SPK REQUIRED READING

Navigation Ancillary Information Facility

Prepared by
W. L. Taber
N. J. Bachman
F. S. Turner

NAIF Document No. 168.07
Last Modified: 06 OCT 1999

Table of Contents

S- and P-Kernel Files

Revisions

October 14, 1999

The routines

```
SPKPOS
SPKEZP
SPKGPS
SPKAPO
```

were added with version N0050 of the SPICE Toolkit. These routines are the "position only" equivalents of state routines

```
SPKEZR
SPKEZ
SPKGEO
SPKAPP
```

respectively. The calling sequences of the position only routines are identical to the state routines. However, where the state routines return 6-vectors (position and velocity), the position only routines return a 3-vector (just position). Moreover, the positions returned by the position only routines agree with the positions returned by the state routines.

Although the position only routines do not return as much information as the state routines (they don't return velocity), they are in some respects more general than the state routines. This is due to the link between the frame system and the SPK system. Some reference frames do not contain rate information. Consequently when a state is requested relative to such a frame, the state routines cannot perform transformations on the velocity components of the state. However, since the position only routines are not sensitive to the rate information, they can still perform position transformations and return the requested position.

March 2, 1998

06 OCT 1999

This version contains corrections of typographical errors and miscellaneous format changes. A note has been added on porting SPK files between SPICELIB and CSPICE (the ANSI C version of SPICELIB).

June 24, 1997

This version of this document is a major reorganization and expansion of the material presented in the December 1994 version.

Overview of the June 24, 1997 revision

Because of the substantial changes made in this revision, the description of those changes is retained here.

When the SPK system was introduced, states of objects (positions and velocities) were stored relative to inertial frames and retrieved relative to inertial frames. Beginning with version 41 of the NAIF Toolkit, states can be stored relative to both inertial and non-inertial frames. Moreover, states may be retrieved relative to both inertial and non-inertial frames. Non-inertial frames may be tied to the rotation of a planet, the orientation of some structure on a spacecraft, an Earth based telescope, etc. By expanding the SPK system in this way, computation that previously required dozens lines of code may now be reduced to three or four lines of code.

This version of the "SPK Required Reading" documents for the first time this important expansion of the SPK system.

Also in this version, we document:

1. the ability to request states of objects by name instead of by object ID codes;
2. the addition of SPK data Type 10 which allows the incorporation of NORAD "two-line" elements for Earth orbiters into the SPK system;
3. the addition of SPK data Type 14 which supports Chebyshev interpolation over non-uniformly spaced time intervals;
4. the addition of SPK data Type 17 which supports the inclusion of equinoctial elements into the SPK system.

The complete list of routines that are documented here for the first time is:

```
LTIME
SPKEZR
SPKPVN
SPKE10
SPKE14
SPKE17
SPKR10
```

```
SPKR14
SPKR17
SPKS10
SPKS14
SPKS17
SPKW10
SPK14A
SPK14B
SPK14E
SPKW17
```

Purpose

The purpose of this document is to describe the NAIF Toolkit software provided in the software library SPICELIB, (SPICE LIBrary) used for producing and accessing SPICE ephemeris data. In addition this document describes SPK—the common file format for NAIF's S-kernel and ephemeris portion of the P-kernel.

Intended Audience

This document is intended for all users of SPK (ephemeris) kernel files.

References

All references are to NAIF documents. The notation [Dn] refers to NAIF document number.

1. [349] Frames Required Reading
2. [174] CK Required Reading
3. [254] PCK Required Reading
4. [222] Spacecraft Clock Time Required Reading. (SCLK)
5. [218] KERNEL Required Reading.
6. [219] NAIF IDS Required Reading.
7. [163] JPL Internal Memorandum on Modified Difference Array polynomials; F. Krogh
8. [164] Precession Matrix Based on IAU (1976) System of Astronomical Constants; E. M. Standish; Astronomy and Astrophysics 73, 282-284 (1979)

9. [165] Orientation of the JPL Ephemerides, DE200/LE200, to the Dynamical Equinox of J2000; E. M. Standish; Astronomy and Astrophysics 114, 297-302 (1982)

10. [166] The JPL Asteroid and Comet Database (as Implemented by NAIF); a collection of papers and memos; assembled by I. Underwood; 11 Dec 1989

11. [167] Double Precision Array Files (DAF) - Required Reading; latest version

12. [212] COMMNT User's Guide

If you're in a hurry

We'll discuss things in more detail in a moment but in case you are just looking for the right name of the routine to perform some ephemeris task, here is a categorization of the most frequently used SPK and related routines in SPICELIB. Input arguments are given in lower case and enclosed in "angle brackets." Output arguments are given in upper case.

High Level Routines

Loading/Unloading an SPK file

```
SPKLEF ( <file>, HANDLE )
SPKUEF ( <handle> )
```

Retrieving states (position and velocity) using names of objects

```
SPKEZR ( <object>, <et>, <frame>, <corr>, <observer>, STATE, LT )
```

Retrieving positions using names of objects

```
SPKPOS ( <object>, <et>, <frame>, <corr>, <observer>, POSTN, LT )
```

Retrieving states using NAIF ID codes

```
SPKEZ  ( <obj_id>, <et>, <frame>, <corr>, <obj_id>, STATE, LT )

SPKGEO ( <obj_id>, <et>, <frame>,         <obj_id>, STATE, LT )
```

Retrieving positions using NAIF ID codes

```
SPKEZP ( <obj_id>, <et>, <frame>, <corr>, <obj_id>, POSTN, LT )

SPKGPS ( <obj_id>, <et>, <frame>,         <obj_id>, POSTN, LT )
```

Calculating “Uplink and Downlink” Light Time

```
LTIME ( <etobs>, <obs_id>, <dir>, <targ_id>, ETTARG, ELAPSD )
```

Loading/Unloading Binary PCK files (see PCK Required Reading)

```
PCKLOF ( <binary_pck>, HANDLE )
PCKUOF ( <handle> )
```

Loading Text based kernels—PCK, SCLK, etc.

```
LDPOOL ( <text_kernel> )
```

Loading/Unloading C-kernels (see CK Required Reading)

```
CKLPF ( <c-kernel>, HANDLE )
CKUPF ( <handle>          )
```

Foundation Routines

The routines listed in this section are the real “work horses” of the SPK and related systems. Not all of the routines in this section are described in this document. In those cases, the appropriate SPICE document is cited.

Selecting files and segments

```
SPKSFS ( <target>, <et>, HANDLE, DESCR, IDENT, FOUND )
```

Computing states from segment descriptors

```
SPKPVN ( <handle>, <descr>, <et>, REF, STATE, CENTER )
```

Correcting for stellar aberration

```
STELAB ( POBJ, VOBS, APPOBJ )
```

Translating between object names and object ID codes (see NAIF_IDS Required Reading)

```
BODN2C ( <name>,   IDCODE, FOUND )
BODC2N ( <idcode>, NAME,   FOUND )
```

Translating between frame names and frame ID codes (see Frames Required Reading)

```
FRMNAM ( <idcode>, NAME   )
NAMFRM ( <name>,   IDCODE )
```

State transformation matrices (see Frames Required Reading)

```
FRMCHG ( <from_idcode>, <to_idcode>, <et>, MAT6X6 )
```

Classifying frames (see Frames Required Reading)

```
FRINFO ( <idcode>, CENTER, CLASS, CLSSID, FOUND )
```

Utility Programs

Examining SPK files

```
brief
commnt
spacit
```

Converting to and from transfer format

```
spacit
tobin
toxfr
```

Introduction

To help fully understand the science data returned from a spacecraft's instruments it is necessary to know, at any given epoch, the positions and possibly the velocities of the spacecraft and all the target bodies of interest. The purpose of the SPK—which stands for S(pacecraft) and P(lanet) Kernel—file is to allow ephemerides for any collection of solar system bodies to be combined under a common file format, and accessed by a common set of subroutines.

Historically, ephemerides for spacecraft have been organized differently from those for planets and satellites. They are usually generated through different processes and using different representations. However, there is no essential reason for keeping them separate. A spacecraft, planet, satellite, comet, or asteroid has a position and velocity relative to some center of mass and reference frame. Consequently all of these objects can be represented in an SPK file.

Consider the Galileo mission. Some of the objects of special interest to the Galileo mission are:

```
Galileo Spacecraft
Galileo Probe
Earth
Moon
Earth Moon Barycenter
Venus
Sun
Solar System Barycenter (S.S.B.)
Asteroid Ida
Ida's Satellite Dactyl
Asteroid Gaspra
Comet Shoemaker-Levy
Jupiter System Barycenter (J.B.)
Jupiter
Io
Ganymede
Europa
Callysto
Goldstone Tracking Station.
```

Each of these objects has a position and velocity (state) relative to some other object. The graph below illustrates which objects will be used as reference objects for representing the states of others.

```
                  +Gll
                 /            probe
                /               |     o Comet
        Gaspra /              Gll+   /  Shoemaker Levy
  Gll +--o    /                  \ /
         |   /   Venus    Jupiter o--probe
         |  /      o--+           |
```

```
 Gll +       |  /      /   Gll         |  Io
     |       | /      /                |  o-----+Gll
     |       |/      /             J.B.| /
Ida  o-------o------o------------------o ----o------+Gll
    /         Sun   S.S.B.            / \    Europa
   o                 \      Ganymede /   \
Dactyl                \             o     \
                       \            |      o Callisto
 Earth-Moon Barycenter  o----o      +      |
                        |   Moon    Gll    |
                        |                  + Gll
                        o Earth
                       / \
                      /   \
                     /     + Gll
                    o
                 Goldstone
```

This graph is somewhat complicated. Nevertheless, the complete ephemeris history for all of these objects can be captured in a single SPK file.

(Although we can store the entire ephemeris history illustrated above in a single SPK file, for the sake of data management a project is likely to use several SPK files. However, even in this case, all of the SPK files can be used simultaneously.)

The SPK format is supported by a collection of subroutines that are part of the SPICELIB library—the major component of the NAIF Toolkit. This family of SPK subroutines provides the following capabilities:

1. Insert ephemeris data from some source into an SPK file.
2. Make the ephemeris data in one or more SPK files available to a user's program.
3. Return the apparent, true, or geometric state (position and velocity) of one ephemeris object as seen from another in some convenient reference frame.

The SPK software allows you to ignore the potential ephemeris complexity associated with the a mission such as Galileo and allows you to more directly compute various quantities that depend upon the position or velocity of one object as seen from another.

SPK Files

SPK files are binary files. The format of these binary files is based upon a more abstract file architecture called Double precision Array File (DAF). Several other SPICE kernels are also based on the DAF architecture. If you are only going to be a consumer of SPK files or if you will be using a SPICE utility program for creating SPK files, you can safely ignore aspects of the DAF system that are not covered by this document. On the other hand, if you plan to write software for creating SPK files you will probably need to familiarize yourself with the DAF software contained in SPICELIB. The DAF architecture and supporting software is discussed in [169]. The particular aspects of the DAF architecture that are relevant to the SPK format are discussed later in this document (see below—SPK Format).

Moving SPK files between computers

A binary file suitable for use on one computer, may not be suitable for use on another computer. For example, SPK files created on a Sun Sparc-10 are not suitable for use on a PC running SPICE software compiled with the Microsoft FORTRAN compiler. As a result you can't always perform a "binary copy" of an SPK file from one machine to another. NAIF provides two utility programs—TOXFR and SPACIT for converting SPICE binary kernels to a "transfer format" that is suitable for text copying from one computer to another. Once the transfer format file has been copied, the NAIF utilities TOBIN and SPACIT are available for converting the transfer format file to the binary format suitable for the new machine.

The utilities TOXFR and TOBIN are "command line" programs. To convert a binary kernel to transfer format you simply type TOXFR followed by the name of the binary kernel at your terminal prompt.

```
prompt> toxfr spk_file
```

To convert a transfer format to binary format, you type TOBIN followed by the name of the transfer format kernel.

```
prompt> tobin transfer_file
```

The utility SPACIT is an interactive program that allows you to select an action to perform on a file from a list of possibilities. It can be used to convert to or from transfer format files.

Note that transfer format files are suitable only for moving data from one machine to another. They cannot be "loaded" into a SPICE based program to retrieve ephemeris data. Only binary format files can be used for retrieving ephemeris data with SPICE software.

Porting files between SPICELIB and CSPICE

In most environments where SPICELIB is supported, CSPICE can use the same binary kernels as does SPICELIB. Environments where this local compatibility exists are:

```
Sun, Solaris; Sun Fortran; Sun C or Gnu gcc
Silicon Graphics, IRIX OS; Silicon Graphics Fortran; Gnu gcc
HP Series 700 computers, HP-UX 9000/750; FORTRAN/9000; HP C
VAX/VMS; VAX FORTRAN; VAX C
PC, DOS/Win95/NT; MS Powerstation Fortran; Borland Bcc32i C++/C
NeXT, Mach; Absoft Fortran; Gnu gcc
Macintosh???
Mac-ppc???
```

Porting between any pair of supported Fortran and C environments is always possible via binary/transfer conversion. The process is identical to that described above for porting between incompatible Fortran environments.

Examining SPK files

Since SPK files are binary files, you can't just open them with your favorite text editor to determine which ephemeris objects are represented in the file. Instead you need to use one of the NAIF utility programs that allow you to summarize the ephemeris contents of an SPK file. The first of these is SPACIT which was introduced above. The second is the command line utility BRIEF.

BRIEF gives a quick summary of the contents of the file and supports a wide set of summary options. SPACIT on the other hand, provides summaries that are more detailed and reflect closely the actual internal structure of the file. Unless you need the more detailed summary, you'll probably find BRIEF to be a better tool for examining the contents of an SPK file.

Meta Data in the SPK file

SPICE kernels may contain "meta" data that describe the contents, intended use, accuracy, etc. of the kernel. This meta data is called the "comments" portion of the kernel. Many SPK files contain comments that can help you decide upon the suitability of the kernel for your application. Two SPICE utilities are available for examining the comments of a binary kernel—COMMNT and SPACIT.

We've already introduced SPACIT. COMMNT is similar to SPACIT in that it too is an interactive program. However, COMMNT also allows you to modify the comments of an SPK file. Using COMMNT you can delete the comments of an SPK file, extract the comments to a text file, or append the text from some text file to the comments already present in the kernel.

If you create SPK files, we strongly recommend that you add comments to the kernel that describe who created it, expected usage of the kernel, and the expected accuracy of the position/velocity information contained in the kernel. A comment template is provided in the appendix "COMMENTS".

Warning: If you add comments to an SPK (or other binary kernel) using COMMNT, you must wait for the program to complete the task before exiting the program. Failure to wait for COMMNT to finish its work will result in irreparable corruption of the binary kernel. (See the COMMNT User's Guide [212] for details on the use of COMMNT).

Terminology

Throughout this document we shall be using terms such as reference frame, state, ephemeris time, etc. We include a brief review of these terms below.

`Reference Frame` A reference frame is a cartesian coordinate system with three axes—x, y and z. The axes are mutually orthogonal. The center of the frame is the origin of the cartesian reference system. For the reference frames in SPICE, the positions of the axes are typically defined by some observable object. For example, in the J2000 reference frame, the x-axis is defined to lie in the intersection of two planes: the plane of the Earth's equator and the plane of the Earth's orbit. The z-axis is perpendicular to the Earth's equator. The y-axis completes a right handed system. The center of the frame is typically taken to be the solar system barycenter. (Note we are not attempting to rigorously define the J2000 frame here. We are only illustrating how reference frames are defined. Many more details are required for a rigorous definition of the J2000 frame. These details are given in the SPICE document "Frames" [349].)

`State`	A state is an array of six double precision numbers. The first three numbers give the x, y, and z coordinates respectively for the position of some object relative to another object in some cartesian reference frame. The next three numbers give the velocity (dx/dt, dy/dt and dz/dt respectively) of the object with respect to the same reference frame.
`Inertial Frame`	An inertial frame, is one in which Newton's laws of motion apply. A frame whose axes are not moving with respect to the observed positions of distant galaxies and quasars approximates an inertial frame.
`Non-Inertial Frame`	A non-inertial frame is a frame that rotates with respect to the celestial background. For example a frame whose axes are fixed with respect to the features on the surface of the Earth is a non-inertial frame.
`Ephemeris Time (ET)`	Ephemeris time, ET, is the independent variable in the equations of motion that describe the positions and velocities of objects in the solar system. In SPICELIB we treat ET as a synonym for Barycentric Dynamical Time. As far as has been experimentally determined, an atomic clock placed at the solar system barycenter, would provide a faithful measure of ET.
`Seconds Past 2000`	In the SPK system times are specified as a count of seconds past a particular epoch—the epoch of the J2000 reference frame. This reference epoch is within a second or two of the the UTC epoch: 12:01:02.184 Jan 1, 2000 UTC. (See the document TIME.REQ for a more thorough discussion of the J2000 epoch). Epochs prior to this epoch are represented as negative numbers. The "units" of ET are designated in several different ways: seconds past 2000, seconds past J2000, seconds past the Julian year 2000, seconds past the epoch of the J2000 frame. All of these phrases mean the same thing and are used interchangeably throughout this document.

`SPK segment` The trajectories of objects in SPK files are represented in pieces called segments. A segment represents some arc of the full trajectory of an object. Each segment contains information that specifies the trajectory of a particular object relative to a particular center of motion in a fixed reference frame over some particular interval of time. From the point of view of the SPK system segments are the atomic portions of a trajectory.

The SPK Family of Subroutines

SPICELIB contains a family of subroutines that are designed specifically for use with SPK files. The name of each routine begins with the letters 'SPK', followed by a two- or three-character mnemonic. For example, the routine that returns the state of one body with respect to another is named SPKEZR, pronounced 'S-P-K-easier'. A complete list of mnemonics, translations, and calling sequences can be found at the end of this document.

Each subroutine is prefaced by a complete SPICELIB header, which describes inputs, outputs, restrictions, and exceptions, discusses the context in which the subroutine can be used, and shows typical examples of its use. Any discussion of the subroutines in this document is intended as an introduction: the final documentation for any subroutine is its header.

Whenever an SPK subroutine appears in an example, the translation of the mnemonic part of its name will appear to the right of the reference, in braces. We also continue with the convention of distinguishing between input and output arguments by listing input arguments in lower case and enclosed in angle brackets. For example,

```
CALL SPKLEF ( <file>, HANDLE )    { Load ephemeris file }
```

All subroutines and functions, including those whose names do not begin with 'SPK', are from SPICELIB.

Code examples will make use of the structured DO ... END DO and DO WHILE ... END DO statements supported by most Fortran compilers.

SPK readers are available to perform the following functions.

1. Determine the apparent, true, or geometric state of a body with respect to another body relative to a user specified reference frame.

2. Determine the apparent, true, or geometric state of a body with respect to an observer with having a user-supplied state.
3. Determine the geometric state of a body with respect to the solar system barycenter.
4. Determine the geometric state of a target body with respect to its center of motion for a particular segment.
5. Determine, from a list of SPK files supplied by the calling program, the files and segments needed to fulfill a request for the state of a particular body.

Computing States

SPKEZR is the most powerful of the SPK readers. It determines the apparent, true, or geometric state of one body (the target) as seen by a second body (the observer) relative to a user specified reference frame.

```
CALL SPKEZR ( <targ>,  <et>, <frame>,
.             <aberr>, <obs>,
.             STATE,    LT         ) { Easier state }
```

The subroutine accepts five inputs—target body, epoch, reference frame, aberration correction type, and observing body—and returns two outputs—state of the target body as seen from the observing body, and one-way light-time from the target body to the observing body.

The target body, observing body and frame are identified by strings that contain the names of these items. For example, to determine the state of Triton as seen from the Voyager-2 spacecraft relative to the J2000 reference frame

```
CALL SPKEZR ( 'TRITON', ET, 'J2000', ABERR,
.             'VOYAGER-2',   STATE,  LT )     { Easier state }
```

By definition, the ephemerides in SPK files are continuous: the user can obtain states at any epoch within the interval of coverage. Epochs are always specified in ephemeris seconds past the epoch of the J2000 reference system (Julian Ephemeris Date 2451545.0) For example, to determine the state of Triton as seen from Voyager-2 at Julian Ephemeris Date 2447751.8293,

```
ET = ( 2447751.8293D0 - J2000() ) * SPD()

CALL SPKEZR ( 'TRITON', ET, 'J2000', <aberr>,
.             'VOYAGER-2',   STATE,  LT )     { Easier state }
```

where the function J2000 returns the epoch of the J2000 frame (Julian Ephemeris Date 2451545.0) and the function SPD returns the number of seconds per Julian day (86400.0).

The ephemeris data in an SPK file may be referenced to a number of different reference frames. States returned by SPKEZR do not have to be referenced to any of these "native" frames. The user can specify that states are to be returned in any of the frames recognized by the frame subsystem. For example, to determine the state of Triton as seen from Voyager-2, referenced to the J2000 ecliptic reference frame,

```
       CALL SPKEZR ( 'TRITON', ET, 'J2000ECLIP', ABERR,
     .               'VOYAGER-2',   STATE,       LT )    { Easier state }
```

SPKEZR returns apparent, true, or geometric states depending on the value of the aberration correction type flag ABERR.

Apparent states are corrected for planetary aberration, which is the composite of the apparent angular displacement produced by motion of the observer (stellar aberration) and the actual motion of the target body (correction for light-time). True states are corrected for light-time only. Geometric states are uncorrected.

Instead of using the potentially confusing terms 'true' and 'geometric' to specify the type of state to be returned, SPKEZR requires the specific corrections to be named. To compute apparent states, specify correction for both light-time and stellar aberration: 'LT+S'. To compute true states, specify correction for light-time only: 'LT'. To compute geometric states, specify no correction: 'NONE'.

In all cases, the one-way light-time from the target to the observer is returned along with the state.

The Computation of Light Time

The light time corrected states returned by the SPK system are simply the 6-vector difference

```
     TARGET_SSB ( ET - LT )   - OBSERVER_SSB ( ET )
```

where TARGET_SSB and OBSERVER_SSB give the position of the target and observer relative to the solar system barycenter. LT is the unique number that satisfies:

```
     | TARGET_SSB ( ET - LT )   - OBSERVER_SSB ( ET ) |
```

```
LT =  --------------------------------------------------
                          Speed of Light
```

Where

```
| STATE |
```

refers to the length of the position component of a state vector.

(Note that the velocity portion of the state returned is simply the difference in the velocity components of

```
TARGET_SSB ( ET - LT ) and OBSERVER_SSB ( ET )
```

This is NOT the derivative of the light time corrected position because this does not take into account the time derivative of LT.)

Mathematically, LT can be computed to arbitrary precision via the following algorithm:

```
LT_0 = 0


        | TARGET_SSB ( ET - LT_(i-1) ) - OBSERVER_SSB ( ET ) |
LT_i =  ------------------------------------------------------
                          Speed of Light


(for i = 1, 2, 3 ... )
```

It can be shown that the sequence LT_0, LT_1, LT_2, ... converges to LT geometrically. Moreover, it can be shown that the difference between LT_i and LT satisfies the following inequality.

```
                             i+1
 LT - LT_i  < LT * ( V/C )
```

where V is the speed of the target body with respect to the solar system barycenter and C is the speed of light. Let's examine the error we make if we use LT_1 as an approximation for LT.

For nearly all objects in the solar system V is less than 60 km/sec. The value of C is 300000 km/sec. Thus the one iteration solution for LT has a potential relative error of not more than 4*10D-8. This is a potential light time error of approximately 2*10D-5 seconds per astronomical unit of distance separating the observer and target. Thus as long as the observer and target are separated by less than 50 Astronomical Units the error in the light time returned using option 'LT' is less than 1 millisecond.

For this reason, we use LT_1 to approximate LT when you request a light time corrected state by setting the aberration correction argument in SPKEZR to 'LT' or 'LT+S'.

You can make SPKEZR perform a better approximation to LT by requesting that it compute a "converged Newtonian" value for LT. To do this set the aberration correction to 'CN' or 'CN+S'. SPKEZR will then return LT_3 as the approximation for light time. The maximum error in LT_3 is less than a nanosecond for any observer/target pair in the solar system.

However, you should note that this is a purely Newtonian approximation to the light time. To model the actual light time between target and observer one must take into account effects due to General relativity. These may be as high as a few hundredths of a millisecond for some objects.

The routines in the SPK family do not attempt to perform either general or special relativistic corrections in computing the various aberration corrections. For many applications relativistic corrections are not worth the expense of added computation cycles. If, however, your application requires these additional corrections we suggest you consult the astronomical almanac (page B36) for a discussion of how to carry out these corrections.

Light Time Corrected Non-Inertial States

When we observe a distant object, we don't see it as it is at the moment of observation. We see it as it was when the photons we have sensed were emitted by or reflected from the object. Thus when we look at Mars through a telescope, we see it not as it is now, but rather as it was one "light-time" ago. This is true not only for the position of Mars, but for its orientation as well.

Suppose that a large balloon has been launched into the Martian atmosphere and we want to determine the Mars bodyfixed state of the balloon as seen from Earth at the epoch ET. We need to determine both the light time corrected position of the balloon, and the light time corrected orientation of Mars. To do this we compute two light times. The light time to the center of the Mars bodyfixed frame (i.e. the center of Mars) and the light time to the balloon. Call the light time to the center of the Mars frame LT_F and call the light time to the balloon LT_T. The light time corrected state of the balloon relative to the Mars bodyfixed frame is the location of the balloon at ET - LT_T in the bodyfixed frame of Mars as oriented at ET - LT_F.

SPKEZR carries out all of these computations automatically. In this case the computation would be computed by a subroutine call similar to this:

```
CALL SPKEZR ( 'Mars_balloon', <et>, 'IAU_MARS', 'LT', 'EARTH',
```

```
.                STATE, LT  )
```

SPKEZR uses the following rules when computing states.

1. When no corrections are requested from SPKEZR (ABCORR = 'NONE'), the state of the target is determined at the request time ET and is represented in the specified reference frame as it is oriented at time ET.
2. When light time corrections are requested from SPKEZR (ABCORR = 'LT'), two light times are determined: LT_F the light time to the center of the specified reference frame, and LT_T the light time to the target. The state of the target is given as it was at ET - LT_T in the frame as it was oriented at ET - LT_F.
3. When light time and stellar aberrations are requested from SPKEZR (ABCORR = 'LT+S'), both LT_F and LT_T are again computed. The state of the target at ET - LT_T is corrected for stellar aberration and represented in the reference frame as it was oriented at ET - LT_F.

In the actual implementation of SPKEZR a few short cuts are taken. When light time requested states relative to an inertial frame are requested, the orientation of the frame is not corrected for light time. The orientation of an inertial frame at ET - LT_F is the same as the orientation of the frame at ET. Computations involving inertial frames take advantage of this observation and avoid redundant computations.

An example

Here we illustrate how you could use SPKEZR together with other SPICELIB routines to determine if at a particular epoch ET the Mars Global Surveyor spacecraft is occulted by Mars.

We will need the lengths of the axes of the triaxial ellipsoid that is used to model the surface of Mars. The SPICELIB routine BODVAR will retrieve this information from a loaded PCK file. Note that BODVAR uses the NAIF ID code for Mars (499) to retrieve the lengths of the axes.

```
CALL BODVAR ( 499, 'RADII', NVALS, AXES )

A = AXES(1)
B = AXES(2)
C = AXES(3)
```

Next we compute the state of Mars relative to Earth and the state of MGS relative to Earth in the Mars bodyfixed frame.

```
 CALL SPKEZR ( 'MARS',  ET, 'IAU_MARS', 'LT+S', 'EARTH',
.               MARSST, LT )
 CALL SPKEZR ( 'MGS',   ET, 'IAU_MARS', 'LT+S', 'EARTH',
.               MGSST,  LT ) {Easier State}
```

Compute the apparent position of the Earth relative to Mars in the apparent Mars bodyfixed frame. This means simply negating the components of MARSST. The SPICELIB routine VMINUS carries out this task.

```
CALL VMINUS ( MARSST, ESTATE )
```

Determine if the line of sight from Earth to MGS intersects the surface of Mars. The SPICELIB routine SURFPT will find this intersection point if it exists.

```
CALL SURFPT ( ESTATE, MGSST, A, B, C, POINT, FOUND )
```

Finally, if a point of intersection was found, was it between the Earth and the MGS spacecraft. To find out we can compare the distances between the intersection point and the spacecraft. The SPICELIB function VNORM computes the length of the vector from Earth to MGS. The function VDIST computes the distance between the point and the Earth.

```
IF ( FOUND ) THEN
   BETWN = VDIST( ESTATE, POINT ) .LT. VNORM ( MGSST )
ELSE
   BETWN = .FALSE.
END IF

IF ( BETWN ) THEN
   WRITE (*,*) 'MGS is behind Mars'
ELSE
   WRITE (*,*) 'MGS is not behind Mars'
END IF
```

Integer ID Codes Used in SPK

Low level SPK software uses integer codes to identify ephemeris objects, reference frames and data representation, etc. At low levels of the SPICE system only integer codes are used to communicate information about objects. To some extent, these codes are a historical artifact in the design of the SPICE system. Nevertheless, these integer codes provide economies in the development of SPICE software.

High-level SPICE software uses names (character strings) to refer to the various SPICE objects and translates between names and integer codes. Thus to some extent you can disregard the integer codes used by the SPICE internals. However, occasionally, due to the introduction of new ephemeris objects, the name translation software will be unable to find a name associated with an ID code. To retrieve states for such an object you will need to use the integer code for the object in question. If you are using SPKEZR, you can supply this integer code as a quoted string. For example the following two subroutine calls will both return the state of TRITON as seen from Voyager-2. (The NAIF integer code for TRITON is 801; the NAIF integer code for Voyager 2 is -32).

```
CALL SPKEZR ( 'TRITON', ET, 'J2000ECLIP', ABERR,
.             'VOYAGER-2',   STATE,       LT )    { Easier state }


CALL SPKEZR ( '801',   ET, 'J2000ECLIP', ABERR,
.             '-32',        STATE,       LT )    { Easier state }
```

Consult the NAIF IDS Required Reading file for the current list of body codes recognized by the NAIF Toolkit software.

SPKEZ and SPKGEO

SPKEZR relies upon two lower level routines that may be useful under certain circumstances.

The routine SPKEZ performs the same functions as SPKEZR. The only difference is the means by which objects are specified. SPKEZ requires that the target and observing bodies be specified using the NAIF integer ID codes for those bodies.

```
SPKEZ ( <targ_id>, <et>, <frame>, <corr>, <obj_id>,
         STATE,    LT )  { SPK Easy }
```

The NAIF-ID codes for ephemeris objects are listed in the NAIF_IDS required reading file.

SPKEZ is useful in those situations when you ID codes for objects stored as integers. There is also a modest efficiency gain when using integer ID codes instead of character strings to specify targets and observers.

The routine SPKGEO returns only geometric (uncorrected) states. The following two subroutine calls are equivalent.

```
CALL SPKEZ  ( <targ_id>, <et>, <frame>,
.             'NONE',    <obj_id>,
.             STATE,     LT )              {SPK Easy}
```

```
 CALL SPKGEO ( <targ_id>, <et>, <frame>,
               <obj_id>,
.              STATE,     LT )              {SPK Geometric }
```

SPKGEO involves slightly less overhead than does SPKEZ and thus may be marginally faster than calling SPKEZ.

Loading Files

Note that SPKEZR, SPKEZ and SPKGEO do not require the name of an SPK file as input. These routines rely on a second routine, SPKLEF, to maintain a database of ephemeris files. Your application program indicates which files are to be used by passing their names to SPKLEF.

```
DO I = 1, N
   CALL SPKLEF ( ephem(I), HANDLE(I) )  { Load ephemeris file }
END DO
```

SPKLEF returns a DAF file handle for each file, which may be used to access the file directly using DAF subroutines. Once an SPK file has been loaded, it may be accessed by SPKEZR.

In general, a state returned by SPKEZR is built from several more primitive states. Consider the following diagram, which shows some of the states that might be needed to determine the state of the Galileo spacecraft as seen from Earth:

```
         Jupiter_Barycenter --- Europa
         /                       \
        /                         \
       /                         Spacecraft
      /
     /
    /
   /
SSB
   \
    \
     \
     EMB --- Earth
```

(SSB and EMB are the solar system and Earth-Moon barycenters.)

Each state is computed from a distinct segment. The segments may reside in a single SPK file, or may be contained in as many as five separate files. For example, the segments needed to compute the Earth-spacecraft state shown above might come from the following set of files:

```
CALL SPKLEF ( 'barycenters.bsp',    H(1) )  { Load ephemeris file }
CALL SPKLEF ( 'planet-centers.bsp', H(2) )  { Load ephemeris file }
CALL SPKLEF ( 'satellites.bsp',     H(3) )  { Load ephemeris file }
CALL SPKLEF ( 'spacecraft.bsp',     H(4) )  { Load ephemeris file }
```

or from the following set:

```
CALL SPKLEF ( 'earth.bsp',      H(1) )      { Load ephemeris file }
CALL SPKLEF ( 'jupiter.bsp',    H(2) )      { Load ephemeris file }
CALL SPKLEF ( 'spacecraft.bsp', H(3) )      { Load ephemeris file }
```

Data Precedence

An SPK file may contain any number of segments. A single file may contain overlapping segments: segments containing data for the same body over a common interval. When this happens, the latest segment in a file supersedes any competing segments earlier in the file. Similarly, the latest file loaded supersedes any earlier files. In effect, several loaded files become equivalent to one large file.

Unloading Files

The number of SPK files that may be open at any one time is limited. For example, some operating systems limit the total number of files that may be open at one time to 20. Consequently, your application program may need to unload some SPK files to make room for others. An SPK file may be unloaded by supplying its handle to subroutine SPKUEF. The sequence of statements shown below,

```
CALL SPKLEF ( 'file.a', HA )     { Load ephemeris file }
CALL SPKLEF ( 'file.b', HB )     { Load ephemeris file }
CALL SPKLEF ( 'file.c', HC )     { Load ephemeris file }
CALL SPKUEF (           HB )     { Unload ephemeris file }
CALL SPKLEF ( 'file.d', HD )     { Load ephemeris file }
CALL SPKUEF (           HC )     { Unload ephemeris file }
```

is equivalent to the following (shorter) sequence:

```
CALL SPKLEF ( 'file.a', HA )     { Load ephemeris file }
CALL SPKLEF ( 'file.d', HD )     { Load ephemeris file }
```

Loading Auxiliary Files

Prior to the inclusion of non-inertial frames in the SPK system, the states of objects computed by the SPK system required only that you load the correct SPK files and call the correct subroutines. The inertial frame transformations needed for converting from one inertial frame to another are "hard wired" into the SPICE system. The transformations are part of the object code of the SPICELIB library—no additional data need be supplied to compute these transformations. This approach to carrying out inertial frame transformations was chosen because the transformations are compactly represented and do not change as the result of further observations. They are essentially definitions.

On the other hand, the orientation of non-inertial frames with respect to other frames are almost always the result of observation. They are improved and extended as further observations are made. For some of these frames (such as spacecraft fixed frames) vary large data sets are needed to express the orientation of the frame with respect to other frames. Frame transformations that are a function of time and require megabytes of data are not suitable for encapsulation in FORTRAN source code. As a result, in the SPICE system, the computation of non-inertial frame transformations depends upon data stored in other SPICE kernels. If you request states relative to a non-inertial frame or use ephemerides that are represented relative to non-inertial frames you must load additional SPICE kernels. The method by which an auxiliary kernel is loaded depends upon the type of the kernel.

There are currently four classes of reference frames that are supported by the SPICE system. We give a brief overview of these frames here. For a more thorough discussion of the various types of frames see the recommended reading file "FRAMES.REQ."

Inertial frames

> Inertial frames are built into the SPICE system. You don't need to do anything to make their definitions available to your program. Inertial frames have NAIF ID codes whose values are in the range from 1 to 10000.

PCK frames

> PCK frames are bodyfixed frames. The orientation of a PCK frame is always expressed relative to an inertial frame. The relationship between a PCK frame and its associated inertial frame is provided by a PCK kernel. PCK frames have ID codes between 10000 and 100000. There are two types of PCK kernels—binary and text. Binary PCK kernels

are loaded (and unloaded) in a fashion analogous to the loading and unloading of SPK files. To load a binary PCK file

```
CALL PCKLOF ( <file>, HANDLE )   {PCK Load
                                  Orientation File}
```

To unload a binary PCK file

```
CALL PCKUOF ( <handle> )         {PCK Unload
                                  Orientation File}
```

text based PCK files are loaded via the routine LDPOOL. Text based PCK files can not be conveniently unloaded (See the document Kernels for further discussion on the manipulation of text based kernels.)

```
CALL LDPOOL ( <file> ) {Load Kernel Pool}
```

CK Frames

CK frames are frames that are defined relative to a spacecraft structure. The orientation of the structure is provided through a binary SPICE kernel called a C-kernel. The ID codes for C-kernel frames are negative and usually less than -999. A C-kernel frame may be defined relative to any other kind of frame. (Most existing C-kernels are defined relative to inertial frames.)

C-kernels are loaded and unloaded via a routine similar to the routines used load and unload SPK kernels. To load a C-kernel

```
CALL CKLPF ( <file>, HANDLE )   {CK Load Pointing File}
```

To unload a C-kernel

```
CALL CKUPF ( <handle> )         {CK Unload Pointing File}
```

The times used to represent C-kernels are spacecraft clock times—not ET. The relationship between ET and spacecraft clock times is stored in a SPICE text kernel called a spacecraft clock kernel—usually abbreviated as SCLK (ess-clock) kernel. To retrieve states relative to a CK frame you need to make the relationship between ET and the spacecraft clock available to your program by loading the appropriate SCLK kernel. SCLK kernels are loaded via the routine LDPOOL.

```
CALL LDPOOL ( <sclk_file_name> )  {Load Kernel Pool}
```

TK Frames

TK frames (short for Text Kernel frames) are frames that are defined via a SPICE text kernel. These frames can be transformed to another reference frame via a constant rotation matrix. Typical examples are topocentric frames and instrument frames. TK frames are loaded via the routine LDPOOL.

```
CALL LDPOOL ( <TK_frame_file> )  {Load Kernel Pool}
```

In addition to the files mentioned above, it may be necessary to load a "frame definition" file along with the one of the SPICE kernels listed above. (If the producer of the file has done his or her homework this step should be unnecessary.) The frame definition file is a SPICE text kernel that specifies the type of the frame, the center of the frame, the name of the frame, and its ID code. (See FRAMES.REQ for more details concerning frame definitions.)

As is evident from the above discussion, the use of non-inertial frames requires more data management on the part of the user of the SPICE system. However, this data management problem is not a new problem. In previous versions of the SPICE system the same kernels would have been required. Moreover, in previous versions of the SPICE system, you would have been required to perform all non-inertial transformations in your own code. With the inclusion of non-inertial frames in the SPK system, we have relieved you of some of the tasks associated with non-inertial frames.

SPK File Structure

An SPK file is made up of one or more data "segments" and a "comment" area. These components are described below.

Segments–The Fundamental SPK Building Blocks

An SPK file contains one or more "segments." Each segment contains ephemeris data sufficient to compute the geometric state (position and velocity) of one solar system body (the 'target') with respect to another (the 'center') at any epoch throughout some finite interval of time.

Either body may be a spacecraft, a planet or planet barycenter, a satellite, a comet, an asteroid, a tracking station, a roving vehicle, or an arbitrary point for which an ephemeris has been calculated. Each body in the solar system is associated with a unique integer code. A list of names and codes for the planets, major satellites, spacecraft, asteroids and comets can be found in the document NAIF_IDS.REQ

The states computed from the ephemeris data in a segment must be referenced to a single, recognized reference frame.

The data in each segment are stored as an array of double precision numbers. The summary for the array, called a 'descriptor', has two double precision components:

1. The initial epoch of the interval for which ephemeris data are contained in the segment, given in ephemeris seconds past Julian year 2000.
2. The final epoch of the interval for which ephemeris data are contained in the segment, given in ephemeris seconds past Julian year 2000.

The descriptor has six integer components:

1. The NAIF integer code for the target.
2. The NAIF integer code for the center.
3. The NAIF integer code for the reference frame.
4. The integer code for the representation (type of ephemeris data).
5. The initial address of the array.
6. The final address of the array.

In addition to a descriptor, each array also has a name. The name of each array may contain up to 40 characters. This space may be used to store a brief description of the segment. For example, the name may contain pedigree information concerning the segment or may contain the name of the object whose position is recorded in the segment.

The Comment Area

Preceding the 'segments', the Comment Area provides space in the SPK file for storing textual information besides what is written in the array names. Ideally, each SPK file would contain internal documentation that describes the origin, recommended use, and any other pertinent information about the data in that file. For example, the beginning and ending epochs for the file, the names and NAIF integer codes of the bodies included, an accuracy estimate, the date the file was produced, and the names of the source files used in making the SPK file could be included in the Comment Area.

The utility programs COMMNT and SPACIT may be used to examine and manipulate the comments in an SPK file. In addition to these utilities, SPICELIB provides a family of subroutines for handling this Comment Area.

The name of each routine in this family begins with the letters 'SPC' which stand for 'SPk and Ck' because this feature is common to both types of files. The SPC software provides the ability to add, extract, read, and delete comments and convert commented files from binary format to SPICE transfer format and back to binary again.

The SPC routines and their functions are described in detail in the SPC Required Reading.

SPK Data Types

The fourth integer component of the descriptor—the code for the representation, or 'data type'—is the key to the SPK format.

For purposes of determining the segment best suited to fulfill a particular request, all segments are treated equally. It is only when the data in a segment are to be evaluated—when a state vector is to be computed—that the type of data used to represent the ephemeris becomes important.

Because this step is isolated within a single low-level reader, SPKPVN, new data types can be added to the SPK format without affecting application programs that use the higher level readers. SPKPVN is designed so that the changes required to implement a new data type are minimal.

There are no real limits on the possible representations that can be used for ephemeris data. Users with access to data suitable for creating an ephemeris may choose to invent their own representations, adapting SPKPVN accordingly. (We recommend that you consult with NAIF prior to implementing an new data type.)

The data types currently supported by SPICELIB software are listed under "Supported Data Types" later in this document.

Lower-level Readers

When computing states, SPKEZR should be sufficient to handle the needs of most users. However, it is possible to exercise more direct control over the way states are computed. In this section we discuss means by which the user of the SPK system can take more direct control over the computation of states.

As noted above, SPKEZR is identical to SPKEZ except that it uses the names of objects as inputs instead of integer ID codes. Indeed, SPKEZR "looks up" the ID codes associated with the named objects, and then calls SPKEZ using these ID codes.

SPKEZ computes apparent and true states using two readers of slightly less power—SPKSSB and SPKAPP.

SPKSSB returns the state of a body with respect to the solar system barycenter. SPKEZ uses it to compute the state of the observer relative to an inertial reference frame.

The second, SPKAPP, returns the state of a target body as seen from an observer. Where SPKEZ requires the integer code for an observer, SPKAPP requires the actual state of the observer with respect to the solar system barycenter relative to an inertial reference frame. A single call to SPKEZ,

```
CALL SPKEZ ( 801, ET, 'J2000', 'LT+S', -32, STARG, LT )  { Easy
                                                           state }
```

is equivalent to a pair of calls to SPKSSB and SPKAPP:

```
CALL SPKSSB ( -32,
               ET,
              'J2000',
               STOBS    )        { Solar system barycenter }

CALL SPKAPP (  801,
               ET,
              'J2000',
               STOBS,
              'LT+S',
               STARG,
               LT       )        { Apparent state }
```

(It is important to note that this equivalence breaks down if SPKEZ is requested to return states relative to a non-inertial frame. When non-inertial apparent or true states are requested, SPKEZ first computes an inertial apparent or true state. After the inertial state has been computed, it is transformed to a non-inertial frame taking into account the light time delay from the center of that frame.)

One possible advantage of using SPKAPP directly is the ability to place an observer somewhere other than at the center of a body (for example, at a specified location on the surface of the Earth).

When computing uncorrected, that is, geometric states, SPKEZ does not need to compute the state of the target and observer relative to the solar system barycenter, but only relative to the first common center of motion of those two bodies. SPKEZ calls SPKGEO to compute geometric states.

Using SPKGEO instead of the combination SPKSSB and SPKAPP as above prevents possible round-off error, may reduce the number of file reads, and may require less data. For example, if SPK ephemeris data for a spacecraft relative to a planet has been loaded, but the ephemeris data for that planet relative to the solar system barycenter is not available, SPKGEO can still compute the state of the spacecraft relative to the planet, whereas the combination SPKSSB and SPKAPP would be unsuccessful at computing that state.

Primitive States

At the lowest level, it is possible to compute states without combining them at all. Given the handle and descriptor for a particular segment, subroutine SPKPVN returns a state from that segment directly.

```
CALL SPKPVN( <handle>,
             <descr>,
             <et>,
             REF,
             STATE,
             CENTER   )  { Position, velocity, native frame }
```

SPKPVN is the most basic SPK reader. It returns states relative to the frame in which they are stored in the SPK file. It does not rotate or combine them: it returns a state relative to the center whose integer code is stored in the descriptor for the segment. This state is relative to the frame whose integer ID code is also stored in the descriptor of the segment. The user is responsible for using that state correctly.

The user is also responsible for using DAF subroutines to determine the particular file and segment from which each state is to be computed.

Note that to use the state returned by SPKPVN in any frame other than the "native frame" of the segment, you must convert the state to the frame of interest. A second low level routine SPKPV can be used to perform the state transformations for you. The calling sequence for SPKPV is identical to that for SPKPVN. However, in the case of SPKPV the reference frame is an input instead of an output argument.

```
CALL SPKPV ( <handle>,
             <descr>,
             <et>,
             <ref>,
             STATE,
             CENTER   )  { Position, velocity }
```

Thus using SPKPV instead of SPKPVN allows you to avoid the details of converting states to the frame of interest.

If files have been loaded by previous calls to SPKLEF, it is possible to use the same segments that would normally be used by SPKEZR, SPKEZ, SPKSSB, SPKAPP, and SPKGEO. Subroutine SPKSFS selects, from the database of loaded files, the file handle and segment descriptor for the segment best suited to the request. If two segments from different files are suitable, SPKSFS selects the one from the file that was loaded later. If two segments from the same file are suitable, SPKSFS selects the one that is stored later in the file. The call

```
CALL SPKSFS ( <801>,
              <et>,
              HANDLE,
              DESCR,
              SEGNAM,
              FOUND )     { Select file and segment }
```

returns the handle, descriptor, and segment name for the latest segment containing data for Triton at the specified epoch. SPKSFS maintains a buffer of segment descriptors and segment names, so it doesn't waste time searching the database for bodies it already knows about.

Examples of Using SPK Readers

Example 1: Computing Latitude and Longitude

The next several sections present sample programs to show how the SPK readers can be used to compute state vectors, and how those vectors can be used to compute derived quantities.

All subroutines and functions used in the examples are from SPICELIB. The convention of expanding SPK subroutine names will be dropped for these examples.

The first example program computes the planetocentric latitude and longitude of the sub-observer point on a target body for any combination of observer, target, and epoch. (Note that planetocentric coordinates differ from planetographic and cartographic coordinates in that they are always right-handed, regardless of the rotation of the body. Also note that for this example we define the sub-observer point to be the point on the "surface" of the target that lies on the ray from the center of the target to the observer.)

```
      PROGRAM LATLON

C
```

```
C     SPICELIB functions
C
      DOUBLE PRECISION  DPR

C
C     Variables
C
      CHARACTER*(32)         TIME
      CHARACTER*(32)         OBS
      CHARACTER*(32)         TARG

      DOUBLE PRECISION       ET
      DOUBLE PRECISION       LAT
      DOUBLE PRECISION       LON
      DOUBLE PRECISION       LT
      DOUBLE PRECISION       RADIUS
      DOUBLE PRECISION       STATE  (   6 )
      DOUBLE PRECISION       TIBF   ( 3,3 )

      INTEGER                H      (  13 )

C
C     Load constants into the kernel pool. Two files are
C     needed. The first (`time.ker') contains the dates
C     of leap seconds and values for constants needed to
C     compute the difference between UTC and ET at any
C     epoch. The second (`pck.ker') contains IAU values
C     needed to compute transformations from inertial
C     (J2000) coordinates to body-fixed (pole and prime
C     meridian) coordinates for the major bodies of the
C     solar system. (These files, or their equivalents,
C     are normally distributed along with SPICELIB.)
C
      CALL CLPOOL
      CALL LDPOOL ( 'time.ker'  )
      CALL LDPOOL ( 'pck.ker'   )

C
C     Several ephemeris files are used. Most contain data for
C     a single planetary system (`jupiter.ker', `saturn.ker',
C     and so on). Some contain data for spacecraft (`vgr1.ker',
C     `mgn.ker').
C
      CALL SPKLEF ( 'MERCURY.BSP', H(1)  )
      CALL SPKLEF ( 'VENUS.BSP',   H(2)  )
      CALL SPKLEF ( 'EARTH.BSP',   H(3)  )
      CALL SPKLEF ( 'Mars.BSP',    H(4)  )
      CALL SPKLEF ( 'JUPITER.BSP', H(5)  )
      CALL SPKLEF ( 'SATURN.BSP',  H(6)  )
      CALL SPKLEF ( 'URANUS.BSP',  H(7)  )
      CALL SPKLEF ( 'NEPTUNE.BSP', H(8)  )
      CALL SPKLEF ( 'PLUTO.BSP',   H(9)  )
      CALL SPKLEF ( 'VGR1.BSP',    H(10) )
      CALL SPKLEF ( 'VGR2.BSP',    H(11) )
      CALL SPKLEF ( 'MGN.BSP',     H(12) )
      CALL SPKLEF ( 'GLL.BSP',     H(13) )

C
C     Inputs are entered interactively. The user enters three
```

```
C     items: the name for the observer , the name
C     for the target, and the UTC epoch at which the
C     sub-observer point is to be computed (a free-format string).
C
C     The epoch must be converted to ephemeris time (ET).
C
      DO WHILE ( .TRUE. )

         CALL PROMPT ( 'Observer? ', OBS  )
         CALL PROMPT ( 'Target?   ', TARG )
         CALL PROMPT ( 'Epoch ?   ', TIME )

         CALL STR2ET ( TIME, ET )
         FRAME = 'IAU_' // TARG

C
C        Compute the true state (corrected for light-time)
C        of the target as seen from the observer at the
C        specified epoch in the target fixed reference frame.
C
         CALL SPKEZR ( TARG, ET, FRAME, 'LT', OBS, STATE, LT )

C
C        We need  the vector FROM the target TO the observer
C        to compute latitude and longitude. So reverse it.
C
         CALL VMINUS ( STATE, STATE )
C
C        Convert from rectangular coordinates to latitude and
C        longitude, then from radians to degrees for output.
C
         CALL RECLAT ( STATE, RADIUS, LON, LAT )

         WRITE (*,*)
         WRITE (*,*) 'Sub-observer latitude (deg): ', LAT * DPR()
         WRITE (*,*) '             longitude     : ', LON * DPR()
         WRITE (*,*)
         WRITE (*,*) 'Range to target (km)       : ', RADIUS
         WRITE (*,*) 'Light-time (sec)           : ', LT
         WRITE (*,*)

C
C        Get the next set of inputs.
C

      END DO

      END
```

Example 2: Faster Latitude and Longitude

The second example computes the same quantities as the first. However, this program assumes that the observer is always the Magellan spacecraft and the target is always Venus. It also ignores light-time from the planet to the spacecraft. These restrictions allow a more primitive reader, SPKPV, to be substituted for the more general reader, SPKEZR.

SPKPV returns this same state as SPKEZR, but avoids much of the overhead associated with SPKEZR—making the second program somewhat faster than the first.

However, the second program is much less flexible. For example, if the spacecraft ephemeris contains cruise data (describing the motion of the spacecraft relative to the solar system barycenter instead of the planet center), the program would produce incorrect results.

Furthermore, the program cannot easily be generalized to work for other orbiters. The motion of the Galileo spacecraft, for instance, would normally be known relative to the Jupiter barycenter, not to the planet itself.

```
      PROGRAM FASTER

C
C     SPICELIB functions
C
      DOUBLE PRECISION  DPR

C
C     Definitions
C
      INTEGER           MGN
      PARAMETER       ( MGN = -18 )

      INTEGER           VENUS
      PARAMETER       ( VENUS = 299 )

C
C     Variables
C
      CHARACTER*(40)    SEGNAM
      CHARACTER*(32)    TIME

      DOUBLE PRECISION  DESCR  (   5 )
      DOUBLE PRECISION  ET
      DOUBLE PRECISION  LAT
      DOUBLE PRECISION  LON
      DOUBLE PRECISION  RADIUS
      DOUBLE PRECISION  STATE  (   6 )
      DOUBLE PRECISION  TIBF   ( 3,3 )

      INTEGER           CENTER
      INTEGER           HANDLE

      LOGICAL           FOUND

C
C     Load constants into the kernel pool. Two files are
C     needed. The first ('time.ker') contains the dates
C     of leap seconds and values for constants needed to
C     compute the difference between UTC and ET at any
```

```
C     epoch. The second (`venus.ker') contains IAU values
C     needed to compute the transformation from inertial
C     (J2000) coordinates to body-fixed (pole and prime
C     meridian) coordinates for Venus.
C
      CALL CLPOOL
      CALL LDPOOL ( 'TIME.KER'  )
      CALL LDPOOL ( 'VENUS.KER' )

C     Only one ephemeris file is needed. This contains data for
C     the Magellan spacecraft relative to Venus. The states of
C     other bodies are not needed.
C
      CALL SPKLEF ( 'MGN.BSP', HANDLE )
C
C     Inputs are entered interactively. The user enters only the
C     epoch at which the sub-spacecraft point is to be computed
C     (a free-format string).
C
C
C     The epoch must be converted to ephemeris time (ET).
C
      DO WHILE ( .TRUE. )

         CALL PROMPT ( 'Epoch? ', TIME )
         CALL STR2ET (  TIME,     ET   )

C
C        Because the ephemeris file might contain many segments
C        for the spacecraft, we need to select the proper segment
C        each time a state is computed.
C
C        For now, we will assume that a segment is found. A more
C        careful program would check this each time. (If FOUND is
C        ever false, the data needed to respond to the user's
C        request are not available, and the program should take
C        appropriate action.)
C
         CALL SPKSFS ( MGN, ET, HANDLE, DESCR, SEGNAM, FOUND )

C
C        Compute the geometric state (uncorrected for light-time)
C        of the spacecraft as seen from the planet. We can compute
C        this directly because light-time is being ignored.
C        Do all computations in J2000 coordinates,
C
C        For now, we will assume that CENTER is always Venus
C        (2 or 299). A more careful program would check this
C        each time.
C
         CALL SPKPV (HANDLE, DESCR, ET, 'IAU_VENUS', STATE, CENTER)

C
C        Convert from rectangular coordinates to latitude and
C        longitude, then from radians to degrees for output.
C
         CALL RECLAT ( STATE, RADIUS, LON, LAT )

         WRITE (*,*)
```

```
            WRITE (*,*) 'Sub-spacecraft latitude (deg): ', LAT * DPR()
            WRITE (*,*) '               longitude     : ', LON * DPR()
            WRITE (*,*)
C
C           Get the next input epoch.
C

         END DO

         END
```

Example 3: Occultation or Transit

The third example determines epochs if one target body (spacecraft, planet, or satellite) is occulted by or in transit across another target body as seen from an observer at a user specified epoch. It is similar in both form and generality to the first example.

```
         PROGRAM OCCTRN

C
C     SPICELIB functions
C
      DOUBLE PRECISION      SUMAD
      DOUBLE PRECISION      VNORM
      DOUBLE PRECISION      VSEP

C
C     Variables
C
      CHARACTER*(32)        TIME
      CHARACTER*(32)        OBS
      CHARACTER*(32)        TARG    ( 2   )

      DOUBLE PRECISION      AVG
      DOUBLE PRECISION      D       ( 2   )
      DOUBLE PRECISION      ET
      DOUBLE PRECISION      R       ( 2   )
      DOUBLE PRECISION      RADII   ( 3   )
      DOUBLE PRECISION      S       ( 6,2 )
      DOUBLE PRECISION      SEP

      INTEGER               I
      INTEGER               T       (  2 )
      INTEGER               H       ( 13 )

      LOGICAL               FOUND

C
C     Load constants into the kernel pool. Two files are
C     needed. The first ('time.ker') contains the dates
C     of leap seconds and values for constants needed to
C     compute the difference between UTC and ET at any
C     epoch. The second ('radii.ker') contains values
```

```
C     for the tri-axial ellipsoids used to model the major
C     major bodies of the solar system.
C
      CALL CLPOOL
      CALL LDPOOL ( 'TIME.KER'  )
      CALL LDPOOL ( 'RADII.KER' )

C
C     Several ephemeris files are needed. Most contain data for
C     a single planetary system (`jupiter.ker', `saturn.ker',
C     and so on). Some contain data for spacecraft (`vgr1.ker',
C     `mgn.ker').
C
      CALL SPKLEF ( 'MERCURY.BSP', H(1)  )
       .
       .
      CALL SPKLEF ( 'GLL.BSP',     H(13) )

C
C     Inputs are entered interactively. The user enters four
C     items: the code for the observer (an integer), the codes
C     for two target bodies (integers), and the epoch at which
C     check for occultation or transit is to be computed
C     (a free-format string).
C
C     The epoch must be converted to ephemeris time (ET).
C
      DO WHILE ( .TRUE. )

         CALL PROMPT ( 'Observer? ', OBS     )
         CALL PROMPT ( 'Target 1? ', TARG(1) )
         CALL PROMPT ( 'Target 2? ', TARG(2) )
         CALL PROMPT ( 'Epoch ?   ', TIME    )


         CALL STR2ET ( TIME, ET )

         Get the ID codes associated with the targets

         CALL BODC2N ( TARG(1), T(1), FOUND )
         CALL BODC2N ( TARG(2), T(2), FOUND )

C
C        Get the apparent states of the target objects as seen from
C        the observer. Also get the apparent radius of each object
C        from the kernel pool. (Use zero radius for any spacecraft;
C        use average radius for anything else.)
C
C           T(i)       is the ID code of the i'th target.
C           S(1-6,i)   is the apparent state of the i'th target.
C           D(i)       is the apparent distance to the i'th target.
C           R(i)       is the apparent radius of the i'th target.
C
C        Function VNORM returns the Euclidean norm (magnitude) of
C        a three-vector.
C
C        Function SUMAD returns the sum of the elements in a
C        double precision array.
C
```

```
         DO I = 1, 2
            CALL SPKEZR ( TARG(I), ET, 'J2000', 'LT+S', OBS,
     .                    S(1,I),  LT )
            D(I) = VNORM( S(1,I) )


            IF ( T(I) .LT. 0 ) THEN
               R(I) = 0.D0

            ELSE
               CALL BODVAR ( T(I), 'RADII', DIM, RADII )
               AVG  = SUMAD ( RADII, 3 ) / 3.D0
               R(I) = ASIN  ( AVG / D(I) )
            END IF
         END DO

C
C        Determine the separation between the two bodies. If the
C        separation between the centers is greater than the sum of
C        the apparent radii, then the target bodies are clear of
C        each other.
C
C        Function VSEP returns the angle of separation between
C        two three-vectors.
C
         SEP = VSEP ( S(1,1), S(1,2) ) - ( R(1) + R(2) )

         IF ( SEP .GT. 0 ) THEN

            WRITE (*,*)
            WRITE (*,*) 'Clear.'

C
C        Otherwise, the smaller body is either occulted or
C        in transit.  We compare ranges to decide which.
C
         ELSE IF ( R(1) .LT. R(2) ) THEN
            IF ( D(1) .LT. D(2) ) THEN
               WRITE (*,*)
               WRITE (*,*) TARG(1), ' in transit across ', TARG(2)
            ELSE
               WRITE (*,*)
               WRITE (*,*) TARG(1), ' occulted by ', TARG(2)
            END IF

         ELSE
            IF ( D(1) .LT. D(2) ) THEN
               WRITE (*,*)
               WRITE (*,*) TARG(2), ' occulted by ', TARG(1)
            ELSE
               WRITE (*,*)
               WRITE (*,*) TARG(2), ' in transit across ', TARG(1)
            END IF
         END IF

C
C        Get the next set of inputs.
C
      END DO
```

```
      END
```

Additional, working examples of using the principal SPK subroutines may be found in the "Cookbook" programs distributed with the NAIF Toolkit.

Supported Data Types

The following representations, or data types, are currently supported by the SPK routines in SPICELIB.

1. Modified Difference Arrays.

   Created by the JPL Orbit Determination Program (ODP), these are used primarily for spacecraft ephemerides.

2. Chebyshev polynomials (position only).

   These are sets of coefficients for the x, y, and z components of the body position. The velocity of the body is obtained by differentiation. This data type is normally used for planet barycenters, and for satellites whose orbits are integrated.

3. Chebyshev polynomials (position and velocity).

   These are sets of coefficients for the x, y, and z components of the body position, and for the corresponding components of the velocity. This data type is normally used for satellites whose orbits are computed directly from theories.

4. Reserved for future use (TRW elements for TDRS and Spacetelescope).

5. Discrete states (two body propagation).

   This data type contains discrete state vectors. A state is obtained for a specified epoch by propagating the state vectors to that epoch according to the laws of two body motion and then taking a weighted average of the resulting states. Normally, this data type is used for comets and asteroids, whose ephemerides are integrated from an initial state or set of osculating elements.

6. Reserved for future use (Analytic Model for Phobos and Deimos).

7. Reserved for future use (Precessing Classical Elements—used by STScI).

8. Equally spaced discrete states (Lagrange interpolation)

   This data type contains discrete state vectors whose time tags are separated by a constant step size. A state is obtained for a specified epoch by finding a set of states 'centered' at that epoch and using Lagrange interpolation on each component of the states.

9. Unequally spaced discrete states (Lagrange interpolation)

   This data type contains discrete state vectors whose time tags may be unequally spaced. A state is obtained for a specified epoch by finding a set of states 'centered' at that epoch and using Lagrange interpolation on each component of the states.

10. Space Command Two-line Elements (Short Period Orbits)

    This data type contains Space Command two-line element representations for objects in Earth orbit (formally called NORAD two-line elements).

11. Reserved for future use.

12. Reserved for future use (Hermite Interpolation Uniform Spacing).

13. Reserved for future use (Hermite Interpolation Non-uniform Spacing).

14. Chebyshev polynomials non-uniform spacing (position and velocity).

    This data type contains Chebyshev polynomial coefficients for the the position and velocity of an object. Unlike SPK Types 2 and 3, the time intervals to which polynomial coefficient sets apply do not have uniform duration.

15. Precessing conic propagation.

    This data type allows for first order precession of the line of apsides and regression of the line of nodes due to the effects of the J2 coefficient in the harmonic expansion of the gravitational potential of an oblate spheroid.

16. Reserved for future use (Elements for European Space Agency's ISO spacecraft).

17. Equinoctial Elements

    This data type represents the motion of an object about another using equinoctial elements. It provides for precession of the line of apsides and regression of the line of nodes. Unlike Type 15, the mean motion, regression of the nodes and precession of the line of apsides are not derived from the gravitational properties of the central body, but are empirical values.

Because SPK files are Double Precision Array Files (DAFs), each segment is stored as an array. Each array corresponding to a particular data type has a particular internal structure. These structures (for the non-reserved types) are described below.

Type 1: Modified Difference Arrays

The first SPK data type contains Modified Difference Arrays (MDA), sometimes called 'difference lines'. This data type is normally used for spacecraft whose ephemerides are produced by JPL's principal trajectory integrator—DPTRAJ. Difference lines are extracted from the spacecraft trajectory file ('P-files' and 'PV-files') created by DPTRAJ.

Each segment containing Modified Difference Arrays contains an arbitrary number of logical records. Each record contains difference line coefficients valid up to some final epoch, along with the state at that epoch. The contents of the records themselves are described in [163]. The subroutine SPKE01 contains the algorithm used to construct a state from a particular record and epoch.

The records within a segment are ordered by increasing final epoch. A segment of this type is structured as follows:

```
+-----------------------------------------+
| Record 1 (difference line coefficients) |
+-----------------------------------------+
| Record 2 (difference line coefficients) |
+-----------------------------------------+
  .
  .
  .
+-----------------------------------------+
| Record N (difference line coefficients) |
+-----------------------------------------+
| Epoch 1                      |
+------------------------------+
| Epoch 2                      |
+------------------------------+
  .
  .
  .
+------------------------------+
| Epoch N                      |
+------------------------------+
| Directory epoch 1            |   (First directory epoch)
+------------------------------+
| Directory epoch 2            |
+------------------------------+
  .
  .
  .
+------------------------------+
| Directory epoch (N/100)*100  |   (Final directory epoch)
+------------------------------+
| N                            |
+------------------------------+
```

The number of records in a segment, N, can be arbitrarily large.

Records 1 through N contain the difference line coefficients and other constants needed to compute state data. Each one of these records contains 71 double precision numbers.

The list of final epochs for the records is stored immediately after the last record.

Following the list of epochs is a second list, the 'directory', containing every 100th epoch from the previous list. If there are N epochs, there will be N/100 directory epochs. If there are fewer than 100 epochs, then the segment will not contain any directory epochs. Directory epochs are used to speed up access to desired records.

The final element in the segment is the number of records contained in the segment, N.

The index of the record corresponding to a particular epoch is the index of the first epoch not less than the target epoch.

Type 2: Chebyshev (position only)

The second SPK data type contains Chebyshev polynomial coefficients for the position of the body as a function of time. Normally, this data type is used for planet barycenters, and for satellites whose ephemerides are integrated. (The velocity of the body is obtained by differentiating the position.)

Each segment contains an arbitrary number of logical records. Each record contains a set of Chebyshev coefficients valid throughout an interval of fixed length. The subroutine SPKE02 contains the algorithm used to construct a state from a particular record and epoch.

The records within a segment are ordered by increasing initial epoch. All records contain the same number of coefficients. A segment of this type is structured as follows:

```
+---------------+
| Record 1      |
+---------------+
| Record 2      |
+---------------+
   .
   .
   .
+---------------+
| Record N      |
+---------------+
| INIT          |
+---------------+
| INTLEN        |
+---------------+
```

```
| RSIZE          |
+----------------+
| N              |
+----------------+
```

A four-number 'directory' at the end of the segment contains the information needed to determine the location of the record corresponding to a particular epoch.

1. INIT is the initial epoch of the first record, given in ephemeris seconds past J2000.
2. INTLEN is the length of the interval covered by each record, in seconds.
3. RSIZE is the total size of (number of array elements in) each record.
4. N is the number of records contained in the segment.

Each record is structured as follows:

```
+------------------+
| MID              |
+------------------+
| RADIUS           |
+------------------+
| X  coefficients  |
+------------------+
| Y  coefficients  |
+------------------+
| Z  coefficients  |
+------------------+
```

The first two elements in the record, MID and RADIUS, are the midpoint and radius of the time interval covered by coefficients in the record. These are used as parameters to perform transformations between the domain of the record (from MID - RADIUS to MID + RADIUS) and the domain of Chebyshev polynomials (from -1 to 1).

The same number of coefficients is always used for each component, and all records are the same size (RSIZE), so the degree of each polynomial is

```
( RSIZE - 2 ) / 3 - 1
```

To facilitate the creation of Type 2 segments, a segment writing routine called SPKW02 has been provided. This routine takes as input arguments the handle of an SPK file that is open for writing, the information needed to construct the segment descriptor, and the data to be stored in the segment. The header of the subroutine provides a complete description of the input arguments and an example of its usage.

Type 3: Chebyshev (position and velocity)

The third SPK data type contains Chebyshev polynomial coefficients for the position and velocity of the body as a function of time. Normally, this data type is used for satellites for which the ephemerides are computed from analytical theories.

The structure of the segment is nearly identical to that of the SPK data Type 2 (Chebyshev polynomials for position only). The only difference is that each logical record contains six sets of coefficients instead of three. The subroutine SPKE03 contains the algorithm used to construct a state from a particular record and epoch.

Each record is structured as follows:

```
+------------------+
```

```
| MID              |
+------------------+
| RADIUS           |
+------------------+
| X  coefficients  |
+------------------+
| Y  coefficients  |
+------------------+
| Z  coefficients  |
+------------------+
| X' coefficients  |
+------------------+
| Y' coefficients  |
+------------------+
| Z' coefficients  |
+------------------+
```

The same number of coefficients is always used for each component, and all records are the same size (RSIZE), so the degree of each polynomial is

```
( RSIZE - 2 ) / 6 - 1
```

To facilitate the creation of Type 3 segments, a segment writing routine called SPKW03 has been provided. This routine takes as input arguments the handle of an SPK file that is open for writing, the information needed to construct the segment descriptor, and the data to be stored in the segment. The header of the subroutine provides a complete description of the input arguments and an example of its usage.

Type 5: Discrete states (two body propagation).

The fifth standard SPK data type contains discrete state vectors. A state is obtained from a Type 5 segment for any epoch that is within the bounds of that segment by propagating the discrete states to the specified epoch according to the laws of two body motion. Normally, this data type is used for comets and asteroids, whose ephemerides are integrated from an initial state or set of osculating elements.

Each segment contains of a number of logical records. Each record consists of an epoch (ephemeris seconds past J2000) and the geometric state of the body at that epoch (x, y, z, dx/dt, dy/dt, dz/dt, in kilometers and kilometers per second). Records are ordered with respect to increasing time.

The records that correspond to an epoch for which a state is desired are the ones whose associated epochs bracket that epoch. The state in each record is used as the initial state in a two-body propagation; a weighted average of the propagated states gives the position of the body at the specified epoch.

The velocity is given by the derivative of the position. Thus the position and velocity at the specified epoch are given by:

```
P  = W(t) * P1(t) + (1-W(t)) * P2(t)

V  = W(t) * V1(t) + (1-W(t)) * V2(t) + W'(t) * ( P1(t) - P2(t) )
```

where P1, V1, P2, and V2 are the position and velocity components of the propagated states and W is the weighting function.

The weighting function used is:

```
W(t) = 0.5 + 0.5 * cos [ PI * ( t - t1 ) / ( t2 - t1 ) ]
```

where t1 and t2 are the epochs that bracket the specified epoch t.

Physically, the epochs and states are stored separately, so that the epochs can be searched as an ordered array. Thus, the initial part of each segment looks like this:

```
+--------------------+
| State 1            |
+--------------------+
          .
          .
          .
+--------------------+
| State N            |
+--------------------+
| Epoch 1            |
+--------------------+
          .
          .
          .
+--------------------+
| Epoch N            |
+--------------------+
```

The number of records in a segment can be arbitrarily large. In order to avoid the file reads required to search through a large array of epochs, each segment contains a simple directory immediately after the final epoch.

This directory contains every 100th epoch in the epoch array. If there are N epochs, there will be N/100 directory epochs. (If there are fewer than 100 epochs, no directory epochs are stored.)

The final items in the segment are GM, the gravitational parameter of the central body (kilometers and seconds), and N, the number of states in the segment. Thus, the complete segment looks like this:

```
+--------------------+
| State 1            |
+--------------------+
          .
          .
          .
+--------------------+
| Epoch 1            |
+--------------------+
          .
          .
          .
+--------------------+
| Epoch N            |
+--------------------+
| Epoch 100          |           (First directory epoch)
+--------------------+
| Epoch 200          |
+--------------------+
          .
          .
          .
+--------------------+
| Epoch (N/100)*100  |           (Final directory epoch)
```

```
+--------------------+
| GM                 |
+--------------------+
| N                  |
+--------------------+
```

To facilitate the creation of Type 5 segments, a segment writing routine called SPKW05 has been provided. This routine takes as input arguments the handle of an SPK file that is open for writing, the information needed to construct the segment descriptor, and the data to be stored in the segment. The header of the subroutine provides a complete description of the input arguments and an example of its usage.

Type 8: Lagrange Interpolation (Equally Spaced)

The eighth SPK data type represents a continuous ephemeris using a discrete set of states and a Lagrange interpolation method. The epochs (also called 'time tags') associated with the states must be evenly spaced: there must be some positive constant STEP such that each time tag differs from its predecessor and successor by STEP seconds. For a request epoch not corresponding to the time tag of some state, the data type defines a state by interpolating each component of a set of states whose epochs are 'centered' near the request epoch. Details of how these states are selected and interpolated are given below.

The SPK system can also represent an ephemeris using unequally spaced discrete states and Lagrange interpolation; SPK Type 9 does this. SPK Type 9 sacrifices some run-time speed and economy of storage in order to achieve greater flexibility.

The states in a Type 8 segment are geometric: they do not take into account aberration corrections. The six components of each state vector represent the position and velocity (x, y, z, dx/dt, dy/dt, dz/dt, in kilometers and kilometers per second) of the body to which the ephemeris applies, relative to the center specified by the segment's descriptor. The epochs corresponding to the states are barycentric dynamical times (TDB), expressed as seconds past J2000.

Each segment also has a polynomial degree associated with it; this is the degree of the interpolating polynomials to be used in evaluating states based on the data in the segment. The identical degree is used for interpolation of each state component.

Type 8 SPK segments have the structure shown below:

```
                                      +--------+
                                      |  x(1)  |
                                  /   +--------+
                                 /    |  y(1)  |
                                /     +--------+
                               /      |  z(1)  |
+----------------------+      /       +--------+
| State 1              |   <          |dx(1)/dt|
+----------------------+    \         +--------+
| State 2              |     \        |dy(1)/dt|
+----------------------+      \       +--------+
            .                  \      |dz(1)/dt|
            .                         +--------+
            .
+----------------------+
| State N              |
+----------------------+
| Epoch of state 1 (TDB)|
+----------------------+
| Step size            |
+----------------------+
| Polynomial degree    |
+----------------------+
| Number of states     |
+----------------------+
```

In the diagram, each box representing a state vector corresponds to six double precision numbers; the other boxes represent individual double precision numbers. Since the epochs of the states are evenly spaced, they are represented by a start epoch and a step size. The number of states must be greater than the interpolating polynomial degree.

The Type 8 interpolation method works as follows: given an epoch at which a state is requested and a segment having coverage for that epoch, the Type 8 reader finds a group of states whose epochs are 'centered' about the epoch. The size of the group is one greater than the polynomial degree associated with the segment. If the group size N is even, then the group will consist of N consecutive states such that the request time is between the epochs of the members of the group having indices, relative to the start of the group, of N/2 and (N/2 + 1), inclusive. When N is odd, the group will contain a central state whose epoch is closest to the request time, and will also contain (N-1)/2 neighboring states on either side of the central one. The Type 8 evaluator will then use Lagrange interpolation on each component of the states to produce a state corresponding to the request time. For the jth state component, the interpolation algorithm is mathematically equivalent to finding the unique polynomial of degree N-1 that interpolates the ordered pairs

```
( epoch(i), state(j,i) ),  i = k ,  k , ... , k
                                1    2         N
```

and evaluating the polynomial at the requested epoch. Here

```
k ,  k , ... , k
 1    2         N
```

are the indices of the states in the interpolation group,

```
epoch(i)
```

is the epoch of the ith state and

```
state(j,i)
```

is the jth component of the ith state.

There is an exception to the state selection algorithm described above: the request time may be too near the first or last state of the segment to be properly bracketed. In this case, the set of states selected for interpolation still has size N, and includes either the first or last state of the segment.

To facilitate the creation of Type 8 segments, a segment writing routine called SPKW08 has been provided. This routine takes as input arguments the handle of an SPK file that is open for writing, the information needed to construct the segment descriptor, and the data to be stored in the segment. The header of the subroutine provides a complete description of the input arguments and an example of its usage.

Type 9: Lagrange Interpolation (Unequally Spaced)

The ninth SPK data type represents a continuous ephemeris using a discrete set of states and a Lagrange interpolation method. The epochs (also called 'time tags') associated with the states need not be evenly spaced. For a request epoch not corresponding to the time tag of some state, the data type defines a state by interpolating each component of a set of states whose epochs are 'centered' near the request epoch. Details of how these states are selected and interpolated are given below.

The states in a Type 9 segment are geometric: they do not take into account aberration corrections. The six components of each state vector represent the position and velocity (x, y, z, dx/dt, dy/dt, dz/dt, in kilometers and kilometers per second) of the body to which the ephemeris applies, relative to the center specified by the segment's descriptor. The epochs corresponding to the states are barycentric dynamical times (TDB), expressed as seconds past J2000.

Each segment also has a polynomial degree associated with it; this is the degree of the interpolating polynomials to be used in evaluating states based on the data in the segment. The identical degree is used for interpolation of each state component.

Type 9 SPK segments have the structure shown below:

```
                                          +--------+
                                          |  x(1)  |
                                      /   +--------+
                                     /    |  y(1)  |
                                    /     +--------+
                                   /      |  z(1)  |
+-----------------------+         /       +--------+
| State 1               |  <              |dx(1)/dt|
+-----------------------+   \             +--------+
| State 2               |    \            |dy(1)/dt|
+-----------------------+     \           +--------+
            .                  \          |dz(1)/dt|
            .                             +--------+
            .
+-----------------------+
| State N               |
+-----------------------+
| Epoch 1               |
+-----------------------+
| Epoch 2               |
+-----------------------+
            .
            .
            .
+-----------------------+
| Epoch N               |
+-----------------------+
| Epoch 100             | (First directory)
+-----------------------+
            .
            .
            .
+-----------------------+
| Epoch ((N-1)/100)*100 | (Last directory)
+-----------------------+
| Polynomial degree     |
+-----------------------+
| Number of states      |
+-----------------------+
```

In the diagram, each box representing a state vector corresponds to six double precision numbers; the other boxes represent individual double precision numbers. The number of states must be greater than the interpolating polynomial degree.

The set of time tags is augmented by a series of directory entries; these entries allow the Type 9 reader to search for states more efficiently. The directory entries contain time tags whose indices are multiples of 100. The set of indices of time tags stored in the directories ranges from 100 to

```
(  (N-1) / 100  ) * 100
```

where N is the total number of time tags. Note that if N is

```
Q * 100
```

then only

```
Q - 1
```

directory entries are stored, and in particular, if there are only 100 states in the segment, there are no directories.

The Type 9 interpolation algorithm is virtually identical to the Type 8 algorithm; see the discussion of SPK Type 8 for details. However, the Type 9 algorithm executes more slowly than the Type 8 algorithm, since the Type 9 reader must search through tables of time tags to find appropriates states to interpolate, while the Type 8 reader can locate the correct set of states to interpolate by a direct computation.

To facilitate the creation of Type 9 segments, a segment writing routine called SPKW09 has been provided. This routine takes as input arguments the handle of an SPK file that is open for writing, the information needed to construct the segment descriptor, and the data to be stored in the segment. The header of the subroutine provides a complete description of the input arguments and an example of its usage.

Type 10: Space Command Two-Line Elements

The SPK data Type 10 uses the SPICE concept of a generic segment to store a collection of packets each of which models the trajectory of some Earth satellite using Space Command two-line elements (formerly the North American Air Defense — NORAD).

The storage, arrangement and retrieval of two-line element sets is handled by the SPICE generic segment software described in the document GENSEG.REQ. (The document GENSEG.REQ is currently in preparation.) We review only the pertinent points about generic segments here.

A generic SPK segment contains several logical data partitions:

1. A partition for constant values to be associated with each data packet in the segment.
2. A partition for the data packets.

3. A partition for epochs.
4. A partition for a packet directory, if the segment contains variable sized packets.
5. A partition for an epoch directory.
6. A reserved partition that is not currently used. This partition is only for the use of the NAIF group at the Jet Propulsion Laboratory (JPL).
7. A partition for the meta data which describes the locations and sizes of other partitions as well as providing some additional descriptive information about the generic segment.

```
+============================+
|         Constants          |
+============================+
|          Packet 1          |
|----------------------------|
|          Packet 2          |
|----------------------------|
|              .             |
|              .             |
|              .             |
|----------------------------|
|          Packet N          |
+============================+
|      Reference Epochs      |
+============================+
|      Packet Directory      |
+============================+
|       Epoch Directory      |
+============================+
|       Reserved  Area       |
+============================+
|     Segment Meta Data      |
+----------------------------+
```

Only the placement of the meta data at the end of a generic segment is required. The other data partitions may occur in any order in the generic segment because the meta data will contain pointers to their appropriate locations within the generic segment.

Each "packet" of a Type 10 segment contains one set of two-line elements, the nutations in longitude and obliquity of the Earth's pole, and the rates of these nutations. Each packet is arranged as shown below. (The notation below is taken from the description that accompanies the code available from Space Command for the evaluation of two-line elements.)

```
  +-------------------+
1 |      NDT20        |
  +-------------------+
2 |      NDD60        |
```

```
   +--------------------+
 3 |      BSTAR         |
   +--------------------+
 4 |      INCL          |
   +--------------------+
 5 |      NODEO         |     Two-line element packet
   +--------------------+
 6 |      ECC           |
   +--------------------+
 7 |      OMEGA         |
   +--------------------+
 8 |      MO            |
   +--------------------+
 9 |      NO            |
   +--------------------+
10 |      EPOCH         |
   +--------------------+
11 |      NU.OBLIQUITY  |
   +--------------------+
12 |      NU.LONGITUDE  |
   +--------------------+
13 |     dOBLIQUITY/dt  |
   +--------------------+
14 |     dLONGITUDE/dt  |
   +--------------------+
```

The constants partition of the Type 10 segment contains the following eight constants.

```
   +-------------------------------------------+
1  |  J2 gravitational harmonic for Earth      |
   +-------------------------------------------+
2  |  J3 gravitational harmonic for Earth      |
   +-------------------------------------------+
3  |  J4 gravitational harmonic for Earth      |
   +-------------------------------------------+
   |  Square root of the GM for Earth where GM |
4  |  is expressed in Earth radii cubed per    |
   |  minutes squared                          |
   +-------------------------------------------+
5  |  Equatorial radius of the Earth in km     |
   +-------------------------------------------+
6  |  Low altitude bound for atmospheric       |
   |  model in km                              |
   +-------------------------------------------+
7  |  High altitude bound for atmospheric      |
   |  model in km                              |
   +-------------------------------------------+
8  |  Distance units/Earth radius (normally 1) |
   +-------------------------------------------+
```

The reference epochs partition contains an ordered collection of epochs. The i'th reference epoch is equal to the epoch in the i'th packet.

The "epoch directory" contains every 100th reference epoch. The epoch directory is used to efficiently locate an the reference epoch that should be associated with a two line element packet.

The "packet directory" is empty.

As noted above the exact location of the various partitions must be obtained from the Meta data contained at the end of the segment. Access to the data should be made via the SPICELIB generic segment routines or via the SPK Type 10 reader—SPKR10. The routine SPKW10 is available for writing a Type 10 generic segment.

Type 14: Chebyshev Polynomials — Unequal Time Steps

The SPK data Type 14 uses the SPICE concept of a generic segment to store a collection of packets each of which models the trajectory of some object with respect to another over some interval of time. Each packet contains a set of coefficients for Chebyshev polynomials that approximate the position and velocity of some object. The time intervals corresponding to each packet are non-overlapping. Moreover their union covers the interval of time spanned by the start and end times of the Type 14 segment. Unlike Types 2 and 3 the time spacing between sets of coefficients for a Type 14 segment may be non-uniform.

The storage, arrangement and retrieval of packets is handled by the SPICE generic segment software. That software is documented in the document GENSEG.REQ. (The document GENSEG.REQ is currently in preparation.) We only review the pertinent points about generic segments here.

A generic SPK segment contains several logical data partitions:

1. A partition for constant values to be associated with each data packet in the segment.
2. A partition for the data packets.
3. A partition for epochs.
4. A partition for a packet directory, if the segment contains variable sized packets.
5. A partition for an epoch directory.
6. A reserved partition that is not currently used. This partition is only for the use of the NAIF group at the Jet Propulsion Laboratory (JPL).

7. A partition for the meta data which describes the locations and sizes of other partitions as well as providing some additional descriptive information about the generic segment.

```
+============================+
|         Constants          |
+============================+
|          Packet 1          |
|----------------------------|
|          Packet 2          |
|----------------------------|
|             .              |
|             .              |
|             .              |
|----------------------------|
|          Packet N          |
+============================+
|      Reference Epochs      |
+============================+
|      Packet Directory      |
+============================+
|       Epoch Directory      |
+============================+
|       Reserved  Area       |
+============================+
|     Segment Meta Data      |
+----------------------------+
```

Only the placement of the meta data at the end of a generic segment is required. The other data partitions may occur in any order in the generic segment because the meta data will contain pointers to their appropriate locations within the generic segment.

In the case of Type 14 SPK segments each "packet" contains an epoch, EPOCH, an allowed time offset, OFFSET, from the epoch, and 6 sets of Chebyshev polynomial coefficients which are used to evaluate the x,y,z, dx/dt, dy/dt, and dz/dt components of the state for epochs within OFFSET seconds of the EPOCH. Each packet is organized with the following structure:

```
------------------------------------------------
|  The midpoint of the approximation interval  |
------------------------------------------------
|  The radius of the approximation interval    |
------------------------------------------------
|  CHBDEG+1 coefficients for the X coordinate  |
------------------------------------------------
|  CHBDEG+1 coefficients for the Y coordinate  |
------------------------------------------------
|  CHBDEG+1 coefficients for the Z coordinate  |
------------------------------------------------
|  CHBDEG+1 coefficients for the X velocity    |
------------------------------------------------
|  CHBDEG+1 coefficients for the Y velocity    |
------------------------------------------------
```

```
|  CHBDEG+1 coefficients for the Z velocity    |
------------------------------------------------
```

The maximum degree Chebyshev representation that can currently be accommodated is 18. Packets are stored in increasing order of the midpoint of the approximation interval.

The "constants" partition contains a single value, the degree of the Chebyshev representation.

The reference epochs partition contains an ordered collection of epochs. The i'th reference epoch corresponds to the beginning of the interval for which the i'th packet can be used to determine the state of the object modelled by this segment.

The "epoch directory" contains every 100th reference epoch. The epoch directory is used to efficiently locate an the reference epoch that should be associated with an epoch for which a state has been requested.

The "packet directory" is empty.

As noted above the exact location of the various partitions must be obtained from the Meta data contained at the end of the segment.

Access to the data should be made via the SPICELIB generic segment routines.

Type 14 segments should be created using the routines SPK14B, SPK14A, and SPK14E. The usage of these routines is discussed in SPK14B.

Type 15: Precessing Conic Propagation.

The SPK data Type 15 represents a continuous ephemeris using a compact analytic model. The object is modelled as orbiting a central body under the influence of a central mass plus first order secular effects of the J2 term in harmonic expansion of the the central body gravitational potential.

Type 15 SPK segments have the structure shown below:

```
+--------------------------------+
| Epoch of Periapsis             |
+--------------------------------+
| Trajectory pole_x              |
+--------------------------------+
| Trajectory pole_y              |
+--------------------------------+
| Trajectory pole_z              |
+--------------------------------+
| Periapsis Unit Vector_x        |
+--------------------------------+
```

```
| Periapsis Unit Vector_y        |
+--------------------------------+
| Periapsis Unit Vector_z        |
+--------------------------------+
| Semi-Latus Rectum              |
+--------------------------------+
| Eccentricity                   |
+--------------------------------+
| J2 Processing Flag             |
+--------------------------------+
| Central Body Pole_x            |
+--------------------------------+
| Central Body Pole_y            |
+--------------------------------+
| Central Body Pole_z            |
+--------------------------------+
| Central Body GM                |
+--------------------------------+
| Central Body J2                |
+--------------------------------+
| Central Body Equatorial Radius |
+--------------------------------+
```

It is important to note that the epoch must be that of periapsis passage. Precession of the line of apsides and regression of the line of nodes is computed relative to this epoch.

The effects of the J2 term are not applied if the eccentricity is greater than or equal to 1.

To facilitate the creation of Type 15 segments, a segment writing routine called SPKW15 has been provided. This routine takes as input arguments the handle of an SPK file that is open for writing, the information needed to construct the segment descriptor, and the data to be stored in the segment. The header of the subroutine provides a complete description of the input arguments and an example of its usage.

Type 17: Equinoctial Elements.

The SPK data Type 17 represents a continuous ephemeris using a compact analytic model. The object is following an elliptic orbit with precessing line of nodes and argument of periapse relative to the equatorial frame of some central body. The orbit is modelled via equinoctial elements.

Type 17 SPK segments have the structure shown below:

```
  +----------------------------------+
1 | Epoch of Periapsis               |
  +----------------------------------+
2 | Semi-Major Axis                  |
```

```
    +-----------------------------------+
 3  | H term of equinoctial elements    |
    +-----------------------------------+
 4  | K term of equinoctial elements    |
    +-----------------------------------+
 5  | Mean longitude at epoch           |
    +-----------------------------------+
 6  | P term of equinoctial elements    |
    +-----------------------------------+
 7  | Q term of equinoctial elements    |
    +-----------------------------------+
 8  | rate of longitude of periapse     |
    +-----------------------------------+
 9  | mean longitude rate               |
    +-----------------------------------+
10  | longitude of ascending node rate  |
    +-----------------------------------+
11  | equatorial pole right ascension   |
    +-----------------------------------+
12  | equatorial pole declination       |
    +-----------------------------------+
```

To facilitate the creation of Type 17 segments, a segment writing routine called SPKW17 has been provided. This routine takes as input arguments the handle of an SPK file that is open for writing, the information needed to construct the segment descriptor, and the data to be stored in the segment. The header of the subroutine provides a complete description of the input arguments and an example of its usage.

Appendix A — Summary of SP-kernel Routines

Summary of Mnemonics

SPICELIB contains a family of subroutines that are designed specifically for use with SPK files. The name of each routine begins with the letters 'SPK', followed by a two- or three-character mnemonic. For example, the routine that returns the state of one body with respect to another is named SPKEZ, pronounced 'S-P-K-E-Z'.

Many of the routines listed are entry points of another routine. If a routine is an entry point, the parent routine's name will be listed inside brackets preceding the mnemonic translation.

The following is a complete list of mnemonics and translations, in alphabetical order.

```
SPK14A             ( S/P-kernel, add to a Type 14 segment )
SPK14B             ( S/P-kernel, begin  a Type 14 segment )
SPK14E             ( S/P-kernel, end    a Type 14 segment )

SPKAPO             ( S/P-Kernel, "apparent" position only )
SPKAPP             ( S/P-kernel, Apparent state           )

SPKCLS             ( S/P-kernel, close after write        )

SPKE01             ( S/P-kernel, Evaluate record, Type 01 )
SPKE02             ( S/P-kernel, Evaluate record, Type 02 )
SPKE03             ( S/P-kernel, Evaluate record, Type 03 )
SPKE05             ( S/P-kernel, Evaluate record, Type 05 )
SPKE08             ( S/P-kernel, Evaluate record, Type 08 )
SPKE09             ( S/P-kernel, Evaluate record, Type 09 )
SPKE10             ( S/P-kernel, Evaluate record, Type 10 )
SPKE14             ( S/P-kernel, Evaluate record, Type 14 )
SPKE15             ( S/P-kernel, Evaluate record, Type 15 )
SPKE17             ( S/P-kernel, Evaluate record, Type 17 )

SPKEZ              ( S/P-kernel, Easy state                 )
SPKEZP             ( S/P Kernel, easy position              )
SPKEZR             ( S/P-kernel, Easier state               )
SPKGEO             ( S/P-kernel, Geometric state            )
SPKGPS             ( S/P Kernel, geometric position         )
SPKLEF    [SPKBSR] ( S/P-kernel, Load ephemeris file        )
```

```
SPKOPA            ( S/P-kernel, open for addition         )
SPKOPN            ( S/P-kernel, open new file             )
SPKPDS            ( S/P-kernel, pack descriptor           )
SPKPOS            ( S/P Kernel, position                  )
SPKPV             ( S/P-kernel, Position, velocity        )
SPKPVN            ( S/P-kernel, Position, velocity---native)

SPKR01            ( S/P-kernel, Read record, Type 01     )
SPKR02            ( S/P-kernel, Read record, Type 02     )
SPKR03            ( S/P-kernel, Read record, Type 03     )
SPKR05            ( S/P-kernel, Read record, Type 05     )
SPKR08            ( S/P-kernel, Read record, Type 08     )
SPKR09            ( S/P-kernel, Read record, Type 09     )
SPKR10            ( S/P-kernel, Read record, Type 10     )
SPKR14            ( S/P-kernel, Read record, Type 14     )
SPKR15            ( S/P-kernel, Read record, Type 15     )
SPKR17            ( S/P-kernel, Read record, Type 17     )

SPKS01            ( S/P-kernel, Subset data, Type 01     )
SPKS02            ( S/P-kernel, Subset data, Type 02     )
SPKS03            ( S/P-kernel, Subset data, Type 03     )
SPKS05            ( S/P-kernel, Subset data, Type 05     )
SPKS08            ( S/P-kernel, Subset data, Type 08     )
SPKS09            ( S/P-kernel, Subset data, Type 09     )
SPKS10            ( S/P-kernel, Subset data, Type 10     )
SPKS14            ( S/P-kernel, Subset data, Type 14     )
SPKS15            ( S/P-kernel, Subset data, Type 15     )
SPKS17            ( S/P-kernel, Subset data, Type 17     )

SPKSFS    [SPKBSR] ( S/P-kernel, file and segment        )
SPKSSB            ( S/P-kernel, Solar system barycenter  )
SPKUDS            ( S/P-kernel, Unpack descriptor        )
SPKUEF    [SPKBSR] ( S/P-kernel, Unload ephemeris file    )
SPKSUB            ( S/P-kernel, Subset a segment         )
SPKW02            ( S/P-kernel, Write segment, Type 02   )
SPKW03            ( S/P-kernel, Write segment, Type 03   )
SPKW05            ( S/P-kernel, Write segment, Type 05   )
SPKW08            ( S/P-kernel, Write segment, Type 08   )
SPKW09            ( S/P-kernel, Write segment, Type 09   )
SPKW10            ( S/P-kernel, Write segment, Type 10   )
SPKW15            ( S/P-kernel, Write segment, Type 15   )
SPKW17            ( S/P-kernel, Write segment, Type 17   )
```

Summary of Calling Sequences

The calling sequences for the SPK subroutines are summarized below. The subroutines are grouped by function.

Loading, unloading files:

```
SPKLEF ( FNAME, HANDLE )
SPKUEF (        HANDLE )
```

Computing states and positions:

```
SPKEZR ( TNAME,  ET, REF,        ABERR, ONAME, STATE, LT )
SPKPOS ( TNAME,  ET, REF,        ABERR, ONAME, POSTN, LT )
SPKEZ  ( TARGET, ET, REF,        ABERR, OBS,   STATE, LT )
SPKEZP ( TARGET, ET, REF,        ABERR, OBS,   POSTN, LT )
SPKAPP ( TARGET, ET, REF, STOBS, ABERR,        STATE, LT )
SPKAPO ( TARGET, ET, REF, STOBS, ABERR,        POSTN, LT )
SPKSSB ( TARGET, ET, REF,                      STATE     )
SPKGEO ( TARGET, ET, REF,               OBS,   STATE, LT )
SPKGPS ( TARGET, ET, REF,               OBS,   POSTN, LT )

SPKPVN ( HANDLE, DESCR, ET, REF, STATE, CENTER )
SPKPV  ( HANDLE, DESCR, ET, REF, STATE, CENTER )
```

Selecting files, segments:

```
SPKSFS ( TARGET, ET, HANDLE, DESCR, PDGREE, FOUND )
```

Reading, evaluating records:

```
SPKR01 ( HANDLE, DESCR, ET, RECORD        )
SPKE01 (                ET, RECORD, STATE )

SPKR02 ( HANDLE, DESCR, ET, RECORD        )
SPKE02 (                ET, RECORD, STATE )

SPKR03 ( HANDLE, DESCR, ET, RECORD        )
SPKE03 (                ET, RECORD, STATE )

SPKR05 ( HANDLE, DESCR, ET, RECORD        )
SPKE05 (                ET, RECORD, STATE )

SPKR08 ( HANDLE, DESCR, ET, RECORD        )
SPKE08 (                ET, RECORD, STATE )

SPKR09 ( HANDLE, DESCR, ET, RECORD        )
SPKE09 (                ET, RECORD, STATE )

SPKR10 ( HANDLE, DESCR, ET, RECORD        )
SPKE10 (                ET, RECORD, STATE )

SPKR14 ( HANDLE, DESCR, ET, RECORD        )
SPKE14 (                ET, RECORD, STATE )

SPKR15 ( HANDLE, DESCR, ET, RECORD        )
SPKE15 (                ET, RECORD, STATE )

SPKR17 ( HANDLE, DESCR, ET, RECORD        )
SPKE17 (                ET, RECORD, STATE )
```

Writing segments to files.

```
SPKPDS ( BODY,   CENTER, FRAME,  TYPE,   FIRST, LAST,  DESCR )
```

```
SPKW02 ( HANDLE, BODY,   CENTER, FRAME,  FIRST, LAST,  SEGID,
         INTLEN, N,      POLYDG, CDATA,  BTIME               )

SPKW03 ( HANDLE, BODY,   CENTER, FRAME,  FIRST, LAST,  SEGID,
         INTLEN, N,      POLYDG, CDATA,  BTIME               )

SPKW05 ( HANDLE, BODY,   CENTER, FRAME,  FIRST, LAST,  SEGID,
         GM,     N,      STATES, EPOCHS                      )

SPKW08 ( HANDLE, BODY,   CENTER, FRAME,  FIRST, LAST,  SEGID,
         DEGREE, N,      STATES, EPOCH1, STEP                )

SPKW09 ( HANDLE, BODY,   CENTER, FRAME,  FIRST, LAST,  SEGID,
         DEGREE, N,      STATES, EPOCHS                      )

SPKW10 ( HANDLE, BODY,   CENTER, FRAME,  FIRST, LAST,
         SEGID,  CONSTS, N,      ELEMS,  EPOCHS      )


SPK14B ( HANDLE, SEGID,  BODY,   CENTER, FRAME,
         FIRST,  LAST,   CHBDEG  )

SPK14A ( HANDLE, NCSETS, COEFFS, EPOCHS )

SPK14E ( HANDLE )


SPKW15 ( HANDLE, BODY,   CENTER, FRAME,  FIRST, LAST,  SEGID,
         EPOCH,  TPOLE,  PERI,   P,      ECC,   J2FLG, CPOLE,
         GM,     J2,     RADIUS                              )

SPKW17 ( HANDLE, BODY,  CENTER, FRAME,  FIRST, LAST,
         SEGID,  EPOCH, EPOCH,  EQEL,   RAPOL, DECPOL )
```

Examining segment descriptors:

```
SPKUDS ( DESCR, BODY, CENTER, FRAME, TYPE,
         FIRST, LAST, BEGIN,  END         )
```

Extracting subsets of data from a segment:

```
SPKS01 ( HANDLE, BADDR, EADDR, BEGIN, END )
SPKS02 ( HANDLE, BADDR, EADDR, BEGIN, END )
SPKS03 ( HANDLE, BADDR, EADDR, BEGIN, END )
SPKS05 ( HANDLE, BADDR, EADDR, BEGIN, END )
SPKS08 ( HANDLE, BADDR, EADDR, BEGIN, END )
SPKS09 ( HANDLE, BADDR, EADDR, BEGIN, END )
SPKS10 ( HANDLE, BADDR, EADDR, BEGIN, END )
SPKS14 ( HANDLE, BADDR, EADDR, BEGIN, END )
SPKS15 ( HANDLE, BADDR, EADDR, BEGIN, END )
SPKS17 ( HANDLE, BADDR, EADDR, BEGIN, END )

SPKSUB ( HANDLE, DESCR, IDENT, BEGIN, END, NEWH )
```

To write new or append segments to SPK files:

```
SPKOPN ( NAME, IFNAME, NCOMCH, HANDLE )
SPKOPA ( FILE, HANDLE                 )
SPKCLS ( HANDLE                       )
```

Appendix B — A Template for SPK Comments

An undocumented ephemeris is in many respects worse than undocumented source code. With source code you can at least read the code and perhaps discern the function of the source code. An ephemeris on the other hand is a binary file. All it contains are numbers. It's very difficult to determine the purpose of an ephemeris simply from the state information it contains. For this reason, any ephemeris created for use by anyone other than yourself needs documentation.

If you create SPK files NAIF strongly recommends that you include descriptive documentation in the comments portion of the SPK file. You can use the utility program COMMNT to insert comments into the file, or you may use the routines in the SPC family to insert the comments when you create the SPK file. (See COMMNT.UG or SPC.REQ for further details.)

This appendix addresses the contents of your comments. What will others (or yourself) want to know about the SPK file weeks, months or years after it has been created? Providing this information can be a challenge. It's difficult to know in advance all the questions someone might ask about an ephemeris you've created. To assist with this task NAIF has devised a "template" that you may wish to use as a starting point when creating the comments for an SPK file.

Constraints

The comments you place in an SPK file must be plain ASCII text. Each line of text must consist of 80 or fewer characters. The text must contain only printing characters (ASCII characters 32 through 126).

The Basic Template

Here's one way to create the comments for an SPK file.

Objects in the Ephemeris

List the names and NAIF ID codes for the objects in the file.

Approximate Time Coverage

Provide a summary of the time for which states are available for the objects in the file. If you use UTC times in this summary and the ephemeris extends more than 6 months into the future, you should probably state that the times are approximate. You don't know when leapseconds will occur more than a few months in advance, so you can't know the exact UTC time boundaries for the ephemeris if it extends years into the future.

Status

Provide the "status" of the ephemeris. Tell the user why this ephemeris was created and for whom it is intended. For example, if this is the second in a series of ephemerides that will be produced for some object tell which ephemeris this one supersedes. Tell the user when the next ephemeris in the series will be available. Is the ephemeris suitable only for preliminary studies? Is it good for all Earth based observations? Is this an official operational product? Are there situations for which the ephemeris is not suitable?

Pedigree

Provide a production summary for the ephemeris. Tell when the ephemeris was produced (the system time stamp may not port if the file is copied to other systems). Say who produced the ephemeris; what source products were used in the production; what version of the producing program was used in the creation of the ephemeris. If the ephemeris is based on a set of recent observations, say so. In short give the user the pedigree of this ephemeris. This information is mostly for your benefit. If a problem arises with the ephemeris, you will know how the problem was created and have a better chance of fixing the problem.

Usage

Provide information the user will need to effectively use the ephemeris. Tell the user what other SPICE kernels are needed to use this ephemeris. For example, if the ephemeris contains only the state of an asteroid relative to the sun, the user will probably need a planetary ephemeris to effectively use the one you've created. Recommend a planetary ephemeris to use with your SPK file. If the ephemeris contains states of objects relative to non-inertial frames, the user will probably need other kernels so that various state transformations can be performed. Recommend which of these kernels the user should use with your SPK file.

Accuracy

If possible give some estimate as to the accuracy of your SPK file. Use numbers. Words such as "this is the best available" do not convey how much you know about the ephemeris.

Special Notes

Provide a description of any special properties of this ephemeris. For example, if some observation seems to be in conflict with this ephemeris you should probably point this out.

References

List any references that may be relevant to the understanding of the ephemeris. For example, if the ephemeris is based upon observations contained in the literature, site the appropriate articles. If there is some technical memorandum or private communication that addresses certain aspects of this ephemeris list it. This will allow you to more easily answer questions about the ephemeris.

Contacts

List your phone number, mail or e-mail address so that users of the ephemeris will be able to get in touch with you to ask questions or offer praise.