Most Useful SPICELIB Subroutines

NAIF

March 1999

Introduction

This document contains a brief description of the most often used SPICELIB routines. These are routines for reading different kinds of navigation and ancillary information and solving tasks of 3-dimensional geometry.

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

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

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

3-dimensional geometry

Solar system bodies and spacecrafts ID codes.

Planet Barycenters

Positive ID from 0 to 10 are assigned to Solar system planets 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 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.

Spacecrafts, on-board scientific instruments

Negative codes are used for spacecrafts. So, S/C “MGS” has ID **-94**, “Mars Polar Lander” — **-116**, “Mars Climate Orbiter” — **-127**, “Galileo orbiter” — **-77**, “Stardust” — **-29**, “Cassini” — **-82**, etc. The ID for a particular spacecraft is assigned by NAIF/JPL. These are often based on NASA DSN designations. See the document ***NAIF IDs Required Reading*** for a complete list.

The code of on-board 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**.

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. So 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.

Universal and Ephemeris times

Routines

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

```
SUBROUTINE LDPOOL( FNAME )
CHARACTER*(*)     FNAME
```

UTC2ET given a Universal Time **UTC**, calculates the corresponding Ephemeris Time **ET**.

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

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

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

ET2UTC 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
```

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
```

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 to ET 2 hours and converts this ET back to UTC in two ways.

```
CALL LDPOOL( "\naif\data\naif000c.tls" )
CALL STR2ET ( '1997 Jan 17 17:44:42.271', ET )
ET = ET + 7200
CALL ET2UTC ( ET, 'C',  3,                    UTC    )
CALL TIMOUT ( ET, 'YYYY MON DD HR: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
1995-08T18:28:12
```

Calendar date, for example

```
1986 JAN 9 03:12:59.22451
Tue Aug 6 11:10:57  1996
2/3/1996 17:18:12.002
```

Day of the year, for example

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

Julian date, for example

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

Ephemeris Time ET is a 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

LDPOOL loads SCLK file **FNAME** containing values required for SCLK string format interpretation and SCLK-to-Ephemeris Time (ET) correspondence calculation.

```
SUBROUTINE LDPOOL( 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 SCE2T( 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 SCE2T( SC, ET, CLKSTR )
SUBROUTINE SCT2E( 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 LDPOOL( "\naif\sc23\data\spcrft23.tsc" )
......
CALL SCE2C ( -23, ET, SCLKDP )
CALL SCDECD( -23, SCLKDP, CLKSTR )
......
```

SCLK formats

String representation: SCLK is represented as string of **"2/123.23.59.59.255"** type and is consisting of two parts. First part **"2/"** is partition number, second **"123.23.59.59.255"** is SCLK time in this partition (in given string dots are separating fields counting days, hours, minutes, seconds, 1/256 of seconds). Different spacecrafts have different set of fields.

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

Constants and matrixes of planets and satellites (PCK).

Routines

LDPOOL loads text PCK file **FNAME** containing constants for Solar system bodies.

```
SUBROUTINE LDPOOL( FNAME )
CHARACTER*(*)     FNAME
```

PCKLOF loads binary PCK file **FNAME** containing orientation data for one or more Solar system bodies and returns the file handle **HANDLE** for this file.

```
SUBROUTINE PCKLOF ( FNAME, HANDLE )
CHARACTER*(*)     FNAME
INTEGER           HANDLE
```

TIPBOD calculates matrix **TIPM** rotating vectors from inertial reference frame with ID **IRF** to body-fixed reference frame for the body with ID **BODY** at the given ephemeris time **ET**.

```
SUBROUTINE TIPBOD( IRF, BODY, ET, TIPM )
CHARACTER*(*)     IRF
INTEGER           BODY
DOUBLE PRECISION  ET
DOUBLE PRECISION  TIPM (3,3)
```

BODEUL returns Euler angles — right ascension **RA**, declination **DEC** and twist **W** that are used for calculation of a matrix used to rotate vectors from inertial frame “J2000” to the body-fixed frame for the body with ID **BODY** at ephemeris time **ET**. Also returns longitude of prime meridian, **LAMBDA**, for this body (**LAMBDA=0** for all planets except Mars).

```
SUBROUTINE BODEUL( BODY, ET, RA, DEC, W, LAMBDA )
INTEGER           BODY
DOUBLE PRECISION  ET
DOUBLE PRECISION  RA, DEC, W, LAMBDA
```

TISBOD calculates the matrix **TISM** used to rotate state vectors (position and velocity) from the inertial frame with name **IRF** to the body-fixed reference frame for the body with ID **BODY** at ephemeris time **ET**.

```
SUBROUTINE TISBOD( IRF, BODY, ET, TISM )
CHARACTER*(*)     IRF
INTEGER           BODY
DOUBLE PRECISION  ET
DOUBLE PRECISION  TISM (6,6)
```

BODVAR returns the vector **VALUES** (and its dimension **DIM)** containing value(s) for the physical parameter named **ITEM** for the body with ID **BODY**. As examples, to retrieve the axes of ellipsoid model of a planet **ITEM** is set to **"RADII"**, for planet nutation precession angles — **"NUT_PREC_ANGLES"**, etc.

```
SUBROUTINE BODVAR( BODY, ITEM, DIM, VALUES )
INTEGER           BODY
CHARACTER*(*)     ITEM
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 body-fixed frame for Mars, and retrieves the lengths of the three axes defining the Mars ellipsoid.

```
CALL LDPOOL( "\naif\data\naif000c.tpc" )
......
CALL TIPBOD( "B1950", 499, ET, MARSMT )
CALL BODVAR( 499, "RADII", DIM, MARSRD )
```

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

Routines

LDPOOL loads text Frame kernel file **FNAME** containing frames definitions for a particular spacecraft, instrument or other structure of interest.

```
SUBROUTINE LDPOOL( FNAME )
CHARACTER*(*)     FNAME
```

SXFORM calculates the matrix **TISM** 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 frame naming convention

The inertial frame naming convention is described in the *Inertial Reference Frames* section of this document.

PCK-based frame naming convention

Another 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 call.

User-defined frame naming convention

Yet 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 spacecrafts and instruments are loaded into SPICE simultaneously, NAIF recommends including abbreviated spacecraft name and instrument name in the prefix of any user-defined frame name. For example the image frame for the MGS camera (MOC) can be called **"MGS_MOC_IMAGE"**, and the base frame for the MCO PMIRR instrument can be called **"MCO_PMIRR_BASE"**, and so on. Refer to a frame kernel for particular mission for a complete list of user-defined frames for that mission.

Example

This fragment of code loads an 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 LDPOOL( "/kernels/mgs/frames/mgs.tf" )
CALL LDPOOL( "/kernels/generic/lsk/naif0007.tls" )
CALL LDPOOL( "/kernels/mgs/sclk/mgs.tsc" )
CALL CKLPF ( "/kernels/mgs/ck/mgs_spice_c_kernel_1998-339.bc" )
CALL CKLPF ( "/kernels/mgs/ck/mgs_solar_array_1998-339.bc" )
......
CALL SXFORM( "IAU_MARS", "MGS_MAG_+Y_SENSOR", ET, XMAT )
......
```

Planet and Spacecraft positions (SPK).

Routines

SPKLEF loads an SPK 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 SPKLEF( FNAME, HANDLE )
CHARACTER*(*)     FNAME
INTEGER           HANDLE
```

SPKEZ 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 integer IDs, accordingly **TARG** for "target" and **OBS** for "observer". The state vector is calculated in the requested reference frame with name **FRAME** from the list of frames supported within 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"**), geometric (**"NONE"**). This routine also returns one-way light-time from "target" to "observer" **LT**.

```
SUBROUTINE SPKEZ( TARG, ET, FRAME, ABERR, OBS, STATE, LT )
INTEGER           TARG
DOUBLE PRECISION  ET
CHARACTER*(*)     FRAME
CHARACTER*(*)     ABERR
INTEGER           OBS
DOUBLE PRECISION  STATE (6)
DOUBLE PRECISION  LT
```

SPKEZR performs the same calculation as **SPKEZ** but "target" body and "observer" body are specified by their names **TARGNM** and **OBSNM** instead of numeric ID codes.

```
SUBROUTINE SPKEZ( 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
```

SPKUEF unloads previously loaded SPK having file handle **HANDLE**.

```
SUBROUTINE SPKUEF( HANDLE )
INTEGER           HANDLE
```

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**) covering the same period of time. 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 Sun and spacecraft with respect to Mars center in Mars body-fixed rotating frame "IAU_MARS". The loading of SPK and PCK files is normaly done only once at the beginning of the program, while the computation of state vectors is usually repeated for many instants of time.

```
......
CALL LDPOOL ( '\naif\data\pck00005.tpc' )
CALL SPKLEF ( '\naif\data\de200.bsp',          HANDLE(1) )
CALL SPKLEF ( '\naif\sc23\data\orbit142.bsp', HANDLE(2) )
......
CALL SPKEZR ( 'SUN', ET, 'IAU_MARS', 'NONE', 'MARS',  SUNST, SUNLT  )
CALL SPKEZ  (   -23, ET, 'IAU_MARS', 'NONE',    499, SC23ST, SC23LT )
```

Attitude of spacecrafts and instrument platforms (CK).

Routines

CKLPF

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

```
SUBROUTINE CKLPF( FNAME, HANDLE )
CHARACTER*(*)     FNAME
INTEGER           HANDLE
```

CKGP

calculates the transformation matrix **CMAT** used to rotate a vector from the reference frame named **REF** from the list of frames supported within SPICE system to the platform-fixed reference frame for the instrument platform with ID **INSTR** at the time **SCLK** that is double precision encoding of SCLK. If pointing data in the loaded file is continuous, then the matrix will be returned at exactly the requested SCLK and **SCLKOUT** will be equal to **SCLK**. If pointing data in the loaded file is discrete then the matrix will be calculated for 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**, otherwise it will be **.FALSE.**

```
SUBROUTINE CKGP( INSTR, SCLK, TOL, REF, CMAT, SCLKOUT, FND )
INTEGER           INSTR
DOUBLE PRECISION  SCLK
DOUBLE PRECISION  TOL
CHARACTER*(*)     REF
DOUBLE PRECISION  CMAT (3,3)
DOUBLE PRECISION  SCLKOUT
BOOLEAN           FND
```

CKGPAV

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 instrument platform with ID **INS** at the time **SCLK** that is double precision 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 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.**

```
SUBROUTINE CKGPAV( INS, SCLK, TOL, REF, CMAT, AV, 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
```

CKUPF

unloads the previously loaded CK having file handle **HANDLE**.

```
SUBROUTINE CKUPF( HANDLE )
INTEGER           HANDLE
```

Example

This fragment of code laods a CK file containing pointing data for the instrument platform with ID **-23001**, calculates a transformation matrix used to rotate vectors from the inertial frame “B1950” to the platform-fixed reference frame and performs this rotation on the vector **X**.

```
......
CALL CKLPF( "\naif\sc23\data\orbit142.bc", HANDLE )
......
CALL CKGP ( -23001, SCLK, 200, "B1950", CMAT, SCLKOUT, FND )
CALL MXV  ( CMAT, X, XOUT )
```

Scientific Instruments Parameters (IK).

Routines

LDPOOL loads an IK file named **FNAME** containing mounting alignment and other parameters for a particular scientific instrument.

```
SUBROUTINE LDPOOL( FNAME )
CHARACTER*(*)     FNAME
```

RTPOOL reads values for the instrument parameter with name **NAME**, stores these values in the array **VALUES** and returns the number of them in **DIM**. Flag **FOUND** becomes **.TRUE.** if the requested parameter (and its values) was found among the loaded parameters.

```
SUBROUTINE RTPOOL( NAME, DIM, VALUES, FOUND )
CHARACTER*(*)     NAME
INTEGER           DIM
DOUBLE PRECISION  VALUES
BOOLEAN           FOUND
```

Example

This fragment of code loads an IK file containing parameters for the instrument with code **-23036**, the reads from the loaded data the values of Euler angles **ANG** and the corresponding rotation axes **AXS** defining mounting alignment of the instrument (attitude of instrument reference frame) relative ot the the instrument platform, and finally calculates the transformation matrix **MAT** that rotates vectors from the platform-fixed reference frame to the instrument-fixed reference.

```
......
CALL LDPOOL( "\naif\sc23\data\ins23036.ti" )
......
CALL RTPOOL( "INS-23036_EULER_ANGLES", N, ANG, FOUND )
CALL RTPOOL( "INS-23036_EULER_AXES",   N, AXS, FOUND )
CALL EUL2M ( ANG(3), ANG(2), ANG(1),
             AXS(3), AXS(2), AXS(1), MAT )
```

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.

Each instrument has its own set of parameters contained in an IK file. This set and the names of the parameters are defined by the file creator. The minimum set of instrument parameters in a SPICE IK file consists of three items:

INS-nnnnn_EULER_ANGLES,
INS-nnnnn_EULER_AXES and
INS-nnnnn_PLATFORM_ID,

containing Euler angles and corresponding axes of rotation that specify the mounting alignment of the instrument relative to its platform and the ID of this platform.

Star Catalog

Routines

STCL01 loads a SPICE type 1 star catalog file named **CATFNM** and returns the catalog table name **TABNAM** and the integer file handle **HANDLE** for this file.

```
SUBROUTINE STCL01 (CATFNM, TABNAM, HANDLE)
CHARACTER*(*)     CATFNM
CHARACTER*(*)     TABNAM
INTEGER           HANDLE
```

STCF01 searches through a type 1 star catalog data table named **CATNAM** and returns the number of stars **NSTARS** within a rectangle specified by West and East right ascensions, **WESTRA** and **EASTRA**, and South and North declinations, **STHDEC** and **NTHDEC**, given in radians in the J2000 inertial frame.

```
SUBROUTINE STCF01 (CATNAM, WESTRA, EASTRA, STHDEC, NTHDEC, NSTARS)
CHARACTER*(*)     CATNAM
DOUBLE PRECISION  WESTRA
DOUBLE PRECISION  EASTRA
DOUBLE PRECISION  STHDEC
DOUBLE PRECISION  NTHDEC
INTEGER           NSTARS
```

STCG01 gets right ascensions **RA**, declination **DEC**, the uncertainties in right ascension, **RASIG**, and declination, **DECSIG**, catalog number **CATNUM**, spectral type **SPTYPE** and visual magnitude **VM** for a single star specified by its index **INDEX** in the list of stars that satisfy the selection criteria specified in the last call to STCF01 from a SPICE type 1 star catalog. The **RA**, **DEC**, **RASIG** and **DECSIG** are returned in the J2000 inertial frame at the catalog epoch and are given in radians.

```
SUBROUTINE STCG01 (INDEX, RA, DEC, RASIG, DECSIG, CATNUM, SPTYPE, VM)
INTEGER           INDEX
DOUBLE PRECISION  RA
DOUBLE PRECISION  DEC
DOUBLE PRECISION  RASIG
DOUBLE PRECISION  DECSIG
INTEGER           CATNUM
CHARACTER*(*)     SPTYPE
DOUBLE PRECISION  VMAG
```

Example

This fragment of code loads a SPICE type 1 star catalog file, searches the loaded catalog for stars within a specified RA—DEC rectangle and retrieves a complete set of data for each star.

```
......
CALL STCL01 ( CATFN, TABNAM, HANDLE )
......
CALL STCF01 ( TABNAM, RAMIN, RAMAX, DECMIN, DECMAX, NSTARS )
DO I = 1, NSTARS
   CALL STCG01 ( I, R(I), D(I), RS(I), DS(I), CN(I), SP(I), VM(I) )
END DO
```

Spacecraft Event Information (Data Base Kernel)

Routines

EKLEF loads an EK file named **FNAME**, making it accessible to the EK readers and returns the integer file handle **HANDLE** for this file.

```
SUBROUTINE EKLEF ( FNAME, HANDLE )
CHARACTER*(*)     FNAME
INTEGER           HANDLE
```

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** (subrotine **EKGD**), character element **CDATA** (subrotine **EKGC**) or integer element **IDATA** (subrotine **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 its 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
```

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 expression> [AND/OR <constraint expression> ...]]
[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 **<>**. Operators **BETWEEN** and **NOT BETWEEN** are also supported.

Example

This fragment of code loads an EK file contaning a table called **EVENTS** containing time-type column **EVENT_TIME**, integer column **EVENT_ID**, double precision column **DURATION** and 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 EKLEF( FNAME, HANDLE )
......
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 ET2UTC( ET, 'C', 2, 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 $\pi/2$, 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 $2*\pi$, 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

Any of the functions above as well as any other SPICELIB function must be implicitly declared in the declaration section of the program before it can be called in the programs code. This assures that this function will return a value of the correct when it will be used in the program statements. Two lines below declare the fucntions **SPD** and **DPR**.

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

Example

This fragment of code declares and calls **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 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 point 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 of Earth ellipsoid radii and calculates the geodetic coordinates of a point **X** given by its rectangular coordinates.

```
......
CALL LDPOOL( "\naif\data\naif000c.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 point 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 flattering 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 PCK file and calculates rectangular coordinates of a point having geodetic longitude **LONG** and latitude **LAT** on the Mars surface.

```
......
CALL LDPOOL( "\naif\data\naif000c.tpc" )
......
CALL SRFREC( 499, LONG, LAT, VECT )
```

Simple operations on 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 with vector **V1**.

```
SUBROUTINE VHAT ( V1, VOUT )
```

UCRSS finds unit vector **VOUT** along with 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 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 from point **V3** and belonging 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 (planes are orthogonal or almost orthogonal).

SUBROUTINE VPRJPI (V1, PROJPL, INVPL, VOUT, FOUND)

VLCOM calculates linear combination of two vectors **V1** multiplied by **A** and **V2** multiplied by **B** and writes result in vector **VOUT**.

SUBROUTINE VLCOM (A, V1, B, V2, VOUT)

VLCOM3 calculates linear combination of three vectors **V1**, **V2** and **V3** multiplied accordingly by **A**, **B** and **C** and returns it in vector **VOUT**.

SUBROUTINE VLCOM3 (A, V1, B, V2, C, V3, VOUT)

Routines arguments

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 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

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 for **`PLANE`** type variable declaration to 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 plane.

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

PSV2PL creates a plane **`PLANE`** using a point in 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 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 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

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 for **ELLIPSE** type variable declaration to 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 it in ellipse **ELLOUT**.

```
SUBROUTINE PJELPL ( ELLIPS, PLANE, ELLOUT )
```

Routines arguments

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 its axes **A**, **B** and **C**, the point **NRPT** nearest to a given point **POINT**, and returnes 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 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 on 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** which is the intersection of ellipsoid given by axes **A**, **B** and **C** and plane **PLANE**. The flag **FOUND** becomes **.TRUE.** if such an intersection exists.

```
SUBROUTINE INEDPL ( A, B, C, PLANE, ELLIPS, FOUND )
```

Routines arguments

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 transformation matrixes.

Routines

ROTATE

calculates the matrix **MOUT** which rotates vectors about axis **IAXIS** (for "X" axis 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 returnes 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 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 given right ascension **RA**, declination **DEC** and twist **TWIST**.

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

Orbital elements.

Orbital elements representation

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 Ω (rad);
- **ELTS(5)** argument of periapse ω (rad);
- **ELTS(6)** mean anomaly at epoch E (rad);
- **ELTS(7)** epoch t (ephemeris seconds past J2000);
- **ELTS(8)** gravitational parameter of planet μ (km^3/sec^2).

Routines

CONICS calculates the position and velocity **STATE** of an orbiting body (spacecraft) from a set of elliptic, hyperbolic or parabolic orbital elements **ELTS** at a time given as 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 inertial frame “J2000” to Mars “equator—north pole” non-rotating reference frame and calculates from this new state the orbital elements for the spacecraft.

```
......
CALL SPKLEF( "\naif\sc23\data\orbit036.bsp", HANDLE )
CALL SPKEZ  ( -23, ET, "J2000", "NONE", 499, 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 )
```