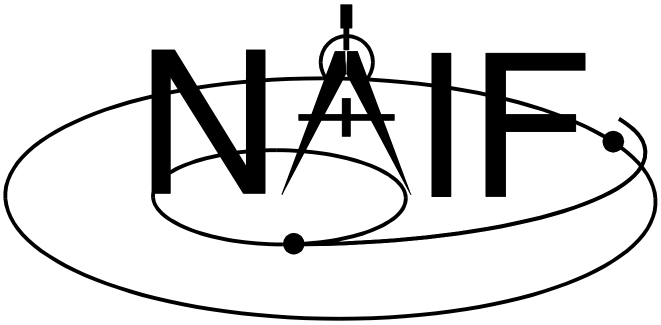
Most Useful SPICELIB Subroutines



NAIF

December 2004

Introduction

This document contains a brief description of the most often used SPICELIB routines. These are routines for reading SPICE files, solving tasks of 2-D and 3-D geometry and dealing with time conversions.

In this document routines are grouped by category. A list of these categories and actions performed by the particular routines is in the first chapter.

For each routine in this document you will find the following information:

1. a short description of action performed by routine;
2. calling sequence;
3. declaration of the routine's arguments.

Full information for each routine can be found in the header section of the routine's source code.

Units for arguments of SPICELIB routines described here are kilometers for distances and radians for angles.

Usage as CSPICE reference

Although this document describes interfaces of the FORTRAN-language version of the SPICE Toolkit (SPICELIB), it can also be used as a reference to the functionality provided in the C-language version of the SPICE Toolkit (CSPICE) and IDL version of the SPICE Toolkit (ICY). Each of the SPICELIB subroutines mentioned in this reference has an equivalent CSPICE function with the same name but with "_c" appended at the end and ICY function with the same name but "cspice_" appended at the beginning.

Solar system bodies ID codes. ***page 7***
- list of SPICE ID codes for solar system bodies.

Spacecraft and instruments ID codes. ***page 8***
- list of SPICE ID codes for spacecraft and instruments;
- using text kernel files for defining new ID code-name mappings.

Inertial reference frames. ***page 9***
- list of codes and names for inertial reference frames;
- list of names for body–fixed rotation frame;
- other frames used within the SPICE system.

Unloading and loading SPICE kernel files. ***page 10***
- loading of a kernel file or a set of kernel files;
- unloading of a kernel file or a set of kernel files.

Universal and ephemeris times. ***page 11***
- loading of an LSK file containing ET–UTC mapping data;
- calculation of ephemeris time ET from universal time UTC;
- calculation of universal time UTC from ephemeris time ET.

Spacecraft on-board time (SCLK). ***page 12***
- loading of an SCLK file containing ET–SCLK mapping data;
- transformation of SCLK given as character string to its double precision encoding;
- transformation of encoded double precision SCLK to corresponding character string;
- calculation of encoded SCLK for a given ephemeris time ET;
- calculation of ephemeris time ET for SCLK given as a double precision encoding;
- calculation of character string SCLK for a given ephemeris time ET;
- calculation of ephemeris time for SCLK given as a character string.

Constants and matrixes of planets and satellites (PCK). ***page 13***
- loading of a PCK file containing physical constants for planets and satellites;
- calculation of transformation matrix from inertial to body-fixed reference frame;
- calculation of transformation matrix from inertial “J2000” to body-fixed reference frame;
- calculation of Euler angles defining transformation from inertial “J2000” to body-fixed reference frame;
- retrieval of the values of body’s physical parameters from previously loaded data.

Frame transformations (FRAMES). ***page 14***
- loading of a FRAMES kernel file containing project specific frame definitions;
- calculation of state transformation matrix, rotating state vectors (position and velocity) from one frame to another;
- calculation of position transformation matrix, rotating position vectors from one frame to another;
- naming convention for inertial and PCK-based frames;
- naming convention for user-defined (CK-based and fixed offset) frames;

Planet and spacecraft positions (SPK). ***page 15***
- loading of an SPK file containing solar system bodies and spacecraft trajectory data;
- calculation of apparent, true or geometric state (position and velocity) of one body (planet, satellite, spacecraft, ...) with respect to another body for a given ephemeris time ET;
- calculation of apparent, true or geometric position of one body (planet, satellite, spacecraft, ...) with respect to another body for a given ephemeris time ET;
- unloading of a previously loaded SPK file.

Attitude of spacecraft and instrument platforms (CK). ***page 16***
- loading of a CK file containing spacecraft attitude data;
- calculation of state or posititon transformation matrix, rotating state or posititon vectors from one frame to another, either of which can be a CK-based frame;
- calculation of the attitude matrix or the attitude matrix and angular velocity of rotation of spacecraft reference frame relative to a given inertial frame for a given encoded SCLK;
- unloading of a previously loaded CK file.

Scientific instruments parameters (IK). ***page 17***
- loading of an IK file containing data describing a scientific instrument;
- retrieval of the values of instrument parameters from previously loaded data;
- retrieval of the instrument field-of-view parameters from previously loaded data;
- IK naming convention for instrument parameters;
- specification of a set of IK keywords used to hold instrument field-of-view definition.

Spacecraft Event Information (Data Base Kernel). ***page 18***
- loading a SPICE EK (DBK) file containing spacecraft event data;
- searching EK data satisfying a set of constraints specified in a query string;
- returning integer, double precision or character elements of an EK data record found during a search;

Physical and mathematical constants. ***page 20***
- values of π, π/2 and 2π;
- numbers of degrees per radian and radians per degree;
- number of seconds per day;
- IAU official value of light speed in vacuum;
- values of Julian date for B1900, B1950, J1900, J1950, J2000 and J2100.

3-dimensional geometry

Rectangular coordinates. ***page 21***
- cylindrical coordinates of a point from its rectangular coordinates;
- geodetic coordinates of a point from its rectangular coordinates;
- latitudinal coordinates of a point from its rectangular coordinates;
- right ascension, declination and distance to origin of a point from its rectangular coordinates;
- spherical coordinates of a point from its rectangular coordinates.

Spherical and cylindrical coordinates. ***page 22***
- latitudinal coordinates of a point from its cylindrical coordinates;
- rectangular coordinates of a point from its cylindrical coordinates;
- spherical coordinates of a point from its cylindrical coordinates;
- cylindrical coordinates of a point from its spherical coordinates;
- latitudinal coordinates of a point from its spherical coordinates;
- rectangular coordinates of a point from its spherical coordinates.

Latitudinal and geodetic coordinates. ***page 23***
- cylindrical coordinates of a point from its latitudinal coordinates;
- rectangular coordinates of a point from its latitudinal coordinates;
- spherical coordinates of a point from its latitudinal coordinates;
- rectangular coordinates of a point from its geodetic coordinates;
- rectangular coordinates of a point on the surface of body from its geodetic latitude and longitude;
- rectangular coordinates of a point from its right ascension, declination and distance to origin.

Simple operations on vectors. ***page 24***
- addition, subtraction, cross product and dot product of two vectors;
- product of vector and scalar;
- negation of given vector;
- copying of given vector;
- indication if a given vector is the zero vector;
- angular separation between two vectors;
- distance between two vectors;
- magnitude of vector;
- unit vector along with given vector;
- unit vector along with cross product of two vectors;
- magnitude and unit vector along with given vector.

Projections, linear combinations and rotation of vectors. ***page 25***
- vector’s component rectangular to another vector;
- projection of vector into another vector;
- rotation of vector by given angle about axis given by another vector;

- rotation of vector by given angles about reference axis X, Y or Z;
- point on a line nearest to a given point;
- orthogonal projection of a vector onto a given plane;
- inverted vector projection onto a plane;
- linear combination of two vectors;
- linear combination of three vectors.

- product of two matrixes;
- product of a matrix and the transpose of another matrix;
- product of a matrix and a vector;
- product of the transpose of a matrix and another matrix;
- product of the transpose of a matrix and a vector;
- product of the transpose of a vector, a matrix and another vector;
- transpose, determinant and trace of a matrix;
- copying of a matrix;

- plane from normal vector and distance from origin;
- plane from point and normal vector;
- plane from point and two vectors;
- normal vector and distance from origin for given plane;
- point and normal vector for given plane;
- point and two vectors for given plane;
- intersection of ray and plane.

- ellipse from center and two generating vectors;
- center and axis for given ellipse;
- axis of ellipse from two generating vectors;
- intersection of ellipse and plane;
- point on given ellipse and nearest to given point;
- projection of ellipse onto plane.

- point on a given ellipsoid nearest to a given point;
- intersection of a ray and an ellipsoid;
- normal vector for a given point on ellipsoid surface;
- limb of ellipsoid surface;
- point on a given ellipsoid nearest to a given line;
- intersection of ellipsoid and plane.

- calculation of matrix rotating a vector about a specified reference axis (X, Y or Z);
- rotation of a matrix about a specified reference axis (X, Y or Z);
- calculation of matrix rotating vectors to a reference frame, the principal axes of which are specified by two given vectors;
- indication if a given matrix is rotation matrix;
- calculation of a matrix from Euler angles;
- calculation of Euler angles from a matrix.

- calculation of spacecraft position and velocity for given time and set of orbital elements;
- calculation of orbital elements for given position and velocity of spacecraft and gravitational parameter of planet.

- calculation of sub-observer point on the surface of a body;
- calculation of illumination angles at a given point on the surface of a body.

- calculation of the intercept point on the surface of a target by a direction from an observer;

Solar system body names and NAIF ID codes.

Planet Barycenters

Positive IDs from 0 to 10 are assigned to planet barycenters, solar system barycenter and Sun.

`0`	Solar system barycenter
`1`	Mercury barycenter
`2`	Venus barycenter
`3`	Earth barycenter
`4`	Mars barycenter
`5`	Jupiter barycenter
`6`	Saturn barycenter
`7`	Uranus barycenter
`8`	Neptune barycenter
`9`	Pluto barycenter
`10`	Sun

Planet Mass Centers

The code for each planet center of mass is computed by adding 99 to the code of the planet's barycenter multiplied by 100.

`199`	Mercury (equivalent to 1)
`299`	Venus (equivalent to 2)
`399`	Earth
`499`	Mars (equivalent to 4)
`599`	Jupiter
`699`	Saturn
`799`	Uranus
`899`	Neptune
`999`	Pluto

Satellites

The code for a satellite is computed by adding its IAU designation to the code of its planet barycenter multiplied by 100.

`301`	Moon

`401`	Phobos
`402`	Deimos

`501`	Io
`502`	Europe
`503`	Ganimede
`etc.`	etc.

Spacecraft and instrument names and NAIF ID codes.

Spacecraft's, scientific instruments

Negative codes are used for spacecraft. So, "MGS" has ID **`-94`**, "Galileo orbiter" ID is **`-77`**, "Stardust" ID is **`-29`**, "Cassini" ID is **`-82`**, etc. The ID for a particular spacecraft is assigned by NAIF/JPL. These are based on NASA DSN designations. See the document ***NAIF IDs Required Reading*** for a complete list.

The code of a scientific instrument or instrument platform is normally computed by adding its ID code assigned by project staff to the code of the spacecraft multiplied by 1000. For example, the code of the first instrument platform of the spacecraft with ID **`-23`** would be **`-23001`**.

Built-in and User Defined names/IDs

Although ID-name mapping for many past and current spacecraft is built into the system, the SPICE toolkit provides a mechanism that allows defining additional ID-name mappings. This is done by specifying ID-name pairs in a text file, following the text kernel file format, using the keywords **`NAIF_BODY_CODE`** and **`NAIF_BODY_NAME`**:

```
\begindata
   NAIF_BODY_CODE += ( id_1 )
   NAIF_BODY_CODE += ( 'name_1' )

   NAIF_BODY_CODE += ( id_2 )
   NAIF_BODY_CODE += ( 'name_2' )
...
\begintext
```

When a file defining ID-name pairs is loaded into a SPICE-based application, this mapping becomes available to all SPICE routines. Note that using the **`'+='`** assignment is required in order to "append" new pairs to those, which are already loaded.

ID-name mappings for a particular project are usually stored in the Frames Kernel (FK) or Instrument Kernels (IKs) for that spacecraft but could be provided in any SPICE text kernel. Sometimes a "mission kernel" is used to provide ID-name mappings as well as other information used in SPICE-based applications. For example, this fragment from FIDO rover FK file defines FIDO rover and its instrument ID-name mapping:

```
\begindata

   NAIF_BODY_NAME += ( 'FIDO_ROVER'                )
   NAIF_BODY_CODE += ( -771000                     )

   NAIF_BODY_NAME += ( 'FIDO_FRONT_HAZCAM_LEFT'    )
   NAIF_BODY_CODE += ( -771011                     )

   NAIF_BODY_NAME += ( 'FIDO_FRONT_HAZCAM_RIGHT'   )
   NAIF_BODY_CODE += ( -771012                     )

   NAIF_BODY_NAME += ( 'FIDO_REAR_HAZCAM_LEFT'     )
   NAIF_BODY_CODE += ( -771021                     )
...
\begintext
```

Inertial Reference Frames

Codes and names of standard inertial reference frames.

The names and integer codes of standard inertial reference frames supported by the SPICE Toolkit are given in the table below. They are used as arguments in some SPICELIB routines.

Code	Name	Description
`1`	`J2000`	Earth mean equator, dynamical equinox of J2000
`2`	`B1950`	Earth mean equator, dynamical equinox of B1950
`3`	`FK4`	Fundamental Catalog (4)
`4`	`DE-118`	JPL Developmental Ephemeris (118)
`5`	`DE-96`	JPL Developmental Ephemeris (96)
`6`	`DE-102`	JPL Developmental Ephemeris (102)
`7`	`DE-108`	JPL Developmental Ephemeris (108)
`8`	`DE-111`	JPL Developmental Ephemeris (111)
`9`	`DE-114`	JPL Developmental Ephemeris (114)
`10`	`DE-122`	JPL Developmental Ephemeris (122)
`11`	`DE-125`	JPL Developmental Ephemeris (125)
`12`	`DE-130`	JPL Developmental Ephemeris (130)
`13`	`GALACTIC`	Galactic System II
`14`	`DE-200`	JPL Developmental Ephemeris (200)
`15`	`DE-202`	JPL Developmental Ephemeris (202)
`16`	`MARSIAU`	Mars Mean Equator and IAU vector of J2000
`17`	`ECLIPJ2000`	Ecliptic coordinates based upon the J2000 frame
`18`	`ECLIPB1950`	Ecliptic coordinates based upon the B1950 frame
`19`	`DE-140`	JPL Developmental Ephemeris (140)
`20`	`DE-142`	JPL Developmental Ephemeris (142)
`21`	`DE-143`	JPL Developmental Ephemeris (143)
`22`	`DE-145`	JPL Developmental Ephemeris (145)

All **`DE-2XX`** and **`DE-4XX`** frames are, by definition, **`J2000`** frames, unique identifiers for these are not needed.

Names of body-fixed rotating frames.

Body-fixed rotating frames for all solar system bodies are defined within the SPICE system. The name of such a frame for a particular body is constructed by adding the prefix **`'IAU_'`** to the body name. For example, the name of the Mars body-fixed rotating frame is **`'IAU_MARS'`**.

Other frames used within SPICE system.

Other types of frames for spacecraft, instrument platforms and instruments can be defined using the SPICE system “frames” mechanism. See Toolkit document “Frames Required Reading”.

Loading and Unloading SPICE Kernels

Routines

`FURNSH` loads a single SPICE kernel file `FNAME` or multiple SPICE kernels provided in a list in a text kernel file `FNAME`.

```
SUBROUTINE FURNSH( FNAME )
CHARACTER*(*)     FNAME
```

`UNLOAD` unloads a single SPICE kernel file `FNAME` or multiple SPICE kernels provided in a list in a text kernel file `FNAME`.

```
SUBROUTINE UNLOAD( FNAME )
CHARACTER*(*)     UTC
```

Kernel List File Format

The `FURNSH` routine provides a mechanism for loading multiple kernels with a single call. In order to do that, the kernel files to be loaded must be listed in the value field of the `KERNELS_TO_LOAD` keyword in a file that follows the text kernel file format. Then, the name such a meta-kernel file should be provided as input to `FURNSH`.

```
\begindata
   KERNELS_TO_LOAD = (
                       'kernel_file_name'
                       'kernel_file_name'
                       'kernel_file_name'
                       )
\begintext
```

Example

The first code fragment individually loads the LSK file `'/kernels/gen/lsk/naif0007.tls'` and the SCLK file `'/kernels/mgs/sclk/mgs.tsc'` needed to perform MGS time conversions.

```
LSKFN = '/kernels/gen/lsk/naif0007.tls'
SCLKFN = '/kernels/mgs/sclk/mgs.tsc'
CALL FURNSH( LSKFN )......
CALL FURNSH( SCLKFN )......
```

The second code fragment loads multiple kernels listed in the meta-kernel file:

```
\begindata
   KERNELS_TO_LOAD = (
                       '/kernels/generic/lsk/naif0007.tls'
                       '/kernels/generic/pck/pck00006.tpc'
                       '/kernels/generic/spk/de405.bsp'
                       '/kernels/mgs/sclk/mgs.tsc'
                       '/kernels/mgs/fk/mgs.fk'
                       '/kernels/mgs/spk/mgs_map1.bsp'
                       '/kernels/mgs/ck/mgs_map1.bc'
                       )
\begintext
```

with a single call to `FURNSH` routine:

```
LISTFN = 'mgs_kernels.furnsh'
CALL FURNSH( LISTFN )......
```

The third code fragment unloads the earlier loaded SPK file `'my_spk.bsp'` using `UNLOAD`:

```
SPKFN = 'my_spk.bsp'
CALL UNLOAD( SPKFN )......
```

Universal and Ephemeris times

Routines

FURNSH loads LSK kernel file **`FNAME`** containing values of constants and leap seconds required for UTC – ET correspondence calculation.

```
SUBROUTINE FURNSH( FNAME )
CHARACTER*(*)     FNAME
```

STR2ET given a **`STRING`** representing a time, calculates the corresponding ephemeris time **`ET`**.

```
SUBROUTINE STR2ET ( STRING, ET )
CHARACTER*(*)     STRING
DOUBLE PRECISION  ET
```

TIMOUT given an ephemeris time **`ET`**, calculates a time string **`STRING`** in a user-specified format and system. The **`PICTUR`** parameter is a string that gives a "picture" of the time format.

```
SUBROUTINE TIMOUT ( ET, PICTUR, STRING )
DOUBLE PRECISION  ET
CHARACTER*(*)     PICTUR
CHARACTER*(*)     STRING
```

UTC2ET is an older, less flexible than **`STR2ET,`** routine which given a Universal Time **`UTC`**, calculates the corresponding ephemeris time **`ET`**.

```
SUBROUTINE UTC2ET( UTC, ET )
CHARACTER*(*)     UTC
DOUBLE PRECISION  ET
```

ET2UTC is an older, less flexible than **`TIMOUT,`** routine which given an Ephemeris Time **`ET`**, calculates the corresponding Universal Coordinated Time, **`UTC`**. The **`FORMAT`** parameter defines the format of UTC (can be **`'C'`** for calendar, **`'D'`** for day of the year, **`'J'`** for Julian date UTC; **`'ISOC'`** for ISO calendar format, **`'ISOD'`** for ISO day of year format). The **`PREC`** parameter defines number of digits after decimal point in UTC seconds.

```
SUBROUTINE UTC2ET( ET, FORMAT, PREC, UTC )
DOUBLE PRECISION  ET
CHARACTER*(*)     FORMAT
INTEGER           PREC
CHARACTER*(*)     UTC
```

Example

This fragment of code loads an LSK file (usually it's done once at the beginning of the program), calculates ET for a given UTC, adds 2 hours and converts this ET back to UTC in ISO date format.

```
CALL FURNSH( '/kernels/generic/sclk/naif0007.tls" )
...
CALL STR2ET ( '1997 Jan 17 17:44:42.271', ET )
ET = ET + 7200
CALL TIMOUT ( ET, 'YYYY-MM-DDTHR:MN:SC.###', STRING )
```

UTC and ET formats

Universal Time UTC is a string and can appear in one of the following formats:

ISO format, for example

```
1986-01-18T12:19:52.18
1995-008T18:28:12
```

Day of the year, for example

```
1993-321/12:28:28.287
1992 183// 12 18 19
```

Calendar date, for example

```
1986 JAN 9 03:12:59.22451
Tue Aug 6 11:10:57  1996
```

Julian date, for example

```
jd 28272.291
2451515.2981 (JD)
```

Ephemeris Time ET is the number of ephemeris seconds past Julian date J2000 (JD = 2451545.0 corresponds to 12:00:00 January 1, 2000 TDB).

Spacecraft On-board Time (SCLK)

Routines

FURNSH loads SCLK kernel file `FNAME` containing values required for SCLK string format interpretation and SCLK-to-ephemeris time (ET) correspondence calculation.

```
SUBROUTINE FURNSH( FNAME )
CHARACTER*(*)     FNAME
```

SCENCD converts character representation of SCLK `CLKSTR` to its double precision encoding `SCLKDP` for the spacecraft with integer code `SC`.

SCDECD makes opposite conversion.

```
SUBROUTINE SCENCD( SC, CLKSTR, SCLKDP )
SUBROUTINE SCDECD( SC, SCLKDP, CLKSTR )
INTEGER           SC
CHARACTER*(*)     CLKSTR
DOUBLE PRECISION  SCLKDP
```

SCE2C calculates for ephemeris time `ET` the corresponding double precision continuous encoding of `SCLKDP` for the spacecraft with ID `SC`.

SCT2E makes opposite conversion.

```
SUBROUTINE SCE2C( SC, ET, SCLKDP )
SUBROUTINE SCT2E( SC, SCLKDP, ET )
INTEGER           SC
DOUBLE PRECISION  ET
DOUBLE PRECISION  SCLKDP
```

SCE2S calculates for ephemeris time `ET` the corresponding `CLKSTR` represented as a character string for the spacecraft with integer code `SC`.

SCS2E makes opposite conversion.

```
SUBROUTINE SCE2S( SC, ET, CLKSTR )
SUBROUTINE SCS2E( SC, CLKSTR, ET )
INTEGER           SC
DOUBLE PRECISION  ET
CHARACTER*(*)     CLKSTR
```

Example

This fragment of code loads a SCLK file for spacecraft with ID `-23` (it's done once at the beginning of the program), calculates for a given ET the corresponding double precision encoding of SCLK and converts it to character representation.

```
...
CALL FURNSH( '/kernels/sc23/sclk/spcrft23.tsc' )
...
CALL SCE2C ( -23, ET, SCLKDP )
CALL SCDECD( -23, SCLKDP, CLKSTR )
```

SCLK formats

String representation: SCLK is represented as string such as `'2/123.23.59.59.255'` consisting of two parts. The first part `'2/'` is the partition number, the second part `'123.23.59.59.255'` is the SCLK time in this partition (the dots are the delimiters separating days, hours, minutes, seconds, 1/256 of seconds). Different spacecraft have different set of field. For example, MGS, Cassini and Stardust SCLKs have two fields – seconds and 1/256 of seconds, NEAR SCLK has one field – milliseconds, etc.

"Encoded" double precision representation: SCLK time is represented by a double precision number containing the number of ticks that the on-board timer has counted from the beginning of the mission. A tick is the shortest time increment expressible by this clock (for example for MGS it is 1/256 of second).

Constants and matrixes of planets and satellites (PCK).

Routines

FURNSH

loads text PCK kernel file `FNAME` containing constants for Solar system bodies or binary PCK kernel file `FNAME` containing orientation data for one or more solar system bodies.

```
SUBROUTINE FURNSH( FNAME )
CHARACTER*(*)      FNAME
```

PXFORM

calculates the matrix `XFORM` used to rotate position vectors from inertial frame with name `FROM` to solar system planet or satellite body-fixed frame with name `TO` at ephemeris time `ET`. The name of a planet of satellite body-fixed frame, orientation for which is determined using rotation constants stored in a generic text PcK file, is constructed by adding the prefix `'IAU_'` to the body name (for example, the name of the Mars IAU body-fixed rotating frame is `'IAU_MARS'`). The name of an Earth body-fixed frame, orientation of which is determined using high-precision Earth rotation data provided in a binary PCB file, is `'ITRF93'`.

```
SUBROUTINE PXFORM ( FROM, TO, ET, XFORM )
CHARACTER*(*)     FROM
CHARACTER*(*)     TO
DOUBLE PRECISION  ET
DOUBLE PRECISION  XFORM ( 3, 3 )
```

SXFORM

calculates the matrix `XFORM` used to rotate state vectors (position and velocity) from inertial frame with name `FROM` to solar system planet or satellite body-fixed frame with name `TO` at ephemeris time `ET`. The name of a planet of satellite body-fixed frame, orientation for which is determined using rotation constants stored in a generic text PcK file, is constructed by adding the prefix `'IAU_'` to the body name (for example, the name of the Mars IAU body-fixed rotating frame is `'IAU_MARS'`). The name of an Earth body-fixed frame, orientation of which is determined using high-precision Earth rotation data provided in a binary PCB file, is `'ITRF93'`.

```
SUBROUTINE SXFORM ( FROM, TO, ET, XFORM )
CHARACTER*(*)     FROM
CHARACTER*(*)     TO
DOUBLE PRECISION  ET
DOUBLE PRECISION  XFORM ( 6, 6 )
```

BODVRD

returns the vector `VALUES` (and its dimension `DIM`) containing up to `MAXN` value(s) for the physical parameter named `ITEM` for the body with name `BODYNM`. As an example, to retrieve the axes of the ellipsoidal model of a planet `ITEM` is set to `'RADII'`. To retrieve planet nutation precession angles, set `ITEM` to `'NUT_PREC_ANGLES'`, etc.

```
SUBROUTINE BODVRD( BODYNM, ITEM, MAXN, DIM, VALUES )
CHARACTER*(*)     BODYNM
CHARACTER*(*)     ITEM
INTEGER           MAXN
INTEGER           DIM
DOUBLE PRECISION  VALUES (*)
```

Example

This fragment of code loads a text PCK file, calculates the matrix, which rotates vectors from the inertial frame `'B1950'` to the IAU body-fixed frame for Mars, and retrieves the lengths of the three axes defining the Mars ellipsoid.

```
CALL FURNSH( '/kernels/generic/pck/pck00006.tpc' )
...
CALL PXFORM( 'B1950', 'IAU_MARS', ET, MARSMT )
CALL BODVRD( 'MARS', 'RADII', 3, DIM, MARSRD )
```

Frame Transformations: Inertial, PCK-based and User-defined frames.

Routines

FURNSH loads Frame Definitions kernel file `FNAME` containing frames definitions for a particular spacecraft, instrument or other structure of interest.

```
SUBROUTINE FURNSH( FNAME )
CHARACTER*(*)     FNAME
```

PXFORM calculates the matrix `XFORM` used to rotate position vectors from one frame with name `FROM` to another frame with name `TO` at ephemeris time `ET`.

```
SUBROUTINE PXFORM ( FROM, TO, ET, XFORM )
CHARACTER*(*)     FROM
CHARACTER*(*)     TO
DOUBLE PRECISION  ET
DOUBLE PRECISION  XFORM ( 3, 3 )
```

SXFORM calculates the matrix `XFORM` used to rotate state vectors (position and velocity) from one frame with name `FROM` to another frame with name `TO` at ephemeris time `ET`.

```
SUBROUTINE SXFORM ( FROM, TO, ET, XFORM )
CHARACTER*(*)     FROM
CHARACTER*(*)     TO
DOUBLE PRECISION  ET
DOUBLE PRECISION  XFORM ( 6, 6 )
```

Inertial and PCK-based frame naming convention

The inertial frame naming convention is described in the *Inertial Reference Frames* section of this document. The second category of frames supported in SPICE is the PCK-based set of frames (IAU body-fixed rotating frames). The naming convention for these frames is `'IAU_[BODY_NAME]'`, where `'[BODY_NAME]'` is the name of the body. For example, to refer to the Mars body-fixed rotating frame use `'IAU_MARS'` in an `SXFORM` or `PXFORM` call

User-defined frame naming convention

Another category of frames supported in SPICE is user-defined frames which can be CK-based or fixed offset. The SPICE system recognizes these frames only when a frames kernel file containing definitions for such frames is loaded into the kernel pool. These frames can be given any name except those, which belong to the standard SPICE inertial set, and any PCK-based frames already defined in the SPICE system. To avoid possible interference when frames for multiple spacecraft and instruments are loaded into SPICE simultaneously, NAIF recommends including the abbreviated spacecraft name and instrument name in the prefix of any user-defined frame name. Refer to the frame kernel for a particular mission for a complete list of user-defined frames for that mission.

Example

This fragment of code loads a meta-kernel file, which contains this `KERNELS_TO_LOAD` assignment:

```
\begindata
   KERNELS_TO_LOAD = ( '/kernels/mgs/frames/mgs.tf',
                 '/kernels/generic/lsk/naif0007.tls',
                 '/kernels/mgs/sclk/mgs.tsc',
                 '/kernels/mgs/ck/mgs_spice_c_kernel_1998-339.bc',
                 '/kernels/mgs/ck/mgs_solar_array_1998-339.bc' )
\begintext
```

pointing to MGS Frames kernel file, generic SPICE LSK file, MGS SCLK file and MGS spacecraft and solar array orientation CK files and calculates the matrix which rotates vectors from the Mars body-fixed rotating frame `'IAU_MARS'` to the MGS magnetometer (MAG) +Y sensor frame `'MGS_MAG_+Y_SENSOR'`:

```
   CALL FURNSH( 'mgs_kernels.list' )
   CALL PXFORM( 'IAU_MARS', 'MGS_MAG_+Y_SENSOR', ET, XMAT )
```

Planet and Spacecraft positions (SPK).

Routines

FURNSH loads an SPK kernel file **FNAME** containing trajectory data for one or more ephemeris bodies (planet, satellite, spacecraft, etc.) for some interval of time, and returns the file handle **HANDLE** for this file.

```
SUBROUTINE FURNSH( FNAME  )
CHARACTER*(*)     FNAME
```

SPKEZR calculates the state vector (position and velocity) **STATE** of one body ("target") with respect to another body ("observer") at ephemeris time **ET**. Both bodies are specified by their names - **TARGNM** for "target" and **OBSNM** for "observer". The state vector is calculated in the requested reference frame with name **FRAME** from the list of frames supported within the SPICE system. In accordance with the correction parameter **ABERR** the state vector can be calculated as apparent (**ABERR='LT+S' or 'CN+S'**), true (**'LT' or 'CN'**), or geometric (**'NONE'**). This routine also returns one-way light time from "target" to "observer" **LT**.

```
SUBROUTINE SPKEZR( TARGNM, ET, FRAME, ABERR, OBSNM, STATE, LT )
CHARACTER*(*)     TARGNM
DOUBLE PRECISION  ET
CHARACTER*(*)     FRAME
CHARACTER*(*)     ABERR
CHARACTER*(*)     OBSNM
DOUBLE PRECISION  STATE (6)
DOUBLE PRECISION  LT
```

SPKPOS performs the same calculation as **SPKEZR** but returns the position vector instead of the state vector.

```
SUBROUTINE SPKPOS( TARGNM, ET, FRAME, ABERR, OBSNM, PTARG, LT )
CHARACTER*(*)     TARGNM
DOUBLE PRECISION  ET
CHARACTER*(*)     FRAME
CHARACTER*(*)     ABERR
CHARACTER*(*)     OBSNM
DOUBLE PRECISION  PTARG (3)
DOUBLE PRECISION  LT
```

UNLOAD unloads previously loaded SPK having file name **FNAME**.

```
SUBROUTINE UNLOAD( FNAME )
CHARACTER*(*)     FNAME
```

Example

This fragment of code loads two SPK files (the first contains ephemeris data for Solar system bodies, the second contains trajectory data for the spacecraft with ID **-23**) having some time coverage in common. It also loads a PCK file to provide data for transformation from inertial to the Mars body-fixed rotating frame. Then it calculates geometric states of the Sun and spacecraft with respect to the Mars center in the Mars body-fixed rotating frame "IAU_MARS". The loading of SPK and PCK files is normally done only once at the beginning of the program, while the computation of state vectors is usually repeated for many instants of time. Note the spacecraft ID formatted as a string in the second call to SPKEZR; this mechanism allows using a body ID in place of a name if an object name isn't recognized by SPICE toolkit.

```
   ...
   CALL FURNSH ( '/kernels/generic/pck/pck00006.tpc'      )
   CALL FURNSH ( '/kernels/generic/spk/de200.bsp'         )
   CALL FURNSH ( '/kernels/sc23/spk/sc23_orbit142.bsp'    )
   ...
   CALL SPKEZR ( 'SUN', ET, 'IAU_MARS', 'NONE', 'MARS', SUNST,  LT )
   CALL SPKEZR ( '-23', ET, 'IAU_MARS', 'NONE', 'MARS', SC23ST, LT )
```

Attitude of spacecraft and instrument platforms (CK).

Routines

FURNSH

loads a CK kernel file **FNAME** containing attitude data for one or more spacecraft or instrument platforms and returns the integer file handle **HANDLE** for this file.

```
SUBROUTINE FURNSH( FNAME )
CHARACTER*(*)     FNAME
```

PXFORM

calculates the matrix **XFORM** used to rotate position vectors from one frame with name **FROM** to another frame with name **TO**, either of which can be a CK-based frame, at ephemeris time **ET**.

```
SUBROUTINE PXFORM ( FROM, TO, ET, XFORM )
CHARACTER*(*)     FROM
CHARACTER*(*)     TO
DOUBLE PRECISION  ET
DOUBLE PRECISION  XFORM ( 3, 3 )
```

SXFORM

calculates the matrix **XFORM** used to rotate state vectors from one frame with name **FROM** to another frame with name **TO,** either of which can be a CK-based frame, at ephemeris time **ET**.

```
SUBROUTINE SXFORM ( FROM, TO, ET, XFORM )
CHARACTER*(*)     FROM
CHARACTER*(*)     TO
DOUBLE PRECISION  ET
DOUBLE PRECISION  XFORM ( 6, 6 )
```

CKGPAV

lower-level CK routine that calculates the transformation matrix **CMAT** and the angular velocity **AV** of rotation of the platform-fixed reference frame with respect to the specified reference frame named **REF** for the instrument platform having ID **INS** at the time **SCLK** that is the DP encoding of SCLK. If pointing data in the loaded file is continuous, then the matrix and the angular velocity will be returned at exactly the requested SCLK and **SOUT** will be equal to **SCLK**. If pointing data in the loaded file is discrete then the matrix and the angular velocity will be calculated for the time that is closest to the requested SCLK and belongs in the interval ±**TOL** from it. This time will be returned in **SCLKOUT**. The flag **FND** will be **.TRUE.** if it was possible to calculate **CMAT** and **AV**, otherwise it will be **.FALSE.**

CKGP

similar to **CKGPAV** but calculates only the transformation matrix **CMAT.**

```
SUBROUTINE CKGPAV( INS, SCLK, TOL, REF, CMAT, AV, SOUT, FND )
SUBROUTINE CKGP ( INS, SCLK, TOL, REF, CMAT,     SOUT, FND )
INTEGER           INS
DOUBLE PRECISION  SCLK
DOUBLE PRECISION  TOL
CHARACTER*(*)     REF
DOUBLE PRECISION  CMAT (3,3)
DOUBLE PRECISION  AV   (3)
DOUBLE PRECISION  SOUT
BOOLEAN           FND
```

UNLOAD

unloads the previously loaded CK having file name **FNAME**.

```
SUBROUTINE UNLOAD( FNAME )
CHARACTER*(*)     FNAME
```

Example

This fragment of code loads a CK file containing pointing data for the MGS spacecraft, calculates a transformation matrix used to rotate vectors from the inertial frame **'J2000'** to the MGS spacecraft frame **'MGS_SPACECRAFT'**, and performs this rotation on the vector **X**.

```
CALL FURNSH( '/kernels/mgs/ck/mgs_map1.bc' )
......
CALL PXFORM( 'J2000', 'MGS_SPACECRAFT', ET, CMAT )
CALL MXV  ( CMAT, X, XOUT )
```

Scientific Instrument Parameters (IK).

Routines

FURNSH loads an IK kernel file named **FNAME** containing field-of-view and other parameters for a particular scientific instrument.

```
SUBROUTINE FURNSH( FNAME )
CHARACTER*(*)     FNAME
```

GDPOOL
GIPOOL
GCPOOL returns in the double precision array **DVALS** (subroutine **GDPOOL**), or in the integer array **IVALS** (subroutine **GIPOOL**) or in the character array **CVALS** (subroutine **GCPOOL**) **N** elements (**N** is less than or equal to **ROOM**) starting with the element indexed **START** of the value for the instrument parameter with name **NAME**. Flag **FOUND** becomes **.TRUE.** if the requested parameter (and its values) was found among the loaded parameters.

```
SUBROUTINE GDPOOL( NAME, START, ROOM, N, DVALS, FOUND )
SUBROUTINE GIPOOL( NAME, START, ROOM, N, IVALS, FOUND )
SUBROUTINE GCPOOL( NAME, START, ROOM, N, CVALS, FOUND )
CHARACTER*(*)     NAME
INTEGER           START
INTEGER           ROOM
INTEGER           N
DOUBLE PRECISION  DVALS (*)
INTEGER           IVALS (*)
CHARACTER*(*)     CVALS (*)
BOOLEAN           FOUND
```

GETFOV returns field-of-view (FOV) configuration including shape **SHAPE**, the name of the frame **FRAME** in which the FOV is defined, the boresight vector **BSIGHT**, and the array **BOUNDS** containing **N** FOV boundary vectors for the instrument with the NAIF ID **INSTID**. (The number of returned boundary vectors **N** is less than or equal to the room **ROOM** available in the array **BOUNDS**.)

```
SUBROUTINE GETFOV( INSTID, ROOM, SHAPE, FRAME, BSIGHT, N, BOUNDS)
INTEGER           INSTID
INTEGER           ROOM
CHARACTER*(*)     SHAPE
CHARACTER*(*)     FRAME
DOUBLE PRECISION  BSIGHT (3)
INTEGER           N
DOUBLE PRECISION  BOUNDS (3,*)
```

Instrument parameters naming convention

The names of instrument parameters are defined in accordance with the following scheme:
INS-nnnnn_<item name>,
where **INS** shows that this parameter belongs to a scientific instrument, **-nnnnn** is the SPICE ID of this instrument and **<item name>** is the name of the specific parameter. For example, the pixel size for the instrument with ID code –**23036** may be stored in the keyword **INS-23036_PIXEL_SIZE**. In order to use **GETFOV**, the following set keywords defining FOV shape, boresight boundary vectors and reference frame must be provided in the instrument IK file (**-nnnnn** is the SPICE ID of the instrument):

```
INS-nnnnn_FOV_FRAME, INS-nnnnn_FOV_BOUNDARY_CORNERS,
INS-nnnnn_FOV_SHAPE, INS-nnnnn_FOV_BORESIGHT
```

Example

This fragment of code loads an IK file containing parameters for the instrument with code **-23036** and returns parameters of its rectangular FOV and the value of its shutter delay.

```
CALL FURNSH( '/kernels/sc23/ik/ins23036.ti' )
CALL GETFOV( -23036, 4, SHAPE, FRAME, BSIGHT, N, BOUNDS)
CALL GDPOOL( 'INS-23036_SHUTTER_DELAY', 1, 1, N, SDELAY, FOUND )
```

Spacecraft Event Information (Data Base Kernel)

Routines

FURNSH loads an EK kernel file named **`FNAME`**, making it accessible to the EK readers.

```
SUBROUTINE FURNSH ( FNAME )
CHARACTER*(*)      FNAME
```

EKFIND finds E-kernel data that satisfy a set of constraints specified in a query string **`QUERY`** and returns the number of found EK data records (rows) **`NMROWS`**. The flag **`ERROR`** will be **`.FALSE.`** if a specified query string didn't contain any errors, otherwise it will be **`.TRUE.`** The error diagnostics string **`ERRMSG`** will contain a description of an error if such was detected (**`ERROR=.TRUE.`**) or it will be set blank if no errors were found in the query string .

```
SUBROUTINE EKFIND ( QUERY, NMROWS, ERROR, ERRMSG )
CHARACTER*(*)      QUERY
INTEGER            NMROWS
LOGICAL            ERROR
CHARACTER*(*)      ERRMSG
```

EKGD
EKGC
EKGI

returns a double precision element **`DDATA`** (subroutine **`EKGD`**), character element **`CDATA`** (subroutine **`EKGC`**) or integer element **`IDATA`** (subroutine **`EKGI`**) from a data record (row) specified by its index **`ROW`** in the list of data records that satisfy the selection criteria submitted in the last call to EKFIND. The column to fetch data from is specified by an index **`SELIDX`** in the SELECT clause of a query string that was used with EKFIND to define that selection criteria, and the index of the element within the column entry is specified by **`ELMENT`** (**`ELMENT`** is always 1 for scalar columns and can be from 1 to the size of the column's entry for vector columns). The flag **`NULL`** will be **`.TRUE.`** if the specified data entry is null, otherwise it will be **`.FALSE.`** The flag **`FOUND`** will be **`.TRUE.`** if the specified element was found, otherwise it will be **`.FALSE.`**

```
SUBROUTINE EKGD ( SELIDX, ROW, ELMENT, DDATA, NULL, FOUND )
SUBROUTINE EKGC ( SELIDX, ROW, ELMENT, CDATA, NULL, FOUND )
SUBROUTINE EKGI ( SELIDX, ROW, ELMENT, IDATA, NULL, FOUND )
INTEGER            SELIDX
INTEGER            ROW
INTEGER            ELMENT
DOUBLE PRECISION   DDATA
CHARACTER*(*)      CDATA
INTEGER            IDATA
LOGICAL            NULL
LOGICAL            FOUND
```

UNLOAD unloads previously loaded EK having file name **`FNAME`**.

```
SUBROUTINE UNLOAD( FNAME )
CHARACTER*(*)      FNAME
```

EKFIND Query Syntax (Single Table Only)

The query consists of four clauses, the third and fourth of which are optional. The general form of a query involving a single table is

```
SELECT <column name> [, <column name> ...]
FROM <table name>
[WHERE <constraint expr.> [AND/OR <constraint expr.> ...]]
[ORDER BY <column name> [<order>] [, <column name> [<order>] ...]]
```

where brackets indicate optional items. The general form of the constraint expression is

```
<column name> <operator> <RHS symbol>
```

where **`<RHS symbol>`** is a column name or a literal value and **`<operator>`** is any of **`EQ`**, **`GE`**, **`GT`**, **`LE`**, **`LIKE`**, **`LT`**, **`NE`**, **`NOT LIKE`**, **`<`**, **`<=`**, **`=`**, **`>`**, **`>=`**, **`!=`** and **`<>`**. The operators **`BETWEEN`** and **`NOT BETWEEN`** are also supported.

Example

This fragment of code loads an EK file containing a table called **`EVENTS`** containing a time-type column **`EVENT_TIME`**, an integer column **`EVENT_ID`**, a double precision column **`DURATION`** and a character column **`DESC`**. The data entries in the first three columns have scalar values, the data entries in the fourth column — **`DESC`** — are variable size arrays of up to 80 character long strings. The code then searches for events within a specified time interval and fetches data from all records that were found.

```
CALL FURNSH( FNAME )
......
CALL PROMPT( 'Enter start UTC time>', BEGUTC )
CALL PROMPT( 'Enter end UTC time>',   ENDUTC )
QUERY = 'SELECT EVENT_TIME, EVENT_ID, DURATION, DESC ' //
        'FROM EVENTS ' //
        'WHERE TIME BETWEEN ' // BEGUTC // ' AND ' // ENDUTC //
        'ORDER BY TIME'
CALL EKFIND ( QUERY, NMROWS, ERROR, ERRMSG )
IF ( .NOT. ERROR ) THEN
   IF ( NMROWS .GT. 0 ) THEN
      DO ROW = 1, NMROWS

         CALL EKGD ( 1, ROW, 1, ET, NULL, FOUND )
         IF ( .NOT. NULL ) THEN
            CALL TIMOUT( ET, 'YYYY-MM-DDTHR:MN:SC.###', UTC(ROW) )
         ELSE
            UTC(ROW) = ' '
         END IF

         CALL EKGI ( 2, ROW, 1, EVNTID(ROW), NULL, FOUND )
         IF ( NULL ) THEN
            EVNTID(ROW) = 0
         END IF

         CALL EKGD ( 3, ROW, 1, DURATN(ROW), NULL, FOUND )
         IF ( NULL ) THEN
            DURATN(ROW) = 0.D0
         END IF

         N = 1
         CALL EKGC ( 4, ROW, N, DESCRP(ROW,N), NULL, FOUND )
         IF ( .NOT. NULL .AND. FOUND ) THEN
            DO WHILE ( FOUND )
               N = N + 1
               CALL EKGC ( 4, ROW, N, DESCRP(ROW,N), NULL, FOUND )
            END DO
         ELSE
            DESCRP(ROW,1) = ' '
         END IF

      END DO
   ELSE
      WRITE( *,* ) 'No records satisfying query (' // QUERY //
                   ') were found.'
   END IF
ELSE
   WRITE( *,* ) 'Bad query string: ' // ERRMSG
END IF
```

Physical and Mathematical constants

Routines

HALFPI	returns value of π/2, calculated as ARCCOS(-1.D0)/2.D0. `DOUBLE PRECISION FUNCTION HALFPI ()`
PI	returns value of π, calculated as ARCCOS(-1.D0). `DOUBLE PRECISION FUNCTION PI ()`
TWOPI	returns value of 2*π, calculated as 2.D0*ARCCOS(-1.D0). `DOUBLE PRECISION FUNCTION TWOPI ()`
DPR	returns number of degrees per radian, calculated as 180.D0/ARCCOS(-1.D0). `DOUBLE PRECISION FUNCTION DPR ()`
RPD	returns number of radians per degree, calculated as ARCCOS(-1.D0)/180.D0. `DOUBLE PRECISION FUNCTION RPD ()`
SPD	returns number of seconds per day (86400). `DOUBLE PRECISION FUNCTION SPD ()`
CLIGHT	returns IAU official value of light speed in vacuum (299792.458 km/sec). `DOUBLE PRECISION FUNCTION CLIGHT ()`
B1900	returns Julian date corresponding to Besselian date 1900.0 (2415020.31352). `DOUBLE PRECISION FUNCTION B1900 ()`
B1950	returns Julian date corresponding to Besselian date 1950.0 (2433282.423). `DOUBLE PRECISION FUNCTION B1950 ()`
J1900	returns Julian date corresponding to 1899 DEC 31 12:00:00 (2415020.0). `DOUBLE PRECISION FUNCTION J1900 ()`
J1950	returns Julian date corresponding to 1950 JAN 01 00:00:00 (2433282.5). `DOUBLE PRECISION FUNCTION J1950 ()`
J2000	returns Julian date corresponding to 2000 JAN 01 12:00:00 (2451545.0). `DOUBLE PRECISION FUNCTION J2000 ()`
J2100	returns Julian date corresponding to 2100 JAN 01 12:00:00 (2488070.0). `DOUBLE PRECISION FUNCTION J2100 ()`

SPICE function declarations

All of the functions above as well as any other SPICELIB function must be explicitly declared in the declaration section of the program before they can be called in the program's code. This assures that the function will return a value of the correct type. The two lines below declare the functions **SPD** and **DPR**.

```
...
DOUBLE PRECISION  DPR
DOUBLE PRECISION  SPD
```

Example

This fragment of code declares and calls the **DPR** function to calculate **ANG** in degrees.

```
DOUBLE PRECISION  DPR
...
ANG = ACOS( X ) * DPR ()
```

Rectangular Coordinates.

Routines

RECCYL calculates cylindrical coordinates — distance to Z axis **R**, angle from XZ plane **LONGC** and height above the XZ plane **Z** — of a point given by its rectangular coordinates **RECTAN**.

```
SUBROUTINE RECCYL ( RECTAN, R, LONGC, Z )
DOUBLE PRECISION  RECTAN (3)
DOUBLE PRECISION  R
DOUBLE PRECISION  LONGC
DOUBLE PRECISION  Z
```

RECGEO calculates geodetic coordinates — longitude **LONG**, latitude **LAT** and distance to center **ALT** — of a point given by its rectangular coordinates **RECTAN**, the equatorial radius of planet ellipsoid **RE** and the flattening coefficient **F** ($F=(R_{equ}-R_{pol})/R_{equ}$) of this ellipsoid.

```
SUBROUTINE RECGEO ( RECTAN, RE, F, LONG, LAT, ALT )
DOUBLE PRECISION  RECTAN (3)
DOUBLE PRECISION  RE
DOUBLE PRECISION  F
DOUBLE PRECISION  LONG
DOUBLE PRECISION  LAT
DOUBLE PRECISION  ALT
```

RECLAT calculates latitudinal coordinates — longitude **LONG**, latitude **LAT** and distance to center **RADIUS** — of a point given by its rectangular coordinates **RECTAN**.

```
SUBROUTINE RECLAT ( RECTAN, RADIUS, LONG, LAT )
DOUBLE PRECISION  RECTAN (3)
DOUBLE PRECISION  RADIUS
DOUBLE PRECISION  LONG
DOUBLE PRECISION  LAT
```

RECRAD calculates right ascension **RA**, declination **DEC** and distance from center **RANGE** for a point given by its rectangular coordinates **RECTAN**.

```
SUBROUTINE RECRAD ( RECTAN, RANGE, RA, DEC )
DOUBLE PRECISION  RECTAN (3)
DOUBLE PRECISION  RANGE
DOUBLE PRECISION  RA
DOUBLE PRECISION  DEC
```

RECSPH calculates spherical coordinates — distance to center **R**, angle between vector and Z axis **COLAT**, and angle between vector and XZ plane **LONG** — of a point given by its rectangular coordinates **RECTAN**.

```
SUBROUTINE RECSPH ( RECTAN, R, COLAT, LONG )
DOUBLE PRECISION  RECTAN (3)
DOUBLE PRECISION  R
DOUBLE PRECISION  COLAT
DOUBLE PRECISION  LONG
```

Example

This fragment of code loads a PCK file containing physical constants of planets, reads values for the Earth ellipsoid radii and calculates the geodetic coordinates of a point **X** given by its rectangular coordinates.

```
...
CALL FURNSH( '/kernels/generic/lsk/pck00006.tpc' )
CALL BODVAR( 399, 'RADII', N, R )
...
CALL RECGEO( X, R(1), (R(1)-R(3))/R(1), LONG, LAT, ALT )
```

Spherical and cylindrical coordinates.

Routines

CYLLAT calculates longitude **LONG**, latitude **LAT** and distance to center **RADIUS** for a point given by its cylindrical coordinates — distance **R**, angle **LONGC** and height **Z**.

```
SUBROUTINE CYLLAT ( R, LONGC, Z, RADIUS, LONG, LAT )
DOUBLE PRECISION  R
DOUBLE PRECISION  LONGC
DOUBLE PRECISION  Z
DOUBLE PRECISION  RADIUS
DOUBLE PRECISION  LONG
DOUBLE PRECISION  LAT
```

CYLREC calculates rectangular coordinates **RECTAN** of a point given by its cylindrical coordinates.

```
SUBROUTINE RECCYL ( R, LONGC, Z, RECTAN )
DOUBLE PRECISION  R
DOUBLE PRECISION  LONGC
DOUBLE PRECISION  Z
DOUBLE PRECISION  RECTAN (3)
```

CYLSPH calculates spherical coordinates — distance to center **RADIUS**, angle between point and Z axis **COLAT** and angle between vector and XZ plane **LONG**, for a point given by its cylindrical coordinates.

```
SUBROUTINE CYLSPH ( R, LONGC, Z, RADIUS, COLAT, LONG )
DOUBLE PRECISION  R
DOUBLE PRECISION  LONGC
DOUBLE PRECISION  Z
DOUBLE PRECISION  RADIUS
DOUBLE PRECISION  COLAT
DOUBLE PRECISION  LONG
```

SPHCYL calculates cylindrical coordinates — distance from Z axis **RADIUS**, angle from XZ plane **LONGC** and height above XZ plane **Z**, of a point given by its spherical coordinates — distance to center **R**, angle between vector and Z axis **COLAT** and angle between vector and XZ plane **LONG**.

```
SUBROUTINE SPHCYL ( R, COLAT, LONG, RADIUS, LONGC, Z )
DOUBLE PRECISION  R
DOUBLE PRECISION  COLAT
DOUBLE PRECISION  LONG
DOUBLE PRECISION  RADIUS
DOUBLE PRECISION  LONGC
DOUBLE PRECISION  Z
```

SPHLAT calculates latitudinal coordinates — longitude **LONG**, latitude **LAT** and distance from center **RADIUS**, of a point given by its spherical coordinates.

```
SUBROUTINE SPHLAT ( R, COLAT, LONG, RADIUS, LONG, LAT )
DOUBLE PRECISION  R
DOUBLE PRECISION  COLAT
DOUBLE PRECISION  LONG
DOUBLE PRECISION  RADIUS
DOUBLE PRECISION  LONG
DOUBLE PRECISION  LAT
```

SPHREC calculates rectangular coordinates **RECTAN** of a point given by its spherical coordinates.

```
SUBROUTINE SPHREC ( R, COLAT, LONG, RECTAN )
DOUBLE PRECISION  R
DOUBLE PRECISION  COLAT
DOUBLE PRECISION  LONG
DOUBLE PRECISION  RECTAN (3)
```

Latitudinal and Geodetic coordinates.

Routines

LATCYL calculates cylindrical coordinates — distance **RADIUS**, angle **LONGC** and height **Z**, of a point given by its latitudinal coordinates — longitude **LONG**, latitude **LAT** and distance to center **R**.

```
SUBROUTINE LATCYL ( R, LONG, LAT, RADIUS, LONGC, Z )
DOUBLE PRECISION  R
DOUBLE PRECISION  LONG
DOUBLE PRECISION  LAT
DOUBLE PRECISION  RADIUS
DOUBLE PRECISION  LONGC
DOUBLE PRECISION  Z
```

LATREC calculates rectangular coordinates **RECTAN** of a point given by its latitudinal coordinates.

```
SUBROUTINE LATREC ( R, LONG, LAT, RECTAN )
DOUBLE PRECISION  R
DOUBLE PRECISION  LONG
DOUBLE PRECISION  LAT
DOUBLE PRECISION  RECTAN (3)
```

LATSPH calculates spherical coordinates of a point given by its latitudinal coordinates.

```
SUBROUTINE LATSPH ( R, LONG, LAT, RADIUS, COLAT, LONG )
DOUBLE PRECISION  R
DOUBLE PRECISION  LONG
DOUBLE PRECISION  LAT
DOUBLE PRECISION  RADIUS
DOUBLE PRECISION  COLAT
DOUBLE PRECISION  LONG
```

GEOREC calculates rectangular coordinates **RECTAN** of a point given by its geodetic coordinates — longitude **LONG**, latitude **LAT** and distance from center **ALT**. Also returns the equatorial radius of the planet ellipsoid **RE** and the flattening coefficient **F** ($F=(R_{equ}-R_{pol})/R_{equ}$) of this ellipsoid.

```
SUBROUTINE GEOREC ( LONG, LAT, ALT, RE, F, RECTAN )
DOUBLE PRECISION  LONG
DOUBLE PRECISION  LAT
DOUBLE PRECISION  ALT
DOUBLE PRECISION  RE
DOUBLE PRECISION  F
DOUBLE PRECISION  RECTAN (3)
```

SRFREC calculates rectangular coordinates **RECTAN** of a point on the surface of a body (planet or satellite) with ID **BODY** given by the point's planetocentric longitude **LONG** and latitude **LAT**. A PCK file containing constants for this body must be loaded before this subroutine is called.

```
SUBROUTINE SRFREC ( BODY, LONG, LAT, RECTAN )
INTEGER           BODY
DOUBLE PRECISION  LONG
DOUBLE PRECISION  LAT
DOUBLE PRECISION  RECTAN (3)
```

Example

This fragment of code loads a PCK file and calculates rectangular coordinates of a point having geodetic longitude **LONG** and latitude **LAT** on the Mars surface.

```
...
CALL FURNSH( '/kernels/generic/pck/pck00006.tpc" )
...
CALL SRFREC( 499, LONG, LAT, VECT )
```

Simple operations on 3-D vectors.

Routines

VADD	adds two vectors **V1** and **V2** and writes result in vector **VOUT**. `SUBROUTINE VADD ( V1, V2, VOUT )`
VSUB	subtracts vector **V2** from vector **V1** and writes result vector in **VOUT**. `SUBROUTINE VSUB ( V1, V2, VOUT )`
VCRSS	computes cross product of vectors **V1** and **V2** and writes result vector in **VOUT**. `SUBROUTINE VCRSS ( V1, V2, VOUT )`
VDOT	returns dot product of two vectors **V1** and **V2**. `DOUBLE PRECISION FUNCTION VDOT ( V1, V2 )`
VSCL	multiplies vector **V1** and scalar **S** and writes result in vector **VOUT**. `SUBROUTINE VSCL ( S, V1, VOUT )`
VMINUS	negates vector **V1** and writes result in vector **VOUT**. `SUBROUTINE VMINUS ( V1, VOUT )`
VEQU	makes vector **VOUT** equal to vector **V1**. `SUBROUTINE VEQU ( V1, VOUT )`
VZERO	indicates whether vector **V1** is the zero vector. If “yes”, returns **.TRUE.** `LOGICAL FUNCTION VZERO ( V1 )`
VSEP	computes the separation angle between two vectors **V1** and **V2**. Returns zero if one of vectors is the zero vector. `DOUBLE PRECISION FUNCTION VSEP ( V1, V2 )`
VDIST	returns distance between two vectors **V1** and **V2**, equal to ‖**V1-V2**‖. `DOUBLE PRECISION FUNCTION VDIST ( V1, V2 )`
VNORM	computes magnitude of vector **V1**. `DOUBLE PRECISION FUNCTION VNORM ( V1 )`
VHAT	finds the unit vector **VOUT** along vector **V1**. `SUBROUTINE VHAT ( V1, VOUT )`
UCRSS	finds unit vector **VOUT** along the cross product of vectors **V1** and **V2**. `SUBROUTINE UCRSS ( V1, V2, VOUT )`
UNORM	finds magnitude **VMAG** of and unit vector **VOUT** along with vector **V1**. `SUBROUTINE UNORM ( V1, VOUT, VMAG )`

Arguments of subroutines

Input and output parameters of the routines listed above should be declared as follows:

```
DOUBLE PRECISION  V1 (3)
DOUBLE PRECISION  V2 (3)
DOUBLE PRECISION  VOUT (3)
DOUBLE PRECISION  S
DOUBLE PRECISION  VMAG
```

Projections, linear combinations and rotations of 3-D vectors.

Routines

VPERP	finds the component of vector **V1** that is rectangular to vector **V2** and writes it into vector **VOUT**. **SUBROUTINE VPERP (V1, V2, VOUT)**
VPROJ	finds the projection of vector **V1** onto vector **V2** and writes it in vector **VOUT**. **SUBROUTINE VPROJ (V1, V2, VOUT)**
VROTV	rotates vector **V1** about axis vector **V2** by angle **ANGLE** and writes result vector in **VOUT**. **SUBROUTINE VROTV (V1, V2, ANGLE, VOUT)**
ROTVEC	rotates vector **V1** about axis **IAXIS** given by its ID (for “X” axis ID is **1**, “Y”—**2**, “Z”—**3**) by angle **ANGLE** and writes the result in vector **VOUT**. **SUBROUTINE ROTVEC (V1, ANGLE, IAXIS, VOUT)**
NPLNPT	finds point **VOUT** nearest to point **V3** and belonging to the line given by point **V1** and direction **V2** and calculates distance **DIST** between points **V3** and **VOUT**. **SUBROUTINE NPLNPT (V1, V2, V3, VOUT, DIST)**
VPRJP	finds projection of vector **V1** into plane **PLANE** and writes result vector in **VOUT**. **SUBROUTINE VPRJP (V1, PLANE, VOUT)**
VPRJPI	finds the vector **VOUT** in specified plane **PROJPL** that maps to vector **V1** in another plane **INVPL** under orthogonal projection. The flag **FOUND** becomes **.FALSE.** if the required vector couldn’t be computed i.e. the planes are orthogonal or almost orthogonal. **SUBROUTINE VPRJPI (V1, PROJPL, INVPL, VOUT, FOUND)**
VLCOM	calculates the linear combination of the two vectors **V1** multiplied by **A** and **V2** multiplied by **B** and writes the result vector **VOUT**. **SUBROUTINE VLCOM (A, V1, B, V2, VOUT)**
VLCOM3	calculates the linear combination of the three vectors **V1**, **V2** and **V3** multiplied by **A**, **B** and **C** and returns it in vector **VOUT**. **SUBROUTINE VLCOM3 (A, V1, B, V2, C, V3, VOUT)**

Routines arguments

The input and output parameters of the routines listed above should be declared as shown below. The **UBPL** parameter is used for **PLANE** type variable declarations.

```
DOUBLE PRECISION  V1 (3)
DOUBLE PRECISION  V2 (3)
DOUBLE PRECISION  V3 (3)
DOUBLE PRECISION  VOUT (3)
DOUBLE PRECISION  ANGLE
DOUBLE PRECISION  DIST
DOUBLE PRECISION  A
DOUBLE PRECISION  B
DOUBLE PRECISION  C
LOGICAL           FOUND
INTEGER           IAXIS

INTEGER           UBPL
PARAMETER       ( UBPL   =   4 )
DOUBLE PRECISION  PLANE ( UBPL )
DOUBLE PRECISION  PROJPL( UBPL )
DOUBLE PRECISION  INVPL ( UBPL )
```

Operations on 3x3 matrixes.

Routines

MXM	multiples matrix **M1** and matrix **M2** and writes result in matrix **MOUT**. **SUBROUTINE MXM (M1, M2, MOUT)**
MXMT	multiplies matrix **M1** and the transpose of matrix **M2** and writes result in matrix **MOUT**. **SUBROUTINE MXMT (M1, M2, MOUT)**
MXV	multiplies matrix **M1** and vector **V1** and writes result in vector **VOUT**. **SUBROUTINE MXV (M1, V1, VOUT)**
MTXM	multiplies the transpose of matrix **M1** and matrix **M2** and writes result in matrix **MOUT**. **SUBROUTINE MTXM (M1, M2, MOUT)**
MTXV	multiplies the transpose of matrix **M1** and vector **V1** and writes result in vector **VOUT**. **SUBROUTINE MTXV (M1, V1, VOUT)**
VTMV	returns the multiplication of the transpose of vector **V1**, matrix **M1** and vector **V2**. **DOUBLE PRECISION FUNCTION VTMV (V1, M1, V2)**
XPOSE	finds the transpose of matrix **M1** and writes it in matrix **MOUT**. **SUBROUTINE XPOSE (M1, MOUT)**
MEQU	sets matrix **MOUT** equal to matrix **M1**. **SUBROUTINE MEQU (M1, MOUT)**
DET	returns the determinant of matrix **M1**. **DOUBLE PRECISION FUNCTION DET (M1)**
TRACE	returns the trace of matrix **M1**. **DOUBLE PRECISION FUNCTION TRACE (M1)**

Routines arguments

The input and output parameters of the routines listed above should be declared as follows:

```
DOUBLE PRECISION  V1 (3)
DOUBLE PRECISION  V2 (3)
DOUBLE PRECISION  VOUT (3)
DOUBLE PRECISION  M1 (3,3)
DOUBLE PRECISION  M2 (3,3)
DOUBLE PRECISION  MOUT (3,3)
```

Example

This fragment of code calculates the transformation matrix **MJ2INS** which rotates vectors from the inertial frame “J2000” to the instrument reference frame using two intermediate transformation matrixes: from “J2000” to instrument platform **MJ2PL**, and from platform to instrument **MPL2IN**. It then finds the position of the Sun **SUNINS** in the instrument reference frame.

```
...
CALL MXM ( MPL2IN, MJ2PL, MJ2INS )
CALL MXV ( MJ2INS, SUNJ, SUNINS )
```

Operations on planes.

`PLANE` data type

An array of dimension 4 is used in the SPICE system for representation of planes. It is recommended that for the **PLANE** type one should use parameter **UBPL** for dimension declarations.

```
INTEGER            UBPL
PARAMETER        ( UBPL   =   4 )
DOUBLE PRECISION   PLANE ( UBPL )
```

Routines

NVC2PL creates a plane **PLANE** using a normal vector **NORMAL** and the distance from origin to plane **CONST**.

```
SUBROUTINE NVC2PL ( NORMAL, CONST, PLANE )
```

NVP2PL creates a plane **PLANE** using a normal vector **NORMAL** and a point **POINT** belonging to the plane.

```
SUBROUTINE NVP2PL ( NORMAL, POINT, PLANE )
```

PSV2PL creates a plane **PLANE** using a point in the plane **POINT** and two linear independent vectors **V1** and **V2**.

```
SUBROUTINE PSV2PL ( POINT, V1, V2, PLANE )
```

PL2NVC calculates for plane **PLANE** its normal vector **NORMAL** and distance from plane to origin **CONST**.

```
SUBROUTINE PL2NVC ( PLANE, NORMAL, CONST )
```

PL2NVP calculates for plane **PLANE** its normal vector **NORMAL** and the point **POINT** belonging to it and nearest to the origin.

```
SUBROUTINE PL2NVP ( PLANE, NORMAL, POINT )
```

PL2PSV calculates for plane **PLANE** the point **POINT** nearest to the origin and two orthogonal vectors **V1** and **V2** lying in it.

```
SUBROUTINE PL2PSV ( PLANE, POINT, V1, V2 )
```

INRYPL finds the intersection of a ray given by a starting point **VERTEX**, direction **DIR** and plane **PLANE**, and returns the number of intersection point in **NXPTS** (can be **0** or **1**) and the coordinates of the point in **XPT** (if **NXPTS=1**).

```
SUBROUTINE INRYPL ( VERTEX, DIR, PLANE, NXPTS, XPT )
```

Routines arguments

The input and output parameters of the routines listed above should be declared as shown below:

```
INTEGER            UBPL
PARAMETER        ( UBPL   =   4 )
DOUBLE PRECISION   PLANE ( UBPL )

DOUBLE PRECISION   NORMAL (3)
DOUBLE PRECISION   CONST
DOUBLE PRECISION   POINT  (3)
DOUBLE PRECISION   V1     (3)
DOUBLE PRECISION   V2     (3)
DOUBLE PRECISION   VERTEX (3)
DOUBLE PRECISION   DIR    (3)
INTEGER            NXPTS
DOUBLE PRECISION   XPT    (3)
```

Operations on ellipses.

`ELLIPSE` *data type*

A double precision array of dimension 9 is used in the SPICE system for representation of ellipses in 3-dimensional space. It is recommended the for the **`ELLIPSE`** type one should use the **`UBEL`** parameter for dimension declarations.

```
INTEGER            UBEL
PARAMETER        ( UBEL   =    9 )
DOUBLE PRECISION   ELLIPS ( UBEL )
```

Routines

CGV2EL	creates an ellipse **`ELLIPS`** using its center **`CENTER`** and two generating vectors **`V1`** and **`V2`** (vectors can be non-orthogonal and even linearly dependent: in the last case the ellipse will be degenerate). **`SUBROUTINE CGV2EL ( CENTER, V1, V2, ELLIPS )`**
EL2CGV	finds for ellipse **`ELLIPS`** its center **`CENTER`** and vectors **`SMAJOR`** and **`SMINOR`** representing its axes. **`SUBROUTINE EL2CGV ( ELLIPS, CENTER, SMAJOR, SMINOR )`**
SAELGV	given two generating vectors, **`V1`** and **`V2`**, finds ellipse's axes vectors **`SMAJOR`** and **`SMINOR`**. **`SUBROUTINE SAELGV ( V1, V2, SMAJOR, SMINOR )`**
INELPL	finds intersection of ellipse **`ELLIPS`** and plane **`PLANE`** and writes number of intersection points to **`NXPTS`** and coordinates of these points in **`XPT1`** and **`XPT2`**. **`SUBROUTINE INELPL ( ELLIPS, PLANE, NXPTS, XPT1, XPT2 )`**
NPELPT	finds on ellipse **`ELLIPS`** the point **`NRPT`** nearest to a given point **`POINT`** and the distance between these points **`DIST`**. **`SUBROUTINE NPELPT ( POINT, ELLIPS, NRPT, DIST )`**
PJELPL	finds projection of ellipse **`ELLIPS`** on the plane **`PLANE`** and write the result in ellipse **`ELLOUT`**. **`SUBROUTINE PJELPL ( ELLIPS, PLANE, ELLOUT )`**

Routines arguments

The input and output parameters of the routines listed above should be declared as shown below:

```
INTEGER            UBEL
PARAMETER        ( UBEL   =    9 )
DOUBLE PRECISION   ELLIPS ( UBEL )
DOUBLE PRECISION   ELLOUT ( UBEL )

INTEGER            UBPL
PARAMETER        ( UBPL   =   4 )
DOUBLE PRECISION   PLANE ( UBPL )

DOUBLE PRECISION   V1     (3)
DOUBLE PRECISION   V2     (3)
DOUBLE PRECISION   SMAJOR (3)
DOUBLE PRECISION   SMINOR (3)
INTEGER            NXPTS
DOUBLE PRECISION   XPT1   (3)
DOUBLE PRECISION   XPT2   (3)
DOUBLE PRECISION   NRPT   (3)
DOUBLE PRECISION   DIST
```

Operations on ellipsoids.

Routines

NEARPT	finds on an ellipsoid given by axes **A**, **B** and **C**, the point **NRPT** nearest to a given point **POINT** on the ellipse, and returns the distance between them in **DIST**. **`SUBROUTINE NEARPT ( POINT, A, B, C, NRPT, DIST )`**
SURFPT	finds intersection of a ray given by its starting point **VERTEX** and direction **DIR** and an ellipsoid given by its axes **A**, **B** and **C**, and writes coordinates of this point in **XPT**. The flag **FOUND** becomes **.TRUE.** if such a point exists. **`SUBROUTINE SURFPT ( VERTEX, DIR, A, B, C, XPT, FOUND )`**
SURFNM	finds the unit normal vector **NORMAL** to the surface of an ellipsoid at the point **POINT** on the ellipsoid given by axes **A**, **B** and **C**. **`SUBROUTINE SURFNM ( A, B, C, POINT, NORMAL )`**
EDLIMB	finds limb of an ellipsoid given by axes **A**, **B** and **C** as seen from point **VIEWPT** and returns it in **ELLIPSE** type variable **LIMB**. **`SUBROUTINE EDLIMB ( A, B, C, VIEWPT, LIMB )`**
NPEDLN	finds on an ellipsoid given by axes **A**, **B** and **C** the point **NRPT** nearest to the line given by point **POINT** and direction **DIR**, and calculates the distance **DIST** between the line and point. **`SUBROUTINE NPEDLN ( A, B, C, POINT, DIR, NRPT, DIST )`**
INEDPL	finds the ellipse **ELLIPS** that is the intersection of the ellipsoid given by axes **A**, **B** and **C** and the plane **PLANE**. The flag **FOUND** becomes **.TRUE.** if such an intersection exists. **`SUBROUTINE INEDPL ( A, B, C, PLANE, ELLIPS, FOUND )`**

Routines arguments

The input and output parameters of the routines listed above should be declared as shown below:

```
DOUBLE PRECISION  A
DOUBLE PRECISION  B
DOUBLE PRECISION  C
DOUBLE PRECISION  POINT  (3)
DOUBLE PRECISION  NRPT   (3)
DOUBLE PRECISION  VERTEX (3)
DOUBLE PRECISION  DIR    (3)
INTEGER           NXPTS
DOUBLE PRECISION  XPT    (3)
DOUBLE PRECISION  DIST
DOUBLE PRECISION  VIEWPT (3)
LOGICAL           FOUND

INTEGER           UBEL
PARAMETER       ( UBEL   =    9 )
DOUBLE PRECISION  LIMB   ( UBEL )
DOUBLE PRECISION  ELLIPS ( UBEL )

INTEGER           UBPL
PARAMETER       ( UBPL   =    4 )
DOUBLE PRECISION  PLANE  ( UBPL )
```

Creation of 3x3 transformation matrixes.

Routines

ROTATE calculates the matrix **MOUT** which rotates vectors about axis **IAXIS** (for the “X” axis the ID is **1**, “Y”–**2**, “Z”–**3**) by angle **ANGLE**.

```
SUBROUTINE ROTATE ( ANGLE, IAXIS, MOUT )
DOUBLE PRECISION  ANGLE
INTEGER           IAXIS
DOUBLE PRECISION  MOUT (3,3)
```

ROTMAT rotates matrix **M1** by angle **ANGLE** about **IAXIS** axis (“X”—**1**, “Y”—**2**,“Z”—**3**) and returns resulting matrix in **MOUT**. So, **MOUT=[ANGLE]**$_{\mathbf{IAXIS}}$***M1**, where **[ANGLE]**$_{\mathbf{IAXIS}}$ is the matrix which rotates vectors about **IAXIS** axis by angle **ANGLE**.

```
SUBROUTINE ROTMAT ( M1, ANGLE, IAXIS, MOUT )
DOUBLE PRECISION  M1 (3,3)
DOUBLE PRECISION  ANGLE
INTEGER           IAXIS
DOUBLE PRECISION  MOUT (3,3)
```

TWOVEC finds transformation matrix **MOUT** which rotates vectors to the reference frame having a given vector **AXDEF** as specified axis **INDEXA** (“X”—**1**, “Y”—**2**,“Z”—**3**) and having a second given vector **PLNDEF** lying in the coordinate plane **INDEXA—INDEXP** (axis **INDEXP** is defined by the same rule). The direction of the third axis is taken from condition that this frame is right-handed.

```
SUBROUTINE TWOVEC ( AXDEF, INDEXA, PLNDEF, INDEXP, MOUT )
DOUBLE PRECISION  AXDEF (3)
INTEGER           INDEXA
DOUBLE PRECISION  PLNDEF (3)
INTEGER           INDEXP
DOUBLE PRECISION  MOUT (3,3)
```

EUL2M calculates the transformation matrix **MOUT** from Euler angles **ANG1**, **ANG2** and **ANG3** and their corresponding axes of rotation **AX1**, **AX2** and **AX3** (“X”—**1**, “Y”—**2**,“Z”—**3**).

```
SUBROUTINE EUL2M ( ANG3, ANG2, ANG1, AX3, AX2, AX1, MOUT  )
DOUBLE PRECISION  ANG3, ANG2, ANG1
INTEGER           AX3,  AX2,  AX1
DOUBLE PRECISION  MOUT (3,3)
```

M2EUL calculates Euler angles **ANG1**, **ANG2** and **ANG3** and the corresponding axes of rotation **AX1**, **AX2** and **AX3** (“X”—**1**, “Y”—**2**,“Z”—**3**) for the transformation matrix **M1**.

```
SUBROUTINE M2EUL ( M1, ANG3, ANG2, ANG1, AX3, AX2, AX1 )
DOUBLE PRECISION  M1 (3,3)
DOUBLE PRECISION  ANG3, ANG2, ANG1
INTEGER           AX3,  AX2,  AX1
```

Example

This fragment of code creates matrix **MROT** from right ascension **RA**, declination **DEC** and twist **TWIST**.

```
   ...
   CALL EUL2M ( TWIST, HALFPI()-DEC, RA, 3, 2, 3, MROT )
```

Orbital elements.

Orbital elements representation

Classical orbital elements are stored in double precision arrays containing `8` numbers.

```
DOUBLE PRECISION  ELTS ( 8 )
```

The elements of this array contain:

`ELTS(1)`	Distance to pericenter R_p, (km);
`ELTS(2)`	Eccentricity e;
`ELTS(3)`	Inclination i (rad);
`ELTS(4)`	Longitude of ascending node ▯ (rad);
`ELTS(5)`	Argument of periapse ▯ (rad);
`ELTS(6)`	mean anomaly at epoch E (rad);
`ELTS(7)`	epoch t (ephemeris seconds past J2000);
`ELTS(8)`	Gravitational parameter of planet ▯ (km^3/sec^2).

Routines

`CONICS` calculates the position and velocity **`STATE`** of an orbiting body from a set of elliptic, hyperbolic or parabolic orbital elements **`ELTS`** at ephemeris time **`ET`**.

```
SUBROUTINE CONICS ( ELTS, ET, STATE )
DOUBLE PRECISION  ELTS (8)
DOUBLE PRECISION  ET
DOUBLE PRECISION  STATE (6)
```

`OSCELT` given the state **`STATE`** of an orbiting body at ephemeris time **`ET`**, and given the gravitational parameter of the planet **`MU`**, calculates orbital elements **`ELTS`** for this orbiting body.

```
SUBROUTINE OSCELT ( STATE, ET, MU, ELTS )
DOUBLE PRECISION  STATE (6)
DOUBLE PRECISION  ET
DOUBLE PRECISION  MU
DOUBLE PRECISION  ELTS (8)
```

Example

This fragment of code reads position and velocity of a Mars-orbiting spacecraft with ID **`-23`** from loaded SPK files, transforms this state from the inertial frame **`'J2000'`** to the Mars “equator—north pole” non-rotating reference frame and calculates from this new state the orbital elements for the spacecraft.

```
...
CALL FURNSH ( '/kernels/sc23/spk/sc23_orbit036.bsp' )
CALL SPKEZR ( '-23', ET, 'J2000', 'NONE', 'MARS', STATE, LT )

DO I = 1, 3
   VEC(I) = STATE (I)
   VEL(I) = STATE (I+3)
END DO

CALL MXV ( MJ2MRS, VEC, VEC )
CALL MXV ( MJ2MRS, VEL, VEL )

DO I = 1, 3
   STATE (I)   = VEC(I)
   STATE (I+3) = VEL(I)
END DO

CALL OSCELT ( STATE, ET, MARSMU, ORBELM )
```

Observation Geometry – Sub-observer Point and Illumination Angles.

Routines

SUBPT

determines the coordinates of the sub-observer point **`SPOINT`** computed using “near point on triaxial ellipsoid” (**`METHOD='NEAR POINT'`**) or “intercept of radius vector with ellipsoid” (**`METHOD='INTERCEPT'`**) method for observer **`OBS`** on a target body **`TARG`** at ephemeris time **`ET`**. The resulting coordinates of the sub-observer point can be uncorrected (**`ABR='NONE'`**) or optionally corrected for light time (**`ABR='LT'`**) or light time and stellar aberration (**`ABERR='LT+S'`**). Also, returns the observer’s altitude **`ALT`** above the target body.

```
SUBROUTINE SUBPT ( METHOD, TARG, ET, ABR, OBS, SPOINT, ALT )
CHARACTER*(*)     METHOD
CHARACTER*(*)     TARG
DOUBLE PRECISION  ET
CHARACTER*(*)     ABR
CHARACTER*(*)     OBS
DOUBLE PRECISION  SPOINT ( 3 )
DOUBLE PRECISION  ALT
```

ILLUM

computes the illumination angles – phase angle **`PHASE`**, solar incidence angle **`SOLAR`**, and emission angle **`EMISSN`** – at a specified surface point **`SPNT`** of a target body **`TARG`** as seen from observer **`OBS`** at ephemeris time **`ET`**. The resulting angles can be computed using uncorrected (**`ABR='NONE'`**) or optionally corrected for light time (**`ABR='LT'`**) or light time and stellar aberration (**`ABR='LT+S'`**) state vectors for the observer, target and the sun.

```
SUBROUTINE ILLUM ( TARG, ET, ABR, OBS, SPNT, PHASE, SOLAR, EMISSN )
CHARACTER*(*)     TARG
DOUBLE PRECISION  ET
CHARACTER*(*)     ABR
CHARACTER*(*)     OBS
DOUBLE PRECISION  SPNT    ( 3 )
DOUBLE PRECISION  PHASE
DOUBLE PRECISION  SOLAR
DOUBLE PRECISION  EMISSN
```

Example

This fragment of code uses data from generic and MGS kernels listed in the following meta-kernel file:

```
\begindata
   KERNELS_TO_LOAD = (
                      '/kernels/generic/lsk/naif0007.tls'
                      '/kernels/generic/pck/pck00006.tpc'
                      '/kernels/generic/spk/de405.bsp'
                      '/kernels/mgs/spk/mgs_map1.bsp'
                     )
\begintext
```

to compute the illumination angles at the MGS sub-spacecraft point on the surface of Mars, determined using “nearest point of ellipsoid” method, at UTC 1999 April 1 12:00.

```
...
CALL FURNSH( 'mgs_kernels.furnsh' )
...
CALL STR2ET( '1999 April 1 12:00', ET )
CALL SUBPT ( 'NEAR POINT', 'MARS', ET, 'LT', 'MGS', SPNT, ALT )
CALL ILLUM ( 'MARS', ET, 'LT', 'MGS', SPNT, PHASE, SOLAR, EMISSN )
...
```

Observation Geometry – Surface Intercept Point.

Routines

SRFXPT determines the body-fixed coordinates of the surface intercept point **SPOINT** on the surface of the target body named **TARGET** with the shape model specified by **METHOD**, by the ray **DVEC** emanating from the observer body named **OBSRVR** and specified in the reference frame **DREF**, corrected according to the specified aberration correction **ABCORR,** at ephemeris time **ET**. The distance between the observer and surface intercept point **DIST**, intercept epoch **TRGEPC**, observer position **OBSPOS**, and a logical flag **FOUND** indicating whether the ray intercepts the target surface are returned in addition to the Cartesian coordinates of the intercept point. The only shape model currently supported by the routine is **‘ELLIPSOID’**.

```
SUBROUTINE SRFXPT ( METHOD,  TARGET,  ET,      ABCORR,
                    OBSRVR,  DREF,    DVEC,    SPOINT,
                    DIST,    TRGEPC,  OBSPOS,  FOUND )
CHARACTER*(*)         METHOD
CHARACTER*(*)         TARGET
DOUBLE PRECISION      ET
CHARACTER*(*)         ABCORR
CHARACTER*(*)         OBSRVR
CHARACTER*(*)         DREF
DOUBLE PRECISION      DVEC   ( 3 )
DOUBLE PRECISION      SPOINT ( 3 )
DOUBLE PRECISION      DIST
DOUBLE PRECISION      TRGEPC
DOUBLE PRECISION      OBSPOS ( 3 )
LOGICAL               FOUND
```

Example

This fragment of code uses data from generic and Mars 2001 Odyssey (M01) kernels listed in the following meta-kernel file:

```
\begindata
   KERNELS_TO_LOAD = (
                      '/kernels/generic/lsk/naif0007.tls'
                      '/kernels/generic/pck/pck00008.tpc'
                      '/kernels/generic/spk/de405.bsp'
                      ‘/kernels/m01/fk/m01.tf’
                      '/kernels/m01/sclk/m01.tsc'
                      ‘/kernels/m01/ik/m01_themis.ti’
                      '/kernels/m01/spk/m01_map1.bsp'
                      '/kernels/m01/ck/m01_map1.bc'
                      )
\begintext
```

to compute the surface intercept point of the THEMIS IR camera boresight at UTC 2002 MAR 12 12:00.

```
...
CALL FURNSH( 'm01_kernels.furnsh' )
...
CALL GETFOV( -53031, 4, SHAPE, FRAME, BSIGHT, N, BOUNDS )
...
CALL STR2ET( '2002-03-12 12:00', ET )
CALL SRFXPT( ‘ELLIPSOID’, ‘MARS’, ET, ‘LT+S’,
             ‘M01’, FRAME, BSIGHT,
             SPOINT, DIST, TRGEPC, OBSPOS, FOUND )
...
```