SPK

a SOFA-based telescope pointing kernel

Patrick Wallace

RAL Space, UK

Software version 1
Document revision 1.0

2022 March 10

Contents

1 Preliminaries

1.1 Quick Start

Compile the demonstrator application `altaz.c`, linked to the SOFA C library[1]. All the SPK functions are explicitly included, so there is no need for `make` or any SPK library.

Execute the program. The report shows a sequence of tracking position+velocity demands for an altazimuth mount (azimuth, elevation and rotator angle), followed by a series of transformations where any two of (i) sky $[\,\alpha, \delta\,]$, (ii) mount $[\,Az, El\,]$ and (iii) focal-plane $[\,x, y\,]$ are transformed into the third.

1.2 What is SPK?

SPK (simple pointing kernel) is a set of ANSI C functions that provides low-level support for the pointing and tracking components of a telescope control system (TCS). The positional-astronomy aspects are handled using tools from the IAU SOFA (Standards of Fundamental Astronomy) collection, while the algorithms that predict the telescope mount angles are all from Wallace (2002).[2]

Existing pointing kernels offer a spectrum of trade-offs between feature-richness, efficiency and accuracy. SPK in comparison offers only basic facilities, with flexibility the primary goal. Computational efficiency is secondary though more than adequate on modern processors. No accuracy compromises are made.

SPK is designed to get the developer off the ground quickly. There are only two key functions to learn about, supplemented by ones to populate the various data structures. The low-level nature of the API naturally means that compared with other pointing kernel products there are many limitions, for example:

- SPK supports only three celestial coordinate systems, namely ICRS $[\,\alpha, \delta\,]$, geocentric apparent $[\,\alpha, \delta\,]$ and observed $[\,Az, El\,]$;
- Only positions are addressed, not velocities.
- It is limited to self-contained telescope optics, hence no provision for Nasmyth or coudé foci.
- Though a pointing model is central to SPK (see 3.3.5), integration with pointing analysis tools (TPOINT, MaxPoint, PointXP *etc.*) is left to the TCS developer.

In any TCS based on SPK, these and other required capabilities would be implemented as a higher-level API, with SPK providing the low-level support. Indeed, this minimalist approach may appeal to many developers, who would sooner devise their own tailored ways of (for example) tracking moving targets or managing multiple focal-plane hotspots than be forced to lock horns with bloated pre-existing mechanisms.

[1] See `https://www.iausofa.org`.

[2] Wallace, P.T., 2002, *A rigorous algorithm for telescope pointing*, Advanced Telescope and Instrumentation Control Software II, edited by Lewis, Hilton, proceedings of the SPIE, 4848, 125. We will refer to this paper as "W02".

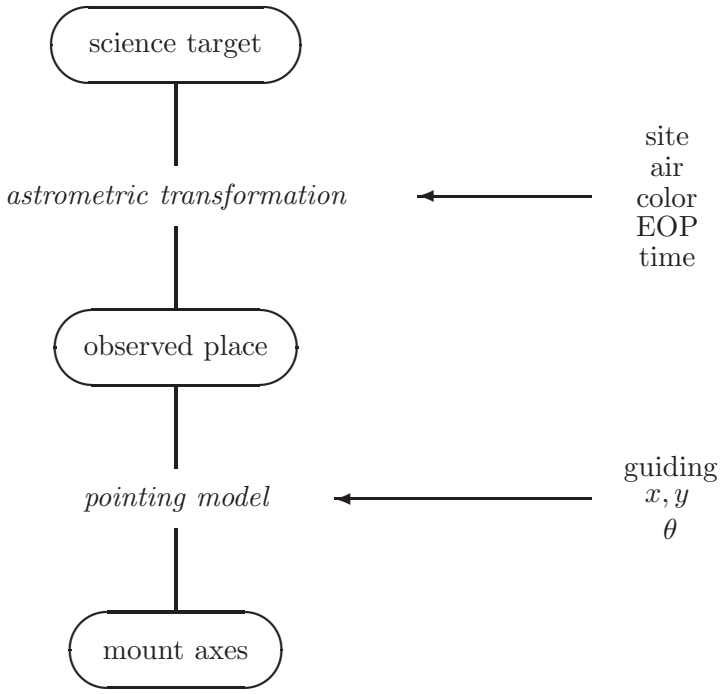

Figure 1: **Telescope pointing**

2 Pointing and tracking: general principles

Figure 1 shows how a telescope is pointed at an astronomical target. The problem breaks down into two stages, namely (i) predicting from what direction in the observer's local sky the radiation appears to be coming, followed by (ii) aiming the mount so that the telescope forms an image of the target on a nominated spot in the focal plane.

Because the astrometric transformation in the first stage is time dependent, primarily as a result of Earth rotation, in order to stabilize the target image the mount axis angles must change continuously; this process is called "tracking".

The first stage, namely predicting from what direction the incoming radiation is coming, can be accomplished efficiently and accurately by calling SOFA functions. The second stage, choosing mount axis angles to achieve the desired imaging, is a problem not addressed directly by SOFA; the method that will be described here implements the rigorous formulas set out in paper W02, with SOFA in merely a supporting role.

The present document describes a set of twelve ANSI C functions that implement Figure 1 and together comprise the rudiments of a "pointing kernel". The essence of SPK comprises only two of the functions; all the rest merely add convenience. Two demonstration `main()` programs, for the equatorial and altazimuth cases respectively, are provided, to illustrate how the functions might be used in a TCS.

Note that the TCS design being discussed here assumes that at some fairly rapid rate, say 20-50 Hz, time-stamped demand positions are sent to the mount drives, where servo mechanisms equipped with digital encoders apply the torques needed to achieve the specified locus. Although this is how modern large telescopes work, it is not what amateur mounts do in most cases – typically such a mount deals with the Earth rotation part of Figure 1 itself, with tracking commanded purely in terms of rates, not positions. In these cases the TCS design would be different from the demonstration programs presented later, although the underlying principles will still apply.

3 SPK design

3.1 Components

The SPK software comprises a set of data structures and the following two key functions:

- `spkAstrom` computes the astrometric context, star-independent quantities such as precession-nutation matrices, for a specified UTC. This function can be called to perform either a (computationally expensive) full refresh or alternatively a first-order update that deals only with Earth rotation.
- `spkVtel` implements the algorithms of paper W02 to transform between celestial directions, focal-plane coordinates and mechanical angles.

. . . In addition:

- The `spkCtar` function accepts an ICRS star catalog entry (epoch J2000.0) and transforms it into astrometric $[\,\alpha, \delta\,]$, taking into account space motion and parallax.[3]
- Nine functions with names `spkI...` initialize various data structures. Their use ensures that all data members of the structures concerned are populated, and some rudimentary validation is provided.

All SPK functions are re-entrant and thread safe.

Two demonstration `main()` programs are provided (Section 4). Called `equat.c` and `altaz.c`, they are essentially the same thing but configured respectively for an equatorial and an altazimuth mount.

[3] Although such an $[\,\alpha, \delta\,]$ is slowly changing, it is in practice always acceptable to regard it as fixed for the duration of an observation.

3.2 Pathways

At first glance, Figure 1 is simply about pointing a telescope: we know the celestial position, typically an $[\alpha, \delta]$, of a science target, and we want to predict the mount axis angles needed to acquire and track its image. However, this is only the first of three pathways through the diagram:

1. Given the target's $[\alpha, \delta]$ and the desired image $[x, y]$, mount axis angles can be predicted.
2. Given the mount axis angles, the $[\alpha, \delta]$ for a given $[x, y]$ in the focal plane can be calculated.
3. Given the mount axis angles, the $[x, y]$ can be predicted at which the image of a given $[\alpha, \delta]$ will appear.

The algorithms in paper W02, and the function `spkVtel`, perform each of these three transformations rigorously, so that repeated paths back and forth through Figure 1 may accumulate rounding errors but nothing worse. W02 calls this scheme the "virtual telescope", acknowledging the degree of information hiding that is going on: the TCS developer is insulated from the subtleties and complications of the astrometric transformation and the various misalignments and flexures in the pointing model, and can concentrate on the science objectives.

The obvious application for pathway (1) is, as we already know, pointing and tracking; pathway (2) provides a "world coordinate system", defining how the sky maps onto the focal plane; and pathway (3) might be used to predict where an autoguider expects to see the image of a guide star. Another use for pathway (2) is to project $[x, y]$ samples onto the celestial coordinate system so that picture orientation can be deduced, and in fact `spkVtel` provides this important capability itself rather than leaving it to the developer.

3.3 Data structures

SPK contains no overall "context". Instead, the application developer sets out the circumstances for the required calculation by choosing an appropriate set of `spkVtel` arguments, each argument a small data structure. There is complete freedom to assemble whatever selection of these describes the problem to be solved, and updating any one instance will affect only those calls that happen to use it. Different sets of structures could define totally different circumstances, all independently computable without any cross-contamination. For example it is perfectly possible to be dealing with two different targets, one for telescope tracking and the other looking ahead to a forthcoming observation. In other words, every `spkVtel` call is a separate instance of a Virtual Telescope.

There are ten of these data structures, containing a total of 40 data members. The particular choice of granularity keeps argument lists reasonably short without significantly reducing versatility.

Even though initialization functions are provided to populate the various structures, the structures' data members are all *public* and may be named explicitly to set or inquire values. There is nothing to stop the developer initializing all data members explicitly, and indeed a sophisticated and reliable pointing kernel could be written that uses only the two key functions `spkAstrom` and `spkVtel`. The initialization functions are mainly a useful checklist for ensuring that no data members are overlooked, in a few cases also providing some elementary validation.

Descriptions of each of the structures follow.

3.3.1 Earth orientation parameters

The `spkEOP` structure contains three `doubles`:

- `xp`, `yp`: the polar motion $[\,x, y\,]$ (radians)
- `dut1`: UT1−UTC (seconds)

Values can be obtained from IERS Bulletins. A simple TCS with no pretensions to high accuracy would probably set all of these values (the polar motion at least) to zero.

The function `spkIeop` can be used to populate such structures.

3.3.2 Observatory

The `spkOBS` structure contains:

- `slon`: site east longitude (ITRF, radians)
- `slat`: site geodetic latitude (ITRF, radians)
- `sh`: site altitude (above WGS84 ellipsoid, meters)
- `mount`: mount type, EQUAT or ALTAZ (see below)
- `pavoid`: closest to mount pole allowed (radians)

All except `mount` are `doubles`; for `mount` the enumeration `spkMOUNT` offers `EQUAT` for an equatorial mount or `ALTAZ` for an altazimuth.[4]

The function `spkIobs` can be used to populate such structures.

[4]Other gimbal orientations could be supported by choosing `ALTAZ` and setting the `paw` and `pan` terms in the pointing model (see 3.3.5) to the appropriate large angles.

3.3.3 Ambient air conditions

The `spkAIR` structure contains three `doubles`:

- `p`: pressure (hPa ≡ mB)[5]
- `t`: temperature (°C)
- `h`: relative humidity (0–1)

It is a fortunate circumstance that the refraction is almost entirely dependent on these local measurements, rigorously so in the case of a plane-parallel atmosphere.

The function `spkIair` can be used to populate such structures.

3.3.4 Optics

The `spkOPT` structure contains two `doubles`:

- `fl`: focal length (user units)
- `wl`: wavelength (μm)

The sole purpose of providing the focal length is to allow the TCS developer to choose in what units to reckon focal-plane coordinates.

The crossover from optical to radio refraction is assumed to occur at `wl`=100 μm.

The function `spkIopt` can be used to populate such structures.

3.3.5 Pointing model

To summarize the mechanical imperfections of the telescope and mount, SPK uses a set of seven geometrical terms. The `spkPM` structure contains this seven-term model, supplemented by temporary adjustments to two of them. All data members are radians and `doubles`:

- `pia`: roll (−HA or π − azimuth) index error
- `pib`: pitch (δ or elevation) index error
- `pvd`: droop

[5] Pressures reported for a local airfield or read from an electronic weather station are usually what aviators call "QNH". This is the sea-level pressure and will always hover around 1013 hPa even for a mountaintop site. What SPK needs in order to calculate the refraction is "QFE", the actual pressure that would be read by a mercury barometer, a much lower figure for high-altitude sites.

- `pca`: telescope/pitch nonperpendicularity
- `pnp`: roll/pitch nonperpendicularity
- `paw`: roll axis misalignment across meridian
- `pan`: roll axis misalignment along meridian

and

- `ga`: guiding correction applied to `pca`
- `gb`: guiding correction applied to `pib`

The function `spkIpm` can be used to populate such structures.

It is important to understand that in practice the operational pointing model is likely to contain more than the basic seven terms, probably several times as many, each one making a contribution to whichever of the seven is most appropriate. Consequently, a TCS for a telescope that is to use a pointing modeler such as TPOINT, MaxPoint, PointXP *etc.* must contain code that translates the fitted model into SPK form.

3.3.6 Hotspot

The hotspot or "pointing origin" is the position in the focal plane that receives the image of the target. The `spkPO` structure contains two `double`s:

- `x`: x-coordinate
- `y`: y-coordinate

These coordinates are in the rotating focal plane, so that in the case of a camera mounted on an instrument rotator a specific pixel will have a fixed $[\,x, y\,]$. The units are whatever was chosen for the focal length in the `spkOPT` structure.

See Section 6 for information about the sign conventions.

The function `spkIpo` can be used to populate such structures.

3.3.7 Mount and rotator angles

The pointing algorithm needs estimates of the current orientation of the three drives (roll, pitch, rotator). The `spkAX3` structure contains three `double`s, all angles in radians:

- `a`: achieved roll (minus hour angle or 180° minus azimuth)
- `b`: achieved pitch (declination or elevation)

- `r`: achieved rotator angle

At first sight it may appear strange that an algorithm the principal use of which is to determine the orientations of the three drives should need *a priori* values for these very angles. The reasons are different for the mount and rotator respectively:

- The mount roll and pitch will be made available to the TCS application in anticipation of being needed for certain operational pointing corrections, for example gear errors. However, because such pointing corrections will not change much over small areas of sky there is no need for the supplied roll and pitch estimates to be particularly precise.
- The rotator angle is needed in order to deal with off-axis pointing origins (*i.e.* where the target's image is to be placed). Again, the precision requirements are fairly lax, depending how far the pointing origin is from the rotator axis.

Suitable values for populating the `spkAX3` structure may be obtained by extrapolating the current motions, using either encoder readings or the predicted demands as the basis.

See Section 6 for information about the sign conventions.

The function `spkIax3` can be used to populate such structures.

3.3.8 Time

The `spkUTC` structure contains a UTC date and time:

- `iy` : year CE
- `mo` : month (1-12)
- `id` : day (1-31)
- `ih` : hour (0-23)
- `mi` : minute (0-59)
- `sec`: seconds (0.0-61.0)

All the data members are `int`s except for `sec`, which is a `double`. The smallest illegal seconds field is almost always 60.0, but would be 61.0 during a positive leap second, which is exceedingly uncommon but nevertheless correctly handled by the SOFA tools used by `spkAstrom`.

The function `spkIutc` can be used to populate such structures.

It should be noted that in order to keep track of UTC leap seconds, SOFA includes a function `iauDat` that for a given UTC returns the TAI − UTC = ΔAT value in force at that time.

Consequently, in order to retain correct handling of leap seconds in the long term, the TCS application must be relinked with the latest SOFA library from time to time, say once a year, so that any updated `iauDat` can be picked up. A preferable alternative, explicitly allowed by the SOFA usage conditions, is for there to be a local variant of `iauDat`, for example one that obtains the current ΔAT from an internet source or even returns a fixed value manually updated whenever a leap second occurs.

3.3.9 Astrometry

The `spkAST` structure contains the target-independent parameters used by the SOFA astrometric functions:

- `tai`: TAI MJD (JD−2400000.5)
- `astrom`: star-independent astrometry quantities
- `eo`: equation of the origins (ERA−GST, radians)

The members `tai` and `eo` are `doubles`; `astrom` is a SOFA `iauASTROM` structure.

For an observed $[\,Az, El\,]$ target (see 3.3.10, below) the `spkAST` argument will not be accessed and its contents arbitrary.

The function `spkAstrom` can be used to populate such structures.

3.3.10 Target

The target is the pointing direction in the sky. The `spkTAR` structure contains the following:

- `sys`: reference system, ICRS, APPT or AZEL (see below)
- `a`: right ascension or left-handed azimuth
- `b`: declination or elevation

The spherical coordinates `a` and `b` are `doubles` and in radians; `sys`, the reference system, is made with the enumeration `spkSYS`:

- ICRS $[\,\alpha, \delta\,]$: International Celestial Reference System. This is close (maximum difference less than 25 mas) to mean J2000.0.
- APPT $[\,\alpha, \delta\,]$: geocentric apparent place.

- AZEL $[\,Az, El\,]$: observed (*i.e.* as affected by refraction) azimuth and altitude. Azimuth iz zero for due north and 90° for due east; altitude is called "elevation" in the present document. This is a left-handed system; internally SPK uses a south through east system, and this right-handed convention applies to one of the arguments returned by the `spkVtel` function.[6]

The functions `spkItar` and `spkCtar` offer two different ways to populate such structures.

3.4 The SPK functions

To use SPK, an application carries out the following steps:

1. Initialize or update all the required data structures (see 3.3 and 3.4.4).
2. Call either `spkAstrom` (p26) to refresh or `spkCtar` (p28) to update the astrometry parameters.
3. Call `spkVtel` (p38) to solve for one of the three virtual-telescope end-points, *i.e.* axis demands, image position or target coordinates.

Step 2 can be omitted in the special case of an `ALTAZ` target or if the time has not changed.

3.4.1 `spkAstrom`

The `spkAstrom` function provides star-independent astrometric quantities (precession matrices *etc.*) for the UTC specified by the caller. For the utmost accuracy, but at significant computational cost, the function can be called specifying a full update, but it also offers the cheaper option of updating only that part that depends on Earth rotation, which most of the time will be good enough. For further details see pp26-27.

3.4.2 `spkCtar`

The `spkCtar` function accepts a target specified as an epoch J2000.0 ICRS catalog $[\,\alpha, \delta\,]$ and calculates its current ICRS astrometric place ready to be used with SPK. (The astrometric place changes so slowly that in most TCS applications it can be used for the entire observing session.) For further details see p28.

[6] The need for a TCS to support two left-handed coordinate systems, namely $[\,h, \delta\,]$ and $[\,Az, El\,]$, is an ever-present tripwire for the software developer. Constant vigilance is essential, and it is strongly to be recommended that in code referring to these coordinates or right-handed equivalents there is a comment spelling out which variant is being used in that particular instance.

3.4.3 spkVtel

The spkVtel function implements the mount pointing formulas set out in paper W02. Its power comes from the fact that it supports all three pathways through Figure 1 and so can provide many different capabilities. The call accepts (i) an option code to specify which of the three pathways is to be taken, together with (ii) a collection of data structures (see 3.4.4, below) that define the context for the calculation; it then returns an array of values that depends on the option code isoln as follows:

- For the AXES option the values returned are the mount $[\,roll, pitch\,]$ and the position angle of the rotating focal plane's y-axis. There may be two sets of $[\,roll, pitch\,]$, the second corresponding to a "beyond the pole" posture.
- For the TARG option, a celestial posiiton is returned.
- For the HOTS option, the image $[\,x, y\,]$ is returned. typically one of the $[\,\alpha, \delta\,]$s.

In addition (and even if no option is specified), the pointing direction is returned in right-handed $[\,Az, El\,]$ and $[\,roll, pitch\,]$ for use when calculating terms in the operational pointing model.

The principal application for the AXES option is to point and track the telescope.

The TARG option might be used to establish the celestial world coordinate system of the focal plane. Another application would be measuring the $[\,\alpha, \delta\,]$ of a guide star that has just been acquired so that it can be used to detect future drifting of the image $[\,x, y\,]$.

The HOTS option might be used to predict the $[\,x, y\,]$ of a guide star image of known $[\,\alpha, \delta\,]$; the guider would compare this to the measured $[\,x, y\,]$ and take guiding action (such as changing the ga and gb members of a pointing model data structure.

Full details of spkVtel are given on pp38-40.

3.4.4 The spkI... functions

To assist with creating the small data structures that declare the contexts for the spkVtel calls, nine functions are provided. The names are such that a fictitious function called spkIxyz would initialize a fictitious data structure called spkXYZ. Note that use of these functions is optional; all data members are public and may be manipulated freely in application code.

3.5 Management of time

SPK has no direct connection with the system clock or any other timing signals, leaving these aspects completely to the TCS application. The pointing calculations are with respect to whatever UTC the caller provides when the spkAstrom function is called, and indeed there is

nothing to prevent different UTCs being used in different contexts, all proceeding in parallel. An example of this might be to explore the future track so as to establish by trial and error when cable-wrap or horizon limits will be reached.

A UTC time and date is supplied to `spkAstrom` as an instance of an `spkUTC` data structure, containing the fields year, month, day, hour, minute, seconds, together with an offset in seconds. The user may elect to supply a fresh `spkUTC` each time, with the offset always left at zero, or alternatively can leave the `spkUTC` fixed and specify different times by using the offset alone. The latter choice has the advantage that the full date and time has to be formatted just once, when the system starts, and can then run on for the rest of the session purely by using the offset.

4 The demonstration programs

SPK comes with two demonstration programs, which illustrate how the various transformations offered by the `spkVtel` function might be used in a TCS. One, `equat.c`, is for an equatorial mount while the other, `altaz.c`, is for an altazimuth. In order to demonstrate full three-axis control, both assume the presence of an instrument rotator, though many small equatorials manage without one.

The two programs are very similar, differing only in which sort of mount is declared during initialization, the names given to the mount angles in reports and the labels for pointing-model terms. This illustrates a crucial point about SPK (and the "virtual telescope" principle more generally) that there should be little or no difference between how the two types of mount are controlled. This may be overlooked by the TCS developer. For example, to displace a star image vertically in the field of view of an altazimuth telescope it may seem obvious that all that needs to be done is to introduce an offset into the demands going to the elevation drive. *But this is not the right thing to do.* The offset should instead be applied in terms of focal plane coordinates for a target being tracked in $[\,Az, El\,]$, just as it would be in the equatorial case. Doing so will deal with small field orientation effects caused by refraction and the pointing model, as well as differential refraction. The motion, almost but not entirely a change in the demands going to the elevation drive, will be a *consequence* of the changed pointing prescription, not something the developer should be conscious of.

Because the `equat.c` and `altaz.c` programs are almost identical, the description will be of both, with the occasional differences mentioned. Certain portions of the code are omitted for brevity; in particular, reports are reproduced but not the code that writes them.

4.1 Preliminaries

The code begins as a *pro forma* `main()` program and then declares some useful constants. Instances (in a couple of cases more than one) of the various data structures are then declared; these will become arguments of the SPK function calls, `spkVtel` in particular:

```
spkEOP eop;    /* polar motion and UT1-UTC (from IERS) */
spkOBS obs;    /* site location and mount type */
spkAIR air;    /* local weather readings */
spkOPT opt;    /* focal length and color */
spkPM pm;      /* 7-term pointing model */
spkAX3 ax3;    /* achieved roll, pitch, rotator angles */
spkPO po;      /* image position in focal plane */
spkUTC utc;    /* UTC date and time */
spkAST ast;    /* target-independent astrometric parameters */
spkTAR tar;    /* sky target */

spkTAR tar2;   /* another target */
spkPO po2;     /* another hotspot */
```

In order to make the reports clearer, names for the different supported target reference systems are declared:

```
static char *csys[] = {
   "illegal",
   "ICRS",
   "apparent",
   "altaz"
};
```

Miscellaneous variables are declared, with one small difference between the `altaz.c` and `equat.c` cases for cosmetic reasons.

The executable code begins with initialization of the data structures, the first being the Earth orientation parameters:

```
if ( spkIeop ( 50.995e-3*AS2R,        /* polar motion x */
               376.723e-3*AS2R,       /* polar motion y */
               155.0675e-3,           /* UT1-UTC */
               &eop ) ) return -1;
```

The EOPs are available from the International Earth Rotation and reference systems Service. An advanced TCS might obtain the parameters automatically via the internet; a simpler method is to read an initialization file that the observatory updates every few days. For a TCS with modest accuracy objectives the EOPs could simply be set to zero, though in the case of UT1−UTC this can lead to pointing errors of nearly 15 arcseconds.[7]

Next, details of the observatory are set up, namely geographical position and type of telescope mount; there is also an opportunity to declare a "no go" area around the mount pole, to prevent unrealistically rapid motion and other unruly behavior:

```
if ( iauAf2a ( '-', 5, 41, 54.2, &slon ) ) return -1;
if ( iauAf2a ( '-', 15, 57, 42.8, &slat ) ) return -1;
if ( spkIobs ( slon,                  /* East longitude */
               slat,                  /* latitude */
               625.0,                 /* altitude (m) */
               ALTAZ,                 /* mount type */
               0.5*D2R,               /* pole no-go */
               &obs ) ) return -1;
```

[7] The situation could get worse if proposals to cease UTC leap seconds, first made in the early 2000s, are eventually put into effect. The safest thing is for all TCSs to take into account UT1−UTC whatever the accuracy aspirations.

. . . and in the equatorial case:

```
  :
obs.mount = EQUAT;                          /* mount type */
  :
```

Local air conditions, needed for calculating atmospheric refraction, are declared:

```
if ( spkIair ( 952.0,                       /* pressure (hPA=mB) */
               18.5,                        /* temperature (Celsius) */
               0.83,                        /* humidity */
               &air ) ) return -1;
```

How frequently this information is updated depends on the TCS and the accuracy objectives, but once per new target may be adequate (or in some cases even once per observing session). Continuous updating from an electronic weather station is the ultimate solution, but some form of smoothing is essential if noisy readings are not to inject jitter into the telescope track. (See also the earlier note about sea-level pressure QNH versus local pressure QFE; it is the latter that SPK needs.)

The information SPK needs about the telescope optical configuration is the focal length and the observing wavelength:

```
if ( spkIopt ( 3000.0,                      /* focal length (mm) */
               0.55,                        /* wavelength (micrometers) */
               &opt ) ) return -1;
```

The only significance of focal length in the pointing calculation is that it defines the units in which $[x, y]$ image positions will be expressed. Entering the focal length as 1.0 would not affect the pointing calculations except that focal-plane $[x, y]$ coordinates are essentially radians. The wavelength plays a more fundametal part in that it affects the refraction calculation.

The starting amplitudes for the seven terms of the pointing model are set, together with small "trimming" adjustments to two of them:

```
if ( spkIpm ( 25.0*AS2R,                    /* +IA */
              15.0*AS2R,                    /* +IE */
              0.0*AS2R,                     /* +FLOP */
              -110.0*AS2R,                  /* +CA */
              8.0*AS2R,                     /* +NPAE */
              999.9*AS2R,                   /* +AW */
              -12.0*AS2R,                   /* +AN */
              0.0*AS2R,                     /* guiding correction to CA */
              0.0*AS2R,                     /* guiding correction to IE */
              &pm ) ) return -1;
```

. . . with only comment changes in the equatorial case:

```
if ( spkIpm ( 25.0*AS2R,              /* -IH */
              15.0*AS2R,              /* +ID */
              0.0*AS2R,               /* +FLOP */
              -110.0*AS2R,            /* -CH */
              8.0*AS2R,               /* -NP */
              999.9*AS2R,             /* -MA */
              -12.0*AS2R,             /* +ME */
              0.0*AS2R,               /* guiding correction to CH */
              0.0*AS2R,               /* guiding correction to ID */
              &pm ) ) return -1;
```

During the operation of the TCS the model will be updated, both continuously to allow for the effects of telescope attitude and abruptly to achieve special effects.

Starting values for the three axis positions are set:

```
if ( spkIax3 ( 0.0*D2R,               /* achieved azimuth (RH) */
               0.0*D2R,               /* achieved elevation */
               30.0*D2R,              /* achieved rotator angle */
               &ax3 ) ) return -1;
```

. . . again, with only comment changes in the equatorial case:

```
if ( spkIax3 ( 0.0*D2R,               /* achieved -HA */
               0.0*D2R,               /* achieved Dec */
               30.0*D2R,              /* achieved rotator angle */
               &ax3 ) ) return -1;
```

These three values will be actively controlled as tracking proceeds, from a combination of encoder readings and extrapolation.

A pointing origin (hotspot) is declared:

```
if ( spkIpo ( 10.0,                   /* x (mm) */
              20.0,                   /* y (mm) */
              &po ) ) return -1;
```

More typically the initialization process would set the pointing origin to $[0, 0]$ (*i.e.* the optical axis) and leave it up to the TCS application to manage where in the focal plane the image is being aimed at any given moment.

The starting UTC is specified:

```
if ( spkIutc ( 2013,                    /* year (CE) */
               4,                       /* month */
               2,                       /* day */
               23,                      /* hour */
               15,                      /* minute */
               43.55,                   /* seconds */
               &utc ) ) return -1;
```

In a real TCS this would of course come from a clock.

The star-independent astrometry parameters for that moment can now be calculated:

```
if ( spkAstrom ( 1, utc, 0.0,
                 &eop, &obs, &air, &opt, &ast ) ) return -1;
```

A celestial target is specified, starting from a catalog entry:

```
if ( iauTf2a ( ' ', 14, 34, 16.81183, &rc ) ) return -1;
if ( iauAf2a ( '-', 12, 31, 10.3965, &dc ) ) return -1;
pr = atan2 ( -354.45e-3 * AS2R, cos(dc) );
pd = 595.35e-3 * AS2R;
px = 164.99e-3;
rv = 0.0;
spkCtar ( rc, dc, pr, pd, px, rv, &ast, &tar );
```

We now specify the image orientation, the position angle with respect to the ICRS meridian of $+y$ at the specified hotspot:

```
ypa = 30.0 * D2R;
```

The choice of 30° is merely illustrative; it would be much more typical to use zero and an on-axis hotspot, meaning the focal plane's y-axis will point north.

4.2 Tracking

We are now ready to illustrate sidereal tracking. This is the part of the demonstrations that least resembles what an actual TCS would do, and is only a general indication of what calculations might be involved. The code shows just one way of computing the tracking demands, and gives some indication of the level of intricacy that a real TCS application would require.

At first sight all that is involved is to take the current target and hotspot and solve for the axis demands. However, there is an interplay between the three axes that has to be resolved, and the servo systems controlling the axis may well expect velocity feedforward information (and conceivably acceleration and higher derivatives). Consequently generating each new set of demands is likely to involve multiple VT solutions, and the precise way this is done will be something for the application developer to decide.

The demonstration starts one second before the UTC that has been specified, and at intervals of 0.05 SI seconds computes the demand angles for the three axes. We will also estimate velocities in the three axes; this is to enable extrapolation to the next iteration but might also be needed for servo reasons. It should also be noted that servo engineers tend to disapprove of even tiny irregularities in the tracking demands, and arranging the computations to avoid, or at least minimize, glitches in the demand stream is highly desirable.

We are assuming that the astrometry parameters have been computed for a particular moment using `spkAstrom` with `jfull = TRUE`, and that the track is relative to that time and involves only `jfull = FALSE` calls to `spkAstrom`. Some TCS developers will decide to call `spkAstrom` with `jfull = TRUE` every tracking step and accept the (considerable) computational expense. Most will opt instead for occasional (say once a minute) full updates, maintaining precision long-term at the expense of (sub-milliarcsecond) glitches. A full refresh only at the start of a new track is a workable plan, incurring errors of less than $0''.1$ for tracks several hours long and avoiding glitches (from this cause at least) completely.

In the demonstration code, updating the pointing model terms is performed each time through, but an economical implementation could elect to do it less frequently, say at 1 Hz. If the update intervals are too long, the corrections will not keep up with changing flexure *etc.* and will also inject small glitches into the tracking demands.

A noteworthy feature of the demonstration is that the velocities are **not** computed by differencing successive demands, though this approach would be efficient and may well look much as it does in the present simulation. The potential difficulty is in the general case, where the observing program is free to introduce abrupt changes to target, hotspot *etc.* The resulting transients can trigger adverse reactions in servo systems, so the demonstration code instead recomputes the demands from last time, but using the latest target and hotspot. Though computationally wasteful, this simple measure means that the velocities always reflect a steady background track.

The first step in the demonstration is to initialize various quantities and then enter the 1 second-at-20-Hz loop:

```
/* Initialize the per-timestep changes in the three axes. */
   da = 0.0;
   db = 0.0;
   dr = 0.0;

/* Initialize the time offset. */
   dt = 0.0;

/* The demonstration track (20Hz for 1s). */
   for ( i = 0; i <= 20; i++ ) {
```

Each tracking update starts by solving for the demands *last time*, but using the current circumstances in case there have been abrupt changes:

```
/* Solve for previous roll,pitch demands and field orientation. */
   j = spkVtel ( AXES, &obs, &opt, &pm, &ast, &ax3, &tar, &po,
                 &tara, &tare, &tarr, &tarp, soln );
     :

/* The roll, pitch and rotator demands then. */
   aold = soln[0];
   bold = soln[1];

/* The field orientation error then (actual minus desired). */
   e = iauAnpm ( soln[4] - ypa );

/* The rotator angle demand then. */
   rold = iauAnp ( ax3.r - e );
```

We now turn our attention to the present UTC, expressing it as an offset to the time originally supplied, and we update the astrometry to take account of Earth rotation:

```
/* New time offset (seconds) relative to provided UTC. */
   dt = ( (double) (i-20) ) / 20.0;

/* Update astrometry to that time. */
   if ( spkAstrom ( 0, iy, mo, id, ih, mi, sec, dt, &eop, &obs,
                    &air, &opt, &ast ) ) return -1;
```

We extrapolate the axis angles and recalculate, twice to let things settle, and noting the changes since last time:

```
/* Iterate to let roll/pitch/rotator angles settle. */
   for ( it = 0; it < 2; it++ ) {

   /* Extrapolated encoder readings. */
      ax3.a = aold + da;
      ax3.b = bold + db;
      ax3.r = rold + dr;

   /* Solve for roll,pitch demands and field orientation. */
      j = spkVtel ( AXES, &obs, &opt, &pm, &ast, &ax3, &tar, &po,
                    &tara, &tare, &tarr, &tarp, soln );
        :

   /* Set achieved roll,pitch to match. */
      ax3.a = soln[0];
      ax3.b = soln[1];

   /* The field orientation error (actual minus desired). */
      e = iauAnpm ( soln[4] - ypa );

   /* Set achieved rotator angle to eliminate that error. */
      ax3.r = iauAnp ( ax3.r - e );

   /* Changes since last time. */
      da = iauAnpm ( ax3.a - aold );
      db = iauAnpm ( ax3.b - bold );
      dr = iauAnpm ( ax3.r - rold );

   /* Next iteration. */
   }
```

Note that we have elected to change the `ax3` data members directly. This is convenient (and legal), but some developers might prefer to use the `spkIax3` function instead.

Next we update the pointing model, which may have changed slightly as a result of the change in attitude. The altazimuth...

```
if ( spkIpm ( 25.0*AS2R,            /* +IA */
              15.0*AS2R,            /* +IE */
              10.0*AS2R*cos(tare),  /* +TF */
              -110.0*AS2R,          /* +CA */
              8.0*AS2R,             /* +NPAE */
```

```
                999.9*AS2R,           /* +AW */
                -12.0*AS2R,           /* +AN */
                0.0*AS2R,             /* guiding correction to CA */
                0.0*AS2R,             /* guiding correction to IE */
                &pm ) ) return -1;
```

. . . and equatorial. . .

```
if ( spkIpm ( 25.0*AS2R,            /* -IH */
              15.0*AS2R,            /* +ID */
              10.0*AS2R*cos(tare),  /* +TF */
              -110.0*AS2R,          /* -CH */
              8.0*AS2R,             /* -NP */
              999.9*AS2R,           /* -MA */
              -12.0*AS2R,           /* +ME */
              0.0*AS2R,             /* guiding correction to CH */
              0.0*AS2R,             /* guiding correction to ID */
              &pm ) ) return -1;
```

. . . cases are the same apart from the comments.

The tracking illustration is complete. In the altazimuth case the report is as follows:

```
    TAI              azimuth        elevation       rotator

56384.9696579861   91.5962372761  47.3611395498  309.8100307441  deg
                   -3.6793457637 +14.4476380766   -0.6156263786  "/s

56384.9696585648   91.5961861739  47.3613402043  309.8100221923  deg
                   -3.6793559349 +14.4476382731   -0.6158962760  "/s

56384.9696591435   91.5961350716  47.3615408588  309.8100136573  deg
                   -3.6793651517 +14.4476382905   -0.6150903831  "/s
    MJD
```

The equatorial report shows an HA tracking rate of about 15 arcseconds per sidereal second and approximately zero in elevation and rotator angle:

```
    TAI              hour angle    declination      rotator

56384.9696579861   43.6619835316 -13.0082797782  210.2268042506  deg
                  +15.0454774090  +0.0546879527   +0.0550319532  "/s

56384.9696585648   43.6617745739 -13.0082790175  210.2268050260  deg
                  +15.0454764623  +0.0546867975   +0.0552061177  "/s

56384.9696591435   43.6615656161 -13.0082782568  210.2268057990  deg
                  +15.0454779687  +0.0546891264   +0.0555078920  "/s
    MJD
```

4.3 The three pathways

We are ready to to demonstrate the three pathways through Figure 1 that `spkVtel` implements. These are the basic ingredients that can be combined to achieve not only the key functions of pointing, tracking and autoguiding, but also establishing an image's world coordinate system, offsetting between acquisition camera and instrument or between multiple feed horns, trailing along a vertical spectrograph slit, chopping between source and sky, and so on.

4.3.1 Target and hotspot to mount angles

The first pathway is just a repeat of the final call of the tracking demonstration; we know the target $[\alpha, \delta]$ and hotspot $[x, y]$, and wish to calculate the corresponding mount $[roll, pitch]$ and also the field orientation, requiring `spkVtel`'s `isoln` option to be `AXES`:

```
j = spkVtel ( AXES, &obs, &opt, &pm, &ast, &ax3, &tar, &po,
              &tara, &tare, &tarr, &tarp, soln );
```

The report from `altaz.c` is as follows:

```
Given:
   TAI = 56384.969659144 MJD
   RMA =   309.810013657 deg
   target = 14 34 16.4960  -12 31 02.524  ICRS
   hotspot [x,y] =    10.000000,    20.000000 mm
Returned:
   encoder [Az,El] =    91.596135072 (N thru E),    47.361540852 deg
   PA of +y =    30.000000009 deg (ICRS)
```

The quoted position angle is the direction of $+y$ at the pointing origin, with respect to the stated celestial coordinate system, in this case ICRS. In other words, zero would mean that at the pointing origin $+y$ points north. If exactly the same pointing direction in the sky were to be specified as a geocentric apparent place, the position angle would again be different, by up to 180° for a point between the ICRS and apparent poles. The `equat.c` report is similar, differing only in the mount axis angles:

```
Given:
   TAI = 56384.969659144 MJD
   RMA =   210.226805799 deg
   target = 14 34 16.4960  -12 31 02.524  ICRS
   hotspot [x,y] =    10.000000,    20.000000 mm
Returned:
   encoder [HA,Dec] =    43.661565623 (W +ve),   -13.008278256 deg
   PA of +y =    29.999999999 deg (ICRS)
```

4.3.2 Mount angles and hotspot to target

The second pathway through Figure 1 is where everything stays the same except that the pointing origin $[\,x, y\,]$ has changed to [25.0,27.5] and we set `isoln - TARG`:

```
po2.x = 25.0;
po2.y = 27.5;
j = spkVtel ( TARG, &obs, &opt, &pm, &ast, &ax3, &tar, &po2,
              &tara, &tare, &tarr, &tarp, soln );
```

The reports show that for the same mount and rotator demands the new hotspot corresponds to a new $[\,\alpha, \delta\,]$. In the `altaz.c` case:

```
Given:
   TAI = 56384.969659144 MJD
   RMA =   309.810013657 deg
   hotspot [x,y] =    25.000000,    27.500000 mm
   encoder [Az,El] =    91.596135072 (N thru E),    47.361540859 deg
Returned:
   target = 14 35 35.1364  -12 32 10.772  ICRS
```

. . . and for `equat:`:

```
Given:
   TAI = 56384.969659144 MJD
   RMA =   210.226805799 deg
   hotspot [x,y] =    25.000000,    27.500000 mm
   encoder [HA,Dec] =    43.661565616 (W +ve),   -13.008278257 deg
Returned:
   target = 14 35 35.1364  -12 32 10.772  ICRS
```

4.3.3 Mount angles and target to hotspot

The third and final pathway through Figure 1 accepts a target and for given mount and rotator demands predicts the focal-plane $[x, y]$ that will receive the image. For the demonstration the mount and rotator demands are left as they are and we use as a target the $[\alpha, \delta]$ found in the previous example, with `spkVtel`'s `isoln` set to `HOTS`:

```
if ( iauTf2a ( ' ', 14, 35, 35.1363560, &rc ) ) return -1;
if ( iauAf2a ( '-', 12, 32, 10.771790, &dc ) ) return -1;
spkCtar ( rc, dc, 0.0, 0.0, 0.0, 0.0, &ast, &tar2 );
j = spkVtel ( HOTS, &obs, &opt, &pm, &ast, &ax3, &tar2, &po,
              &tara, &tare, &tarr, &tarp, soln );
```

The code is identical in `altaz.c` and `equat.c`. The `altaz.c` report is:

```
Given:
TAI = 56384.969659144 MJD
RMA =   309.810013657 deg
target = 14 35 35.1364  -12 32 10.772  ICRS
encoder [Az,El] =    91.596135072 (N thru E),    47.361540859 deg
Returned:
   hotspot [x,y] =    25.000000,    27.500000 mm
```

... recovering the test $[x, y]$ that we used. The `equat.c` report is similar:

```
Given:
   TAI = 56384.969659144 MJD
   RMA =   210.226805799 deg
   target = 14 35 35.1364  -12 32 10.772  ICRS
   encoder [HA,Dec] =    43.661565616 (W +ve),   -13.008278257 deg
Returned:
   hotspot [x,y] =    25.000000,    27.500000 mm
```

That completes the three Figure 1 pathways.

4.4 Direction of the vertical

Even though a target is usually tracked in $[\alpha, \delta]$, and the field orientation requirement is given relative to north, it is often necessary to know which direction in the focal plane corresponds to the vertical. This is easy to do, simply by specifying the pointing direction as an $[Az, El]$ target but leaving other `spkVtel` arguments as they were. The latest call to `spkVtel` has already returned the pointing direction as an $[Az, El]$, so the new target is simply that but in left-handed form:

```
tar2.sys = AZEL;
tar2.a = D180 - tara;
tar2.b = tare;
j = spkVtel ( AXES, &obs, &opt, &pm, &ast, &ax3, &tar2, &po,
              &tara, &tare, &tarr, &tarp, soln );
```

Note that the original hotspot has been retained, so it is the vertical through that point that we are interested in. The `altaz.c` report is:

```
Given:
   TAI = 56384.969659144 MJD
   RMA =   309.810013657 deg
   target =    88.499802127 (N thru E),    46.938342447 deg (altaz)
   hotspot [x,y] =    10.000000,    20.000000 mm
Returned:
   encoder [Az,El] =    91.596135072 (N thru E),    47.361540859 deg
   PA of +y =   129.948849398 deg (altaz)
```

...and the `equat.c` report is similar:

```
Given:
   TAI = 56384.969659144 MJD
   RMA =   210.226805799 deg
   target =    88.499802127 (N thru E),    46.938342447 deg (altaz)
   hotspot [x,y] =    10.000000,    20.000000 mm
Returned:
   encoder [HA,Dec] =    43.597998256 (W +ve),   -11.016168751 deg
   PA of +y =   129.948849392 deg (altaz)
```

The demonstrations end at this point.

5 Function specifications

---

spkAstrom *update astrometry context* **spkAstrom**

CALL :

```
j = spkAstrom ( jfull, utc, dt, &eop, &obs, &air, &opt, &ast );
```

ACTION :

For a specified UTC, update the star-independent astrometry parameters.

GIVEN :

jfull	`int`	TRUE/FALSE = full/partial update (Notes 1,2)
utc	`spkUTC`	UTC date and time (Notes 2-4)
dt	`double`	offset (SI seconds, Note 2)
eop	`spkEOP*`	Earth orientation parameters
obs	`spkOBS*`	site location and mount type
air	`spkAIR*`	ambient weather conditions
opt	`spkOPT*`	for color

RETURNED :

ast	`spkAST*`	astrometry parameters (Note 1)

RETURNED (function value) :

`int`	status: +3 = both of next two
	+2 = time is after end of day (Note 3)
	+1 = dubious year (Note 4)
	0 = OK
	−1 = bad year
	−2 = bad month
	−3 = bad day
	−4 = bad hour
	−5 = bad minute
	−6 = bad second (< 0)
	−7 = other error (Note 5)

NOTES :

1. The flag `jfull` selects whether the entire set of star-independent astrometry parameters are refreshed (`jfull = TRUE`) or just the Earth rotation angle. Other quantities such as precession matrices are not recomputed in the `jfull = FALSE` case, and will as time goes on become less accurate. How frequently the parameters are fully refreshed is a decision for the application developer, but once for each new target could be adequate. The TAI included in the star-independent astrometry parameters corresponds to the ERA.

2. The UTC date and time is supplied as an `spkUTC` structure, containing y,m,d,h,m,s fields, with an offset in seconds. The user may elect to supply a fresh `spkUTC` structure each time, with the offset always zero, or can supply the same spkUTC structure each time and move forward using the offset alone.

3. The warning status "time is after end of day" usually means the second argument is greater than 60.0. However, in a day ending in a leap second the limit changes to 61.0 (or 59.0 in the case of a negative leap second).

4. The warning status "dubious year" flags UTCs that predate the introduction of the time scale or that are too far in the future to be trusted. See the `iauDat` function for further details.

5. The error status −7 should be impossible.

spkCtar *accept ICRS catalog star as target* spkCtar

CALL :

```
spkCtar ( rc, dc, pr, pd, px, rv, &ast, &tar );
```

ACTION :

Transform an epoch J2000.0 ICRS star catalog entry into an ICRS astrometric target.

GIVEN :

rc	`double`	ICRS right ascension at J2000.0 (radians, Note 1)
dc	`double`	ICRS declination at J2000.0 (radians, Note 1)
pr	`double`	α proper motion (radians/year, Note 2)
pd	`double`	δ proper motion (radians/year)
px	`double`	parallax (arcsec, Note 3)
rv	`double`	radial velocity (km/s, +ve if receding)
ast	`spkAST*`	astrometry parameters

RETURNED :

tar	`spkTAR*`	target, ICRS astrometric

NOTES :

1. Star data for an epoch other than J2000.0 (for example from the Hipparcos catalog, which has an epoch of J1991.25) will require preliminary treatment.

2. The proper motion in α is $\dot{\alpha}$ rather than $\dot{\alpha}\cos\delta$.

3. In catalogs where the α and δ proper motions are given in milliarcseconds per year, both will need scaling into radians before the μ_α value is divided by $\cos\delta$. *n.b.* Some catalogs quote proper motions per century rather than per year.

4. The parallax value is in arcseconds rather than radians as it is a distance measure rather than an angle.

spkIair *initialize an ambient air conditions structure* spkIair

CALL :

```
j = spkIair ( p, t, h, &air );
```

ACTION :

Initialize an `spkAIR` ambient air conditions structure.

GIVEN :

p	`double`	pressure (hPa ≡ mB)
t	`double`	temperature (∘C)
h	`double`	relative humidity (0-1)

RETURNED :

air	`spkAIR*`	ambient air conditions

RETURNED (function value) :

	`int`	status: 0 = OK else = errors

spkIax3 *initialize an axis orientations structure* spkIax3

CALL :

```
j = spkIax3 ( a, b, r, &ax3 );
```

ACTION :

Initialize an `spkAX3` structure.

GIVEN :

a	`double`	achieved −HA or π−Az (radians)
b	`double`	achieved Dec or El (radians)
r	`double`	achieved rotator angle (radians)

RETURNED :

ax3	`spkAX3*`	orientations of the three axes

RETURNED (function value) :

	`int`	status: 0 = OK else = errors

NOTE :

No validation is attempted, and consequently the status returned is always zero.

spkIeop *initialize an EOP structure* spkIeop

CALL :

```
j = spkIeop ( xp, yp, dut1, &eop );
```

ACTION :

Initialize an `spkEOP` Earth orientation parameters structure.

GIVEN :

xp	`double`	polar motion in x (radians)
yp	`double`	polar motion in y (radians)
dut1	`double`	UT1−UTC (seconds)

RETURNED :

eop	`spkEOP*`	Earth orientation parameters

RETURNED (function value) :

	`int`	status: 0 = OK else = errors

NOTE :

The parameters can be obtained from the International Earth Rotation and reference systems Service.

spkIobs *initialize an observatory structure* spkIobs

CALL :

```
j = spkIobs ( slon, slat, sh, mount, pavoid, &obs );
```

ACTION :

Initialize an `spkOBS` observatory structure.

GIVEN :

slon	`double`	site east longitude (ITRF, radians)
slat	`double`	site geodetic latitude (ITRF, radians)
sh	`double`	site altitude above WGS84 ellipsoid (m)
mount	`spkMOUNT`	mount type
pavoid	`double`	closest to mount pole allowed (radians)

RETURNED :

obs	`spkOBS*`	observatory

RETURNED (function value) :

	`int`	status: 0 = OK else = errors

spkIopt *initialize an optics structure* spkIopt

CALL :

```
j = spkIopt ( fl, wl, &opt );
```

ACTION :

Initialize an `spkOPT` optics structure.

GIVEN :

fl	`double`	focal length (user units)
wl	`double`	wavelength (μm)

RETURNED :

opt	`spkOPT*`	optics

RETURNED (function value) :

	`int`	status: 0 = OK else = errors

spkIpm *initialize a pointing model structure* spkIpm

CALL :

```
j = spkIpm ( pia, pib, pvd, pca, pnp, paw, pan, ga, gb, &pm );
```

ACTION :

Initialize an `spkPM` pointing origin structure.

GIVEN :

pia	`double`	IA: roll ($-$HA or $\pi -$ azimuth) index error
pib	`double`	IB: pitch (Dec or elevation) index error
pvd	`double`	VD: downward droop
pca	`double`	CA: telescope/pitch nonperpendicularity
pnp	`double`	NP: roll/pitch nonperpendicularity
paw	`double`	AW: roll axis misalignment across meridian
pan	`double`	AN: roll axis misalignment along meridian
ga	`double`	dCA guiding correction
gb	`double`	dIB guiding correction

RETURNED :

pm	`spkPM*`	pointing model

RETURNED (function value) :

	`int`	status: 0 = OK else = errors

NOTES :

1. All the coefficients are in radians.
2. No validation is attempted, and consequently the status returned is always zero.

spkIpo — *initialize a pointing origin structure* — spkIpo

CALL :

```
j = spkIpo ( x, y, &po );
```

ACTION :

Initialize an `spkPO` pointing origin structure.

GIVEN :

x	`double`	x-coordinate in rotating focal plane (user units)
y	`double`	y-coordinate in rotating focal plane (user units)

RETURNED :

po	`spkPO*`	pointing origin

RETURNED (function value) :

	`int`	status: 0 = OK else = errors

NOTES :

1. The units are the same as for the focal length supplied as part of the `spkOPT` structure.
2. No validation is attempted, and consequently the status returned is always zero.

spkItar *initialize a target structure* spkItar

CALL :

```
j = spkItar ( sys, a, b, &tar );
```

ACTION :

Initialize an `spkTAR` target structure.

GIVEN :

sys	`spkSYS`	`ICRS`, `APPT` or `AZEL`
a	`double`	RA or left-handed azimuth (radians)
b	`double`	Dec or elevation (radians)

RETURNED :

tar	`spkTAR*`	target

RETURNED (function value) :

	`int`	status: 0 = OK else = errors

spkIutc — *initialize a time (UTC) structure* — spkIutc

CALL :

```
j = spkIutc ( iy, mo, id, ih, mi, sec, &utc );
```

ACTION :

Initialize an `spkUTC` time (UTC) structure.

GIVEN :

iy	`int`	year CE
mo	`int`	month
id	`int`	day
ih	`int`	hour
mi	`int`	minute
sec	`double`	seconds

RETURNED :

utc	`spkUTC*`

RETURNED (function value) :

`int`	status: +3 = both of next two +2 = time is after end of day (Note 1) +1 = dubious year (Note 2) 0 = OK −1 = bad year −2 = bad month −3 = bad day −4 = bad hour −5 = bad minute −6 = bad second (< 0)

NOTES :

1. The warning status "time is after end of day" usually means that the `sec` argument is greater than 60.0. However, in a day ending in a leap second the limit changes to 61.0 (or 59.0 in the case of a negative leap second).

2. The warning status "dubious year" flags UTCs that predate the introduction of the time scale or that are too far in the future to be trusted.

spkVtel *solve virtual telescope* spkVtel

CALL :

```
j = spkVtel ( isoln, &obs, &opt, &pm, &ast, &ax3, &tar, &po,
                              &tara, &tare, &tarr, &tarp, soln[5] );
```

ACTION :

Solve a "virtual telescope" (Note 1).

GIVEN :

isoln	`spkVTS`	what to solve for (Notes 3,4)
obs	`spkOBS*`	observatory
opt	`spkOPT*`	optics (for focal length)
pm	`spkPM*`	pointing model (Note 4)
ast	`spkAST*`	astrometry parameters (Note 6)
ax3	`spkAX3*`	achieved roll, pitch, rotator angles (radians)
tar	`spkTAR*`	target (ICRS, apparent or altazimuth)
po	`spkPO*`	hotspot $[x, y]$ (Note 2, same units as focal length)

RETURNED :

tara	`double*`	target observed azimuth (N thru E, radians)
tare	`double*`	target observed elevation (radians)
tarr	`double*`	target roll (radians)
tarp	`double*`	target pitch (radians)
soln	`double[5]`	solution (Notes 4,9)

RETURNED (function value) :

	`int`	status: $+1$ = pole avoidance occurred
		0 = OK
		-1 = no solutions
		-2 = geometrically impossible
		-3 = illegal target system

NOTES :

1. The virtual telescope (see reference) is a data construct that describes the relationship between three aspects of pointing a telescope, namely:

   - the coordinates of the celestial target,
   - the position of the target's image in the focal plane, and
   - the angles to which the mount axes must be set in order to form the image.

Given any two of these, along with certain other information, including astrometric parameters some of which are changing with time, the third can be deduced.

Thus a telescope can track a star by knowing the $[\alpha, \delta]$ and the desired image $[x, y]$, and repeatedly solving for mount angles as the astronomical transformations progress. Or the mapping between sky coordinates and focal plane coordinates can be sampled, in both directions.

2. Focal plane $[x, y]$ is left-handed when projected onto the sky. Zero rotator angle is when the projection on the sky of the y-axis points at the negative mount pole, and the angle increases counter-clockwise on the sky.

3. The argument `isoln` specifies what quantities are to be solved for:

   - If `isoln = AXES`, the result is the mount axis angles demands required to image the target onto the desired $[x, y]$, known as the "hotspot". There can be two sets of these angles, one corresponding to the "beyond the pole" state of the mount. Also computed is the field orientation, in the form of the position angle of the rotating focal plane's y-axis. *n.b.* The given argument `ax3` is needed for other purposes and must contain valid "achieved" angles.
     - If `isoln = HOTS`, the result is the $[x, y]$ position of the image in the rotating focal plane. The given argument `po` is not used and need not be initialized.
     - If `isoln = TARG`, the result is the celestial coordinates of the target. The supported systems are ICRS astrometric $[\alpha, \delta]$, geocentric apparent $[\alpha, \delta]$ and observed $[Az, El]$, the latter left-handed (north through east). The given argument `tar` is needed for other purposes and must contain a valid target.
     - If `isoln = DEFT` (or any other value) no solution takes place – see the next sentence.

4. Irrespective of the `isoln` value, the target's observed place as right-handed $[Az, El]$ and mount $[roll, pitch]$ are returned, for general purposes including the calculation of pointing-model terms.

5. The pointing model `pm` comprises seven terms, called IA, IB, VD, CA, NP, AW and AN. A typical operational pointing model would contain a greater variety, each being mapped onto the most natural one of the seven.

6. The astrometry parameters `ast` can be formed by calling the `spkAstrom` function with the appropriate UTC specified. In `spkAstrom` the argument `jfull` controls whether a complete recalculation is to be carried out or merely the Earth rotation angle. A common strategy would be to recompute the parameters prior to acquiring a new target (or even once a night), and to update only the ERA at the rate of the tracking loop, typically 20 Hz.

7. The returned `tara` and `tare` are the observed place (*i.e.* the pointing direction) as left-handed $[Az, El]$, where azimuth runs north through east.

8. The returned `tarr` and `tarp` are the target observed place in mount roll,pitch, called `AIM` in the code. For an equatorial mount, the nominal orientation of the $[roll, pitch]$ frame makes `tarr` minus hour angle and `tarp` declination. For an

altazimuth mount, the nominal orientation makes `tarr` π minus the north-through-east azimuth and `tarp` elevation (more properly called altitude). For both types of mount, the actual orientation depends on data members `pan` and `paw` in the `spkPM` pointing-model structure. These are typically a few (or a few tens of) arcseconds, and are called polar misalignment for an equatorial and azimuth axis tilt in an altazimuth.

9. The `soln` array receives from one to five doubles, depending on the `isoln` argument, as follows:

`isoln`	`AXES`	`HOTS`	`TARG`	*else*
`soln[0]`	r	x	f	-
`soln[1]`	p	y	g	-
`soln[2]`	rb	-	-	-
`soln[3]`	pb	-	-	-
`soln[4]`	θ	-	-	-

Unused `soln` elements (marked -) are undefined. Everything but x and y are radians.

For the `AXES` solution, the results r, p and rb, pb are the $[\,roll, pitch\,]$ mount pointing demands needed to image the target onto the hotspot. The "b" pair are for the "beyond the pole" configuration. It is up to the telescope control application to decide whether r, p or rb, pb are to be used as drive demands. For equatorials the 'b" set represent below the pole in a fork or horseshoe mount, and one of the meridian flip cases in GEMs and cross-axis mounts. The "b" set are not commonly used for altazimuths, though they could be in some designs. The θ result is the position angle at the pointing origin of the rotating focal plane's $+y$ direction, with respect to the target's celestial coordinate system, counter-clockwise on the sky. In an altazimuth telescope this can be used to generate the tracking demands for the rotator. The same is true for equatorials, though usually the rotator is not driven continuously.

For the `HOTS` solution, the x and y results are the coordinates of the target image given the current mount pointing. They are in the units chosen for the telescope focal length in the `spkOBS` structure, for example millimeters, and are on the rotating focal plane, so will stay constant for a specific place on an instrument, such as a spectrograph slit.

For the `TARG` solution, the f and g results are the sky position that corresponds to the hotspot $[\,x, y\,]$ given the current mount pointing. This is an $[\,\alpha, \delta\,]$ when the target's system is `ICRS` or `APPT`, or $[\,Az, El\,]$ (left-handed) for the `AZEL` case.

6 Coordinate conventions

The coordinate conventions that SPK uses are a fruitful cause of confusion, mainly for historical reasons. The present section spells out exactly what they are.

As far as possible, the SPK code deals internally with right-handed systems, transforming into left-handed systems only to harmonize with existing astronomical practice. It is advisable in code to provide explanatory comments each and every time there is opportunity for misunderstanding.

Angles may be expressed in any range (in particular $\pm\pi$ or $0 - 2\pi$) without changing meaning as far as SPK is concerned. A TCS which had reason to be careful about this aspect, for example where dealing with cable wraps and limits, would have to condition values returned by SPK appropriately.

6.1 Vectors

Throughout SPK, vectors are right-handed. There are, for example, no cases where left-handed spherical coordinates such as $[\,Az, El\,]$ are transformed into 3-vectors using standard formulas; the angular coordinates would be converted into right-handed forms before transformation.

6.2 Spherical coordinates

Celestial coordinates in $[\,\alpha, \delta\,]$ form are naturally right-handed, and no special care is needed dealing with them.

The opposite is true of $[\,h, \delta\,]$, which is a left-handed system. The right-handed form is $[\,-h, \delta\,]$.

The other awkward case is $[\,Az, El\,]$, with the additional complication that in practice different zero points are encountered. In SPK the left-handed form is clockwise from north as seen from above, while the right-handed form is counter-clockwise from south.

SPK calls generic mount coordinates $[\,roll, pitch\,]$, a right-handed system. For equatorial mounts, zero roll is on the meridian; for altazimuth mounts, zero roll is south. In code that transforms $[\,roll, pitch\,]$ into either $[\,Az, El\,]$ or $[\,h, \delta\,]$ there must be comments spelling out exactly what is happening.

6.3 Focal plane

The focal plane $[\,x, y\,]$ system is right-handed as seen by the light approaching the focal plane being delivered from Cassegrain optics or another non-mirror-imaging optical design. This means that **when projected onto the sky** the $[\,x, y\,]$ axes are **left-handed**.

6.4 Rotator angle

Zero rotator angle is when the projection on the sky of the y-axis points at the negative mount pole, hence downwards in an altazimuth and south in an equatorial. The angle increases counter-clockwise on the sky.

7 Files

The following lists all the SPK files and the purpose of each one:

`altaz.c`	C source for the `ALTAZ` demonstrator
`astrom.c`	C source for the `spkAstrom` function
`ctar.c`	C source for the `spkCtar` function
`equat.c`	C source for the `EQUAT` demonstrator
`iair.c`	C source for the `spkIair` function
`iax3.c`	C source for the `spkIax3` function
`ieop.c`	C source for the `spkIeop` function
`iobs.c`	C source for the `spkIobs` function
`iopt.c`	C source for the `spkIopt` function
`ipm.c`	C source for the `spkIpm` function
`ipo.c`	C source for the `spkIpo` function
`itar.c`	C source for the `spkItar` function
`iutc.c`	C source for the `spkIutc` function
`spk.h`	C source for the `spk.h` header file
`spk.pdf`	documentation PDF
`spk.tex`	documentation LaTeX source
`vtel.c`	C source for the `spkVtel` function