,i, j\,]$ coordinates are orthogonal.
5. The orientation, `orient`, is the angle of the $+j$ axis clockwise of $+y$ under the following circumstances:
   - The i and j axes appear right-handed.
   - The x and y axes appear left-handed.
6. The first two elements of the `cij` array are the pixel $[\,i, j\,]$ of the origin of the $[\,x, y\,]$ coordinate system. The $[\,x, y\,]$ coordinates corresponding to the centre of the pixel array can be found by directly applying the $[\,i, j\,]$ to $[\,x, y\,]$ transformation (see note 1).

---

astFITS — Generate FITS Headers — astFITS

ACTION :

Express a world coordinate system transformation as FITS headers.

CALL :

```
status = astFITS (wcsij, frame, eqx, tt,
                  ctype1, crpix1, crval1, cdelt1, cunit1,
                  ctype2, crpix2, crval2, cdelt2, cunit2,
                  pc001001, pc001002, pc002001, pc002002,
                  radecsys, equinox, mjdobs)
```

GIVEN :

wcsij	**struct WCS**	WCS transformation, pixels to sky
frame	**FRAMETYPE**	type of sky coordinate system
eqx	**struct EPOCH**	equinox (mean $[\,\alpha, \delta]$s only)
tt	**double**	epoch (TT MJD)

RETURNED :

ctype1	**char***	FITS header record: CTYPE1
crpix1	**char***	FITS header record: CRPIX1
crval1	**char***	FITS header record: CRVAL1
cdelt1	**char***	FITS header record: CDELT1
cunit1	**char***	FITS header record: CUNIT1
ctype2	**char***	FITS header record: CTYPE1
crpix2	**char***	FITS header record: CRPIX2
crval2	**char***	FITS header record: CRVAL2
cdelt2	**char***	FITS header record: CDELT2
cunit2	**char***	FITS header record: CUNIT2
pc001001	**char***	FITS header record: PC001001
pc001002	**char***	FITS header record: PC001002
pc002001	**char***	FITS header record: PC002001
pc002002	**char***	FITS header record: PC002002
radecsys	**char***	FITS header record: RADECSYS
equinox	**char***	FITS header record: EQUINOX
mjdobs	**char***	FITS header record: MJD-OBS

RETURNED (function value) :

int	status:	$0 =$	OK
		$-1 =$	illegal

NOTES :

1. The output strings are null-terminated without trailing spaces. However, the recipient arrays must contain room for the maximum 81 characters to allow for any subsequent spacefilling.
2. Unused keywords produce empty strings.
3. Here is an example of the set of 17 strings output by this routine:

```
CTYPE1  = 'RA---TAN'           / TAN projection
CRPIX1  =    984.044158264385 / pixel i-coordinate at rotator axis
CRVAL1  =     36.000410440627 / RA at rotator axis
CDELT1  =     -0.000010086847 / degrees per 1st-axis unit
CUNIT1  = 'deg     '           / radians per degree
CTYPE2  = 'DEC--TAN'           / TAN projection
CRPIX2  =    438.644955473382 / pixel j-coordinate at rotator axis
CRVAL2  =    +44.997610683246 / Dec at rotator axis
CDELT2  =     +0.000010110504 / degrees per 2nd-axis unit
CUNIT2  = 'deg     '           / radians per degree
PC001001=      0.905380195975 / xi rotation/skew matrix element
PC001002=     -0.424601814334 / xj rotation/skew matrix element
PC002001=      0.420605831504 / yi rotation/skew matrix element
PC002002=      0.907243481379 / yj rotation/skew matrix element
RADECSYS= 'FK5     '           / type of RA/Dec
EQUINOX =   2000.000000000000 / epoch of mean equator & equinox
MJD-OBS =  49560.643763703702 / epoch of observation (TT MJD)
```

astGetctx Obtain Context astGetctx

ACTION :

Obtain the context information needed for generating world coordinate system transformations.

CALL :

```
status = astGetctx ( &ctxp)
```

RETURNED :

ctxp	**struct WCS_CTX***	WCS context

RETURNED (function value) :

	int	status: 0 = OK

NOTES :

1. This routine can only be run in an IOC. It builds a WCS context by copying information held in the IOC that has been obtained from the TCS via EPICS at some earlier time.
2. The structure elements `ctxp->tel.pox` and `ctxp->tel.poy` are set to zero, rather than, for example, the current mount pointing origin coordinates. This is because they have no effect on the WCS, though other routines which use the WCS_CTX structure *do* require them.
3. The structure element `aoprms[12]` is not used by the other WCS functions and is set to zero.

astGetpo Obtain Pointing Origins astGetpo

ACTION :

Obtain the current pointing origins.

CALL :

```
status = astGetpo ( pop )
```

RETURNED :

pop	**struct PO***	pointing origins

RETURNED (function value) :

	int	status: 0 = OK

NOTE :

This routine can only be run in an IOC. It builds a PO structure by copying information held in the IOC that has been obtained from the TCS via EPICS at some earlier time.

astGettr Get Current WCS Transformation **astGettr**

ACTION :

Obtain the currently-available world coordinate system transformation.

CALL :

```
status = astGettr ( frame, equinox, wavel, ichop, &wcsp, &time )
```

GIVEN :

frame	**FRAMETYPE**	type of coordinate system
equinox	**struct EPOCH**	equinox (applied only to mean $[\alpha, \delta]$ cases)
wavel	**double**	wavelength (μ)
ichop	**int**	chop state (0=A, 1=B, 2=C)

RETURNED :

wcsp	**struct WCS***	WCS transformation ($[x, y]$ to sky)
time	**double***	Gemini raw time at which the transformation was correct

RETURNED (function value) :

int	status:	0 =	OK
		−1 =	error from `astGetctx`
		−2 =	error from `astCtx2tr`

astInvtr Invert WCS Transformation **astInvtr**

ACTION :

Invert a world coordinate system transformation, from $[x, y]$-to-sky to sky-to-$[x, y]$ or *vice versa.*

CALL :

```
status = astInvtr ( wcs, &iwcs )
```

GIVEN :

wcs	**struct WCS**	a WCS transformation

RETURNED :

iwcs	**struct WCS***	the opposite WCS transformation

RETURNED (function value) :

int	status:	0 =	OK
		−1 =	error from `slaInvf`

astRot Predict Rotator Angle **astRot**

ACTION :

Predict the rotator orientation required to achieve a given field orientation at the pointing origin.

CALL :

```
status = astRot ( cosys, eqx, ra, de, date, aoprms, tel, pay, &rot )
```

GIVEN :

cosys	**FRAMETYPE**	reference frame...
eqx	**struct EPOCH**	...for specifying field orientation
ra	**double**	RA or azimuth in that frame (radians)
de	**double**	Dec or elevation in that frame (radians)
date	**double**	epoch of observation (TT MJD
aoprms	**double[15]**	star-independent apparent-to-observed parameters
tel	**struct TELP**	telescope-dependent parameters
pay	**double**	desired position-angle of $+y$ through pointing-origin

RETURNED :

rot	**double***	required rotator orientation (range $\pm\pi$)

RETURNED (function value) :

int	status:	$0 =$	OK
		$-1 =$	failed to converge
		$-2 =$	invalid `cosys`

NOTES :

1. The desired position-angle, `pay`, is that of the $+y$ direction passing through the pointing-origin. It is zero when $+y$ points north or up in the specified reference frame, and increases anti-clockwise on the sky.
2. When using this routine, a potential source of confusion is that there are two sorts of "now":

`date`	epoch of observation
`aoprms[13]`	sidereal time

In principle, the date and the sidereal time ought to be consistent. However, in practice this is unimportant: the date is used for calculating precession *etc.*, while the sidereal time is used for converting between $[\alpha, \delta]$ and $[Az, El]$. Note, however, that the sidereal time must be accurate if the field is rapidly rotating as the telescope tracks.

3. In most circumstances, one coordinate system will be used for (a) supplying the target coordinates, (b) controlling the telescope tracking and (c) specifying the field orientation. In particular, it will be usual for star positions to be supplied as J2000 $[\alpha, \delta]$, for the telescope to be tracking in J2000 $[\alpha, \delta]$ and for the field to be oriented with respect to north in J2000 $[\alpha, \delta]$. However, there will also be circumstances where two or three different coordinate systems are involved – the star position may be supplied as B1950 $[\alpha, \delta]$, and the rotator angle may be fixed in $[Az, El]$ for example. Where this is so, preliminary calls to the `astCoco` function will be required before the present routine can be used.

4. The `aoprms` array contains the star-independent apparent-to-observed parameters:

`[0]`	geodetic latitude (radians)
`[1,2]`	sine and cosine of geodetic latitude
`[3]`	magnitude of diurnal aberration vector
`[4]`	height (metres)
`[5]`	ambient temperature (°K)
`[6]`	pressure (mB)
`[7]`	relative humidity (0–1)
`[8]`	effective wavelength (microns)
`[9]`	tropospheric lapse rate (°K/metre)
`[10,11]`	refraction constants A and B (radians)
`[12]`	longitude + eqn of equinoxes + sidereal ΔUT (radians)
`[13]`	local apparent sidereal time (radians)
`[14]`	polar-motion adjustment to azimuth (radians)

   It can be constructed by calling the SLALIB routine `slaAoppa` to generate elements 0-13 and then `slaPolmo` to generate the final element. The sidereal time (element 13) can be selectively updated by calling `slaAoppat`. Adjustment of the refraction constants can be accomplished most efficiently by using `slaAtmdsp` to provide new values for elements 8, 10 and 11.

   Element 12 is not used by the present routine. It is included for compatibility with `slaAoppat`.

5. The function always uses the telescope parameters and so a valid structure must be supplied. However, for many purposes nominal values for the various telescope parameters will be perfectly acceptable.

6. The transformations between topocentric and mount coordinates assume tangent-plane geometry in the focal plane.

7. Azimuths are with respect to terrestrial rather than celestial north.

astS2xy Celestial to Focal-Plane astS2xy

ACTION :

Transform celestial coordinates in a specified celestial coordinate system into a focal-plane $[x, y]$ position.

CALL :

```
status = astS2xy ( a, b, frame, equinox, wavel, ichop, &x, &y )
```

GIVEN :

a	**double**	right ascension or azimuth (radians)
b	**double**	declination or elevation (radians)
frame	**FRAMETYPE**	type of coordinate system
equinox	**struct EPOCH**	equinox (applied only to mean $[\alpha, \delta]$ cases)
wavel	**double**	wavelength (μ)
ichop	**int**	chop state (0=A, 1=B, 2=C)

RETURNED :

x	**double***	focal-plane x-coordinate (mm)
y	**double***	focal-plane y-coordinate (mm)

RETURNED (function value) :

int	status:	$0 =$	OK
		$-1 =$	error from `astGettr`
		$-2 =$	error from `astInvtr`
		$-3 =$	error from `astS2xyq`

NOTES :

1. Azimuths are north-through-east.
2. The $[x, y]$ coordinates are in the Cassegrain focal-plane and rotate with the instrument mount.

---

astS2xyq Quick Celestial to Focal-Plane astS2xyq

ACTION :

Transform celestial coordinates into a focal-plane $[x, y]$ position using pre-computed WCS parameters.

CALL :

```
status = astS2xyq ( a, b, iwcs, &x, &y )
```

GIVEN :

a	**double**	right ascension or azimuth (radians)
b	**double**	declination or elevation (radians)
iwcs	**struct WCS**	WCS parameters for the inverse transformation

RETURNED :

x	**double***	focal-plane x-coordinate (mm)
y	**double***	focal-plane y-coordinate (mm)

RETURNED (function value) :

int	status:	$0 =$	OK
		$-1 =$	star too far from axis
		$-2 =$	antistar on tangent plane
		$-3 =$	antistar too far from axis

NOTES :

1. Azimuths are north-through-east.
2. The $[x, y]$ coordinates are in the Cassegrain focal-plane and rotate with the instrument mount.

astSetctx — Set the Context Information — astSetctx

ACTION :

Set the context information needed for generating world coordinate system transformations.

CALL :

```
status = ( ctxa )
```

RETURNED :

ctxa	**double***	array containing the context

RETURNED (function value) :

	int	status: 0 = OK

NOTE :

This routine copies data into the IOC's global memory. It is intended only to be called by the EPICS record that copies data from the TCS.

astSimctx — Simulate Context — astSimctx

ACTION :

Generate a simulated context from which world coordinate system transformations can be obtained.

CALL :

```
status = astSimctx (tai,
                    elong, phi, hm,
                    tdc, pmb, rh, tlr,
                    wl, tel, m2xy,
                    a1, b1, cosys, equinox,
                    &ctxp )
```

GIVEN :

Time and place

tai	**double**	TAI (MJD)
elong	**double**	east longitude (true, radians)
phi	**double**	latitude (true geodetic, radians)
hm	**double**	height above reference spheroid (metres)
tdc	**double**	ambient temperature (°C)
pmb	**double**	pressure (millibar)
rh	**double**	relative humidity (range 0–1)
tlr	**double**	tropospheric lapse rate (°K/metre)

Telescope

wl	**double**	effective wavelength (μm)
tel	**struct TELP**	telescope-dependent parameters
m2xy	**double[3][2]**	M2 tip/tilts for chop A/B/C

Target

a1	**double**	right ascension or azimuth (radians)
b1	**double**	declination or elevation (radians)
cosys	**FRAMETYPE**	coordinate system
equinox	**struct EPOCH**	equinox (mean $[\alpha, \delta]$ only)

RETURNED (argument) :

ctxp	**struct WCS_CTX***	WCS context

RETURNED (function value) :

int	status:	$0 =$	OK
		$-1 =$	error calling `timeThenD`
		$-2 =$	error calling `astCoco`

NOTES :

1. This function can be used online, in an IOC, or offline, in a Unix host.
2. Normally, the context is obtained from the telescope control system and applies to the circumstances prevailing at the time. The present routine allows a context to be generated for a different set of circumstances. This technique has both online and offline uses. Online, it can be used to simulate a future observation, for example so that guide probe coordinates can be predicted. Offline, it has applications in reconstructing past observations and in pre-observing planning.
3. For the present function to work, the time system must be operating. For online use, this means that `timeInit` must have been called. For offline use, `timeOffline` must have been called. An attempt to use the present function with previously having started the time system produces an error status.
4. The structure element `aoprms[12]` is not used by the other WCS functions and is set to zero.
5. When setting up the `tel` telescope-parameter structure, note that the instrument mount position-angle is the mechanical one, *i.e.* with respect to the vertical rather than north. When starting from sky position-angle, use the function `astRot` to determine the rotator angle that will be needed.
6. The target coordinates `[a1,b1]` are for the epoch of observation (*i.e.* `tai`).
7. If the pressure is not known, it can be estimated using the expression:

   ```
   pmb=1013.25*exp(-hm/8149.9415).
   ```

astXtndtr Extend WCS to Instrument Coordinates astXtndtr

ACTION :

Extend a world coordinate system transformation to start from instrumental $[i, j]$ instead of focal-plane $[x, y]$.

CALL :

```
status = astXtndtr (cij, wcsxy, &wcsxy )
```

GIVEN :

cij	**double[6]**	affine transformation, $[i, j]$ to $[x, y]$
wcsxy	**struct WCS**	WCS transformation ($[x, y]$ to sky)

RETURNED :

wcsij	**struct WCS**	extended WCS transformation ($[i, j]$ to sky)

RETURNED (function value) :

	int	status: $0 = $ OK

NOTE :

The coefficients `cij` describe the affine transformation between instrumental coordinates $[i, j]$ (for example pixel coordinates in the case of a CCD) and focal-plane coordinates $[x, y]$. Writing `cij[0-5]` as a, b, c, d, e, f:

$$x = a + bi + cj$$

$$y = d + ei + fj$$

The coefficients can be generated by calling `astFitij`, or alternatively `slaFitxy`.

astXy2s Focal-Plane to Celestial astXy2s

ACTION :

Transform a focal-plane $[x, y]$ position into a specified celestial coordinate system.

CALL :

```
status = astXy2s ( x, y, frame, equinox, wavel, ichop, &a, &b )
```

GIVEN :

x	**double**	focal-plane x-coordinate (mm)
y	**double**	focal-plane y-coordinate (mm)
frame	**FRAMETYPE**	type of coordinate system
equinox	**struct EPOCH**	equinox (applied only to mean $[\alpha, \delta]$ cases)
wavel	**double**	wavelength (μ)
ichop	**int**	chop state (0=A, 1=B, 2=C)

RETURNED :

a	**double***	right ascension or azimuth (radians)
b	**double***	declination or elevation (radians)

RETURNED (function value) :

int	status:	$0 =$	OK
		$-1 =$	error from `astGettr`
		$-2 =$	error from `astXy2sq`

NOTES :

1. The $[x, y]$ coordinates are in the Cassegrain focal-plane and rotate with the instrument mount.
2. Azimuths are north-through-east.

---

astXy2sq Quick Focal-Plane to Celestial astXy2sq

ACTION :

Transform a focal-plane $[x, y]$ position into celestial coordinates using pre-computed WCS parameters.

CALL :

```
status = astXy2sq ( x, y, wcs, &a, &b )
```

GIVEN :

x	**double**	focal-plane x-coordinate (mm)
y	**double**	focal-plane y-coordinate (mm)
wcs	**struct WCS**	WCS parameters ($[x, y]$ to sky)

RETURNED :

a	**double***	right ascension or azimuth (radians)
b	**double***	declination or elevation (radians)

RETURNED (function value) :

int	status: $0 =$ OK (always)

NOTES :

1. The $[x, y]$ coordinates are in the Cassegrain focal-plane and rotate with the instrument mount.
2. Azimuths are north-through-east.

4.1.4 Definitions

The routines just described use several structure and enumeration definitions. For convenience, the current versions of these are listed below. However, note that they are subject to change and that the **#include** files are the definitive source of information about them.

```
------------------------------------------------------------------------

  Defined in astLib.h:
    WCS_CTX   struct     WCS context
    WCS       struct     WCS transformation parameters
    FRAMETYPE enum       coordinate system IDs
    PMPXRV    struct     proper motion, parallax, radial velocity
    TELP      struct     telescope parameters
    EPOCH     struct     B or J epoch
    PO        struct     Pointing origin structure

  struct PMPXRV {
        int    pm;       /* False = proper motion inertially-zero */
        double pmRA;     /* RA proper motion (radians/year) */
        double pmDec;    /* Dec proper motion (radians/year) */
        double px;       /* Parallax (arcsec) */
        double rv;       /* Radial velocity (km/s) */
  };

  struct WCS_CTX {
        double ab0[2];      /* Mount Az/El, pre-flexure */
        struct TELP tel;    /* Telescope parameters */
        double aoprms[15];  /* Apparent-to-observed parameters */
        double m2xy[3][2];  /* M2 tip/tilt (3 chop states) */
        double time;        /* Gemini raw time */
  };

  struct WCS {
        double ab0[2];      /* Celestial coordinates at x=y=0 */
        double coeffs[6];   /* Affine transformation coefficients */
  };

  typedef enum { AZEL_MNT = 0,    /* Mount Az/El, pre-flexure */
                 AZEL_TOPO = 1,   /* Topocentric Az/El */
                 APPT = 2,        /* Geocentric apparent RA/Dec */
                 FK5 = 3,         /* IAU 1976 RA/Dec, any equinox */
                 FK4 = 4          /* Pre IAU 1976 RA/Dec, any equinox */
               } FRAMETYPE;

  struct EPOCH {
        double year;     /* Epoch:  Byear (B), TT Jyear (J) */
        char   type;     /* Type of epoch ('B', 'J' or ' ') */
  };

  struct TELP {
        double fl;       /* Focal length (metres) */
```

```
      double rpa;      /* Rotator orientation (radians) */
      double an;       /* Azimuth axis tilt NS (radians) */
      double aw;       /* Azimuth axis tilt EW (radians) */
      double pnpae;    /* Az/El nonperpendicularity (radians) */
      double ca;       /* LR collimation (metres) */
      double ce;       /* UD collimation (metres) */
      double pox;      /* Pointing-origin x-component (mm) */
      double poy;      /* Pointing-origin y-component (mm) */
 };

 Defined in timelib.h:
  typedef enum {
              TAI,      /* International Atomic Time */
              UTC,      /* Coordinated Universal Time */
              UT1,      /* Universal Time */
              TT,       /* Terrestrial Time */
              TDB,      /* Barycentric Dynamical Time */
              GMST,     /* Greenwich Mean Sidereal Time */
              LAST      /* Local Apparent Sidereal Time */
             } timescale;
------------------------------------------------------------------------
```

4.2 SLALIB Functions

The SLALIB library contains essentially all the transformations required by the Gemini systems in addition to a large number of other useful functions. The library was used to implement the above Gemini astrometry functions and is also directly usable by the Gemini subsystems. The list below includes only a small selection of SLALIB functions, ones potentially useful in conjunction with the Gemini WCS facilities. Omitted from the list are functions dealing with transformation between celestial reference-frames; such transformations should be handled with `astCoco` rather than with direct calls to SLALIB functions (except for ecliptic or galactic coordinates, which `astCoco` doesn't support).

Many of the SLALIB routines exist in both `float` and `double` forms; only the `double` forms appear in the list below. In addition, many of the routines exist in forms which use $[\,x, y, z\,]$ rather than spherical coordinates; the list concentrates on the latter.

String Decoding:

- `slaIntin` – convert free-format string into `long` integer.
- `slaDfltin` – convert free-format string into `double` real.
- `slaDafin` – convert free-format string from deg,arcmin,arcsec *etc.* to radians (`double`).

Sexagesimal Conversions:

- `slaDtf2r` – hours, minutes, seconds to radians.
- `slaDr2tf` – radians to hours, minutes, seconds.
- `slaDaf2r` – degrees, arcminutes, arcseconds to radians.

- `slaDr2af` – radians to degrees, arcminutes, arcseconds.

Angles:

- `slaDrange` – Normalize angle into range $\pm\pi$.
- `slaDranrm` – Normalize angle into range $0-2\pi$.
- `slaDsep` – Distance between two stars.
- `slaDbear` – Position angle of one star with respect to another.

Astrometry:

- `slaDs2tp` – Spherical coordinates to tangent plane.
- `slaDtp2s` – Tangent plane coordinates to spherical.
- `slaDtps2c` – Field centre from star $[\alpha,\delta]$ and tangent plane coordinates.
- `slaPcd` – Apply pincushion/barrel distortion.
- `slaUnpcd` – Remove pincushion/barrel distortion.
- `slaFitxy` – Fit a linear model to relate two sets of $[x,y]$ coordinates.
- `slaPxy` – Compute predicted coordinates and residuals.
- `slaInvf` – Invert a linear model.
- `slaXy2xy` – Transform one $[x,y]$.
- `slaDcmpf` – Decompose a linear fit into scales *etc.*

5 PROGRAMMING EXAMPLES

5.1 Predicting the Focal-Plane Coordinates of a Star Image

```
#include <timeLib.h>
#include <slalib.h>
#include <astLib.h>
/*
** - - - - - - - - - - - -
**  W C S _ d e m o _ 1
** - - - - - - - - - - - -
**
** Demonstration of offline use of the Gemini World Coordinate
** System facilities.  A context is simulated, for a given time,
** target and pointing origin, and then the [x,y] coordinates
** corresponding to a specified [RA,Dec] are obtained.
**
** The ideal answer is:
```

```
**
**    [x,y] =  +179.000000  -152.000000 (mm)
**
** A typical output from this program is:
**
**    [x,y] =  +178.999751  -152.000004
**
** P.T.Wallace   21 March 1997
**
** Copyright 1997 RAL.  All rights reserved.
*/
#define AS2R 4.84813681109536e-6
#define D2R 0.0174532925199433
int main ( )
{
/* For this test, asssume M2 is at neutral tip/tilt in all chop states */
   double m2xy[3][2] = { { 0.0, 0.0},
                         { 0.0, 0.0},
                         { 0.0, 0.0}
                       };

   struct TELP tel;
   struct WCS_CTX ctx;
   double track_ra, track_dec, track_wl,
          test_ra, test_dec, test_wl;
   FRAMETYPE track_frame, test_frame;
   struct EPOCH track_equinox, test_equinox;
   struct WCS wcs, iwcs;
   double tai, elong, phi, hm, dleap, dat, dut, tdc, pmb, rh, tlr,
          timestamp, x, y;
   int j;


/*
** =============
** FIXED CONTEXT
** =============
**
** In online applications the information set out below would come from
** the TCS automatically as part of the "get WCS context" call.  For
** offline applications like the present one, default values will
** normally be adequate.  Only if the telescope is grossly displaced
** from its simulated orientation will inaccuracies in the final result
** start to appear, caused by the consequent imperfect prediction of the
** field distortions due to refraction.  However, the rotator position
** angle must be correct for the given time, and obviously the target
** coordinates, pointing origin and focal length have to be correct.
**
** The values used here come from the TCS/PTW/4 test case.
*/

/* Time service. */
   dleap = 50083.0;
   dat = 29.0;
   dut = 0.746;

/* Site location and meteorological readings. */
   elong = -2.71349330022745;
   phi = 0.346040321258437;
   hm = 4145.0;
   tdc = 1.85;
   pmb = 605.0;
   rh = 0.8;
   tlr = 0.0065;

/* Telescope. */
   tel.fl = 128000.0;
   tel.rpa = 30.0 * D2R;
   tel.an = -12.0 * AS2R;
   tel.aw = -5.0 * AS2R;
   tel.pnpae = 8.0 * AS2R;
   tel.ca = -110.0 * AS2R;
   tel.ce = 22.0 * AS2R;

/*
** ====
** TIME
** ====
*/

/* Epoch for the simulation. */
   tai = 49560.643391203703;

/* Initialize the time system for offline use. */
   if ( j = timeOffline ( tai, elong, phi, hm, dleap, dat, dut ) ) {
      printf ( "bad status from timeOffline: %d\n", j );
      return -1;
   }

/*
** ============================
** GENERATE A WCS TRANSFORMATION
** ============================
*/

/* Pointing origin, tracking frame and target. */
   tel.pox = 179.0;
   tel.poy = -152.0;
   track_frame = FK4;
   track_equinox.type = 'B';
   track_equinox.year = 1975.0;
   track_ra = 0.365544667419779;
   track_dec = 0.331903446087589;
   track_wl = 0.5;

/* Create a WCS context for the above pointing state. */
```

```
   if ( j = astSimctx ( tai,
                        elong, phi, hm, tdc, pmb, rh, tlr, track_wl,
                        tel, m2xy,
                        track_ra, track_dec, track_frame, track_equinox,
                        &ctx ) ) {
      printf ( "bad status from astSimctx: %d\n", j );
      return -1;
   }

/*
** NOTE:  in the online case, all of the above code is
**        replaced by a call to the astGetctx function.
*/

/* Translate the context into a focal-plane-to-sky WCS transformation. */
   test_frame = FK5;
   test_equinox.type = 'J';
   test_equinox.year = 2000.0;
   test_wl = 0.5;
   if ( j = astCtx2tr ( ctx, test_frame, test_equinox,
                        test_wl, 0, &wcs, &timestamp ) ) {
      printf ( "bad status from astCtx2tr: %d\n", j );
      return -1;
   }

/*
** ======================================
** PREDICT THE [X,Y] FOR A GIVEN [RA,DEC]
** ======================================
*/

/* Generate the sky-to-focal-plane transformation by inverting the */
/* focal-plane-to-sky transformation that we have just created.    */
   if ( j = astInvtr ( wcs, &iwcs ) ) {
      printf ( "bad status from astInvtr: %d\n", j );
      return -1;
   }

/* Apply the transformation to turn a J2000 [RA,Dec] into an [x,y]. */
   test_ra = 0.371441549993331;
   test_dec = 0.334169069920611;
   if ( j = astS2xyq ( test_ra, test_dec, iwcs, &x, &y ) ) {
      printf ( "bad status from astS2xyq: %d\n", j );
      return -1;
   } else {
      printf ( "[x,y] = %+12.6f %+12.6f\n", x, y );
   }
   return 0;

}
```

5.2 Generating FITS Headers for a Sky Imager

```
#include <timeLib.h>
#include <slalib.h>
#include "astLib.h"
/*
** - - - - - - - - - - - -
**  W C S _ d e m o _ 2
** - - - - - - - - - - - -
**
** Demonstration of offline use of the Gemini World Coordinate System
** facilities.  We consider the case of using FITS headers to describe
** observations made with a CCD sky imager.
**
** We have at our disposal the following information:
**
**   . For three sample locations on the detector, the pixel coordinates
**     and the instrument mount coordinates.  (For the purposes of
**     planning an observation, the locations might be three corners
**     of the detector, and the coordinates obtained from a knowledge of
**     the geometry of the detector and the nominal mounting arrangements.
**     For online use, the three locations might be observed star centroids
**     obtained from a start-of-night online calibration procedure.)
**
**   . The approximate time of observation.
**
**   . The J2000 RA/Dec of the centre of the detector.
**
**   . Assumed telescope parameters (not critical for this application,
**     in fact).
**
** We wish to calculate:
**
**   . The instrument mount position angle required to align the
**     detector's y-axis to J2000 north through the given [RA,Dec].
**
**   . The FITS WCS headers for the given position-angle, [RA,Dec] etc.
**
** Typical output:
**
** i/j scales = 22.496 / 22.604 micron
** i/j non-perp =  -0.058 deg
** orientation = +170.006 deg
** CCD centre at  +2.008 /  -4.985
** rotator angle =  +139.209555 deg
** RA/Dec of BLC =   36.0158078524 /  +44.9935268211 deg
** RA/Dec of TRC =   35.9841884447 /  +45.0064708578 deg
** Centre to TR =  28.857862 mm
** CTYPE1  = 'RA---TAN'          / TAN projection
** CRPIX1  =     984.048322324526 / pixel i-coordinate at rotator axis
** CRVAL1  =      36.001799989450 / RA at rotator axis
** CDELT1  =      -0.000010074244 / degrees per 1st-axis unit
** CUNIT1  = 'deg      '          / radians per degree
** CTYPE2  = 'DEC--TAN'           / TAN projection
** CRPIX2  =     438.647166460703 / pixel j-coordinate at rotator axis
** CRVAL2  =     +44.997957315555 / Dec at rotator axis
** CDELT2  =      +0.000010123061 / degrees per 2nd-axis unit
** CUNIT2  = 'deg      '          / radians per degree
** PC001001=       0.999999860620 / xi rotation/skew matrix element
** PC001002=      -0.000527976691 / xj rotation/skew matrix element
** PC002001=      -0.000577633366 / yi rotation/skew matrix element
** PC002002=       0.999999833170 / yj rotation/skew matrix element
** RADECSYS= 'FK5     '           / type of RA/Dec
** EQUINOX =    2000.000000000000 / epoch of mean equator & equinox
** MJD-OBS =   49560.643763703702 / epoch of observation (TT MJD)
**
** P.T.Wallace   25 April 1997
**
** Copyright 1997 RAL.  All rights reserved.
*/

#define AS2R 4.84813681109536e-6
#define D2R 0.0174532925199433
#define PI 3.141592653589793238462643

/* CCD dimensions (pixels) */
#define IPIX 2220.0
#define JPIX 1280.0

int main ( )
{
/* For this test, asssume M2 is at neutral tip/tilt in all chop states */
   double m2xy[3][2] = { { 0.0, 0.0},
                         { 0.0, 0.0},
                         { 0.0, 0.0}
                       };

/* Sample points: focal-plane x/y, pixel i/j */
   double fpxy[4][2], pixij[4][2];

/* Transformation coefficients, i/j to x/y */
   double cij[6];

/* WCS transformations, focal-plane and pixel coordinates to sky */
   struct WCS wcs, wcsij;

/* Miscellaneous */
   double pixis, pixjs, perp, orient, pa;
   struct TELP tel;
   struct WCS_CTX ctx;
   double track_ra, track_dec, track_wl;
   FRAMETYPE track_frame;
   struct EPOCH track_equinox;
   double tai, elong, phi, hm, dleap, dat, dut, tdc, pmb, rh, tlr,
          timestamp, x, y, tt, ut, aoprms[15], r, d, xi, eta;
   int j;
   char ctype1[81], crpix1[81], crval1[81], cdelt1[81], cunit1[81],
        ctype2[81], crpix2[81], crval2[81], cdelt2[81], cunit2[81],
        pc001001[81], pc001002[81], pc002001[81], pc002002[81],
        radecsys[81], equinox[81], mjdobs[81];


/*
** ===============
** GENERAL CONTEXT
** ===============
**
** In online applications the information set out below would come from
** the TCS.  For offline applications like the present one, default values
** will normally be adequate.  Only if the telescope is grossly displaced
** from its simulated orientation will inaccuracies in the final result
** start to appear, caused by the consequent imperfect prediction of the
** field distortions due to refraction.  However, the rotator position
** angle must be correct for the given time, and obviously the target
** coordinates, pointing origin and focal length have to be correct.
*/

/* Time service. */
   dleap = 50083.0;
   dat = 29.0;
   dut = 0.746;

/* Site location and meteorological readings. */
   elong = -2.71349330022745;
   phi = 0.346040321258437;
   hm = 4145.0;
   tdc = 1.85;
   pmb = 605.0;
   rh = 0.8;
   tlr = 0.0065;

/* Telescope. */
   tel.fl = 128000.0;
   tel.rpa = 30.0 * D2R;
   tel.an = -12.0 * AS2R;
   tel.aw = -5.0 * AS2R;
   tel.pnpae = 8.0 * AS2R;
   tel.ca = -110.0 * AS2R;
   tel.ce = 22.0 * AS2R;

/* Tracking frame and target information. */
   track_frame = FK5;
   track_equinox.type = 'J';
   track_equinox.year = 2000.0;
   track_ra = PI / 5.0;
   track_dec = PI / 4.0;

/*
** =====================
** SIMULATE TIME SERVICE
** =====================
*/

/* Epoch for the simulation. */
   tai = 49560.643391203703;

/* Initialize the time system for offline use. */
   if ( j = timeOffline ( tai, elong, phi, hm, dleap, dat, dut ) ) {
      printf ( "bad status from timeOffline: %d\n", j );
      return -1;
   }

/* TAI to raw time. */
   timestamp = timeTai2raw ( tai );

/* Raw time to TT and UT. */
   if ( j = timeThenD ( timestamp, TT, &tt ) ) {
      printf ( "bad status from timeThend: %d\n", j );
      return -1;
   }
   if ( j = timeThenD ( timestamp, UT1, &ut ) ) {
      printf ( "bad status from timeThend: %d\n", j );
      return -1;
   }

/*
** ============================================
** DETERMINE PIXEL TO FOCAL PLANE TRANSFORMATION
** ============================================
*/

/* Pixel i/j and focal-plane x/y for four sample points (with */
/* noise, as if measured during start-of-night calibrations). */
   pixij[0][0] = 51.3;
   pixij[0][1] = 49.5;
   fpxy[0][0] = -19.15;
   fpxy[0][1] = 12.31;

   pixij[1][0] = 50.7;
   pixij[1][1] = 1227.8;
   fpxy[1][0] = -23.77;
   fpxy[1][1] = -13.95;

   pixij[2][0] = 2179.6;
   pixij[2][1] = 1230.4;
   fpxy[2][0] = 23.36;
   fpxy[2][1] = -22.26;

   pixij[3][0] = 2182.3;
   pixij[3][1] = 53.1;
   fpxy[3][0] = 28.08;
   fpxy[3][1] = 3.90;
```

```
/* Perform the i/j to x/y fit. */
   if ( j = astFitij ( 4, fpxy, pixij, cij,
                       &pixis, &pixjs, &perp, &orient ) ) {
      printf ( "bad status from astFitijxy: %d\n", j );
      return -1;
   }
   printf ("i/j scales = %6.3f / %6.3f micron\n",
                             fabs ( pixis ) * 1e3, fabs ( pixjs ) * 1e3 );
   printf ("i/j non-perp = %+7.3f deg\n", perp /D2R );
   printf ("orientation = %+8.3f deg\n", orient / D2R );

/* Set pointing origin to centre of CCD. */
   slaXy2xy ( ( IPIX + 1.0 ) / 2.0, ( JPIX + 1.0 ) / 2.0, cij,
                                                  &tel.pox, &tel.poy );
   printf ( "CCD centre at %+7.3f / %+7.3f\n", tel.pox, tel.poy );

/*
** =======================
** DETERMINE ROTATOR ANGLE
** =======================
*/

/* Generate the current apparent-to-observed parameters. */
   slaAoppa ( ut, 0.0, elong, phi, hm, 0.0, 0.0,
              tdc + 273.15, pmb, rh, track_wl, tlr, aoprms );
   aoprms[14] = 0.0;

/* Position-angle of +y when +j points north.  (The angle  */
/* "orient" was obtained earlier, from the i/j to x/y fit. */
/* It is the orientation of the detector i/j coordinate    */
/* system with respect to the instrument-mount x/y         */
/* coordinate system.)                                     */
   pa = orient + PI;

/* Rotator orientation for that condition at the pointing origin. */
   if ( j = astRot ( track_frame, track_equinox, track_ra, track_dec,
                     tt, aoprms, tel, pa, &tel.rpa ) ) {
      printf ( "bad status from astRot: %d\n", j );
      return -1;
   }
   printf ( "rotator angle = %+12.6f deg\n", tel.rpa / D2R );

/*
** ========================================
** GENERATE PIXEL-TO-SKY WCS TRANSFORMATION
** ========================================
*/

/* Simulate a WCS context.  The telescope information was given  */
/* earlier, and the rotator angle has just been changed to that  */
/* required to achive the specified detector orientation.        */
   if ( j = astSimctx ( tai,
                       elong, phi, hm, tdc, pmb, rh, tlr, track_wl,
                       tel, m2xy,
                       track_ra, track_dec, track_frame, track_equinox,
                       &ctx ) ) {
      printf ( "bad status from astSimctx: %d\n", j );
      return -1;
   }

/* Translate the context into a focal-plane-to-sky WCS transformation. */
   if ( j = astCtx2tr ( ctx, track_frame, track_equinox,
                       track_wl, 0, &wcs, &timestamp ) ) {
      printf ( "bad status from astCtx2tr: %d\n", j );
      return -1;
   }

/* Combine the i/j-to-x/y and x/y-to-sky models. */
   if ( j = astXtndtr ( cij, wcs, &wcsij ) ) {
      printf ( "bad status from astXtndtr: %d\n", j );
      return -1;
   }

/* Transform the BL and TR corner pixel coordinates to RA/Dec. */
   if ( j = astXy2sq ( 0.5, 0.5, wcsij, &r, &d ) ) {
      printf ( "bad status from astXy2sq: %d\n", j );
      return -1;
   }
   printf ( "RA/Dec of BLC = %15.10f / %+15.10f deg\n", r / D2R, d / D2R );
   if ( j = astXy2sq ( IPIX + 0.5, JPIX + 0.5, wcsij, &r, &d ) ) {
      printf ( "bad status from astXy2sq: %d\n", j );
      return -1;
   }
   printf ( "RA/Dec of TRC = %15.10f / %+15.10f deg\n", r / D2R, d / D2R );

/* Measure semi-diagonal in mm. */
   slaDs2tp ( r, d, track_ra, track_dec, &xi, &eta, &j );
   if ( j ) {
      printf ( "bad status from slaDs2tp: %d\n", j );
      return -1;
   }
   printf ( "Centre to TR = %10.6f mm\n",
            sqrt ( xi * xi + eta * eta ) * tel.fl );

/*
** ========================
** GENERATE THE FITS HEADERS
** ========================
*/

   if ( j = astFITS ( wcsij, track_frame, track_equinox, tt,
                     ctype1, crpix1, crval1, cdelt1, cunit1,
                     ctype2, crpix2, crval2, cdelt2, cunit2,
                     pc001001, pc001002, pc002001, pc002002,
                     radecsys, equinox, mjdobs ) ) {
```

```
      printf ( "bad status from astFITS: %d\n", j );
      return -1;
   }
   printf ( "%s\n", ctype1 );
   printf ( "%s\n", crpix1 );
   printf ( "%s\n", crval1 );
   printf ( "%s\n", cdelt1 );
   printf ( "%s\n", cunit1 );
   printf ( "%s\n", ctype2 );
   printf ( "%s\n", crpix2 );
   printf ( "%s\n", crval2 );
   printf ( "%s\n", cdelt2 );
   printf ( "%s\n", cunit2 );
   printf ( "%s\n", pc001001 );
   printf ( "%s\n", pc001002 );
   printf ( "%s\n", pc002001 );
   printf ( "%s\n", pc002002 );
   printf ( "%s\n", radecsys );
   printf ( "%s\n", equinox );
   printf ( "%s\n", mjdobs );

   return 0;
}
```