C- Kernel Files

October 14, 1999

This October 1999 version of the document differs from the previous version of November 1992 in the following areas:

Code examples showing calls to the routine DAFONW now show calls to CKOPN in its place, and code examples showing calls to the routine SCE2T now show calls to SCE2C instead.

All statements referring to the base frame of a C-matrix or quaternion have been modified so as not to indicate that the base frame is inertial.

The source-code-level discussion of the implementation of the high level CK readers has been removed. The implementation is not part of the C-kernel software interface and is not guaranteed to remain unchanged. The data selection algorithms used by the readers ARE part of the interface, and the descriptions of the algorithms have been retained.

In addition, some minor changes have been made to simplify maintenance of both the Fortran and C versions of this document.

References

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

1. [167] Double Precision Array Files Required Reading. (DAF)

2. [222] Spacecraft Clock Time Required Reading. (SCLK)
3. [214] Rotations Required Reading. (ROTATIONS)
4. [211] SPC Required Reading: Comments in SPK and CK Files. (SPC)

Introduction

In the SPICE system, pointing data for science instruments are stored in the C-kernel, the ``C'' in SPICE. The pointing of an instrument is often expressed in terms of a transformation matrix from some standard base reference frame to a local, instrument-fixed frame. In the past, the instrument was often a camera, and the transformation was thus dubbed the ``C-matrix''; hence the choice of C as the name for the pointing kernel.

The data contained in C-kernel files can be accessed and manipulated by a collection of FORTRAN 77 subroutines which are part of the SPICELIB library. These subroutines can be integrated into user application programs. The purpose of this document is to describe both the C-kernel file structure and the associated SPICELIB software to the level of detail necessary for the user to program almost any application.

With few exceptions, all subroutines and functions appearing in this document are part of SPICELIB. The exceptions are placeholders for user-supplied routines which appear in some of the code examples. Each SPICELIB routine is prefaced with a complete SPICELIB header which describes inputs, outputs, restrictions and context, and provides examples of usage. The authoritative documentation for any subroutine is its header, which should be consulted before using the routine in any program. A summary of the CK subroutines presented in this document is included as Appendix A.

Preliminaries

In this chapter we discuss four concepts that are essential to using the C-kernel: specification of spacecraft and instruments, C-matrices, angular velocity vectors, and spacecraft clock time.

Specifying Spacecraft and Instruments

C-kernel files and software use integer codes to refer to instruments and the spacecraft on which they are mounted. You will use these instrument numbers with C-kernel readers to request pointing data.

In order to avoid confusion, NAIF, in cooperation with the science teams from each flight project, will assign instrument codes using the following scheme.

If you're familiar with SPICE S- and P-kernels, you know that NAIF codes for spacecraft are negative integers: -31 for Voyager 1, -32 for Voyager 2, -94 for Mars Global Surveyor, and so on. We borrow from this convention in defining instrument codes.

For example, the Voyager 2 instruments could have been given these IDs:

-32000	Instrument Scan Platform
-32001	ISSNA (Imaging science narrow angle camera)
-32002	ISSWA (Imaging science wide angle camera)
-32003	PPS (Photopolarimeter)
-32004	UVSAG (Ultraviolet Spectrometer, Airglow port)
-32005	UVSOCC (Ultraviolet Spectrometer, Occultation port)
-32006	IRIS (Infrared Interferometer Spectrometer and Radiometer)

The simple coding formula is

```
   SPICE s/c instrument code = (s/c code)*(1000) - instrument number
```
which allows for 999 instruments on board any one spacecraft.

The term ``instrument'' is used loosely throughout this document since the concept of orientation is applicable to structures other than just science instruments. For example, some of the Galileo instruments are in a fixed position relative to the scan platform. It might therefore be prudent to have a single file containing the orientation of the scan platform, and then produce the pointing for each of the scan platform science instruments by applying instrument offset angles obtained from the I-kernel.

C-Matrices

A C-matrix is a 3x3 matrix that transforms Cartesian coordinates referenced to a ``base frame'' to coordinates in an instrument-fixed reference frame. In earlier versions of SPICELIB, the base frame was required to be inertial; this restriction has been removed.

The C-matrix transforms coordinates as follows: if a vector v has coordinates (x, y, z) in some base reference frame (like J2000), then v has coordinates (x', y', z') in instrument-fixed coordinates, where

```
   [          ] [ x ]     [ x']
   | C-matrix | | y |  =  | y'|
   [          ] [ z ]     [ z']
```

The transpose of a C-matrix rotates vectors from the instrument-fixed frame to the base frame:

```
   [          ]T [ x']     [ x ]
   | C-matrix |  | y'|  =  | y |
   [          ]  [ z']     [ z ]
```

Therefore, if the coordinates of an instrument in the instrument fixed frame are known, then the transpose of the C-matrix can be used to determine the corresponding coordinates in a base reference frame. This information can be used to help answer questions such as, ``What is the latitude and longitude of the point on the planet that the camera was pointing at when it shuttered this picture?''

The high-level CK file reader CKGP (Get Pointing) returns a C-matrix that specifies the pointing of a spacecraft structure at a particular time. An example program is included in

Appendix B, which solves the longitude and latitude problem presented above using CKGP and other SPICELIB subroutines.

Angular Velocity Vectors

In the C-kernel an angular velocity vector is a vector with respect to a base frame whose direction gives the right-handed axis about which an instrument-fixed reference frame is rotating, and whose magnitude is equal to the magnitude of the rotation velocity, in radians per second.

Angular rate information may be important for certain types of science analysis. For instance, investigators for imaging instruments might use angular rates to determine how much smear to expect in their images.

CK files are capable of storing angular velocity data for instruments, although the presence of such data is optional. The CK reader CKGPAV (Get Pointing and Angular Velocity) returns an angular velocity vector in addition to a C-matrix.

Spacecraft Clock Time

Each piece of data within the C-kernel is associated with a spacecraft clock time (SCLK). This is because the spacecraft clock time is typically appended to the telemetry data that is the source for pointing information.

Within the SPICE system, SCLK is represented as an encoded double precision number. You will need this form when using CK reader routines to read from CK files.

SPICELIB includes routines to convert between character SCLK format and the double precision encoding. There are also routines to convert between SCLK and standard time systems such as ET and UTC.

The SCLK Required Reading contains a full description of SCLK including the clock formats for individual spacecraft. You should read that document before writing any C-kernel programs. A brief description of SCLK is included here because many of the subroutines presented require a clock time as an input argument.

Encoded SCLK

Encoded SCLK values may be discrete or continuous.

Discrete encoded SCLK values have units of ``ticks''; ticks represent the least significant counts representable by a clock. Continuous encoded SCLK supports non-integral tick values. This enables tranlation of other time systems to encoded SCLK without rounding.

Throughout this document, encoded SCLK should be assumed to be continuous unless otherwise specified.

To convert from a character string representation of SCLK to its double precision encoding, use the routine SCENCD (Encode SCLK):

```
   CALL SCENCD ( SC, SCLKCH, SCLKDP )
```

Use SCDECD (Decode SCLK) to recover the character representation from its double precision encoding.

```
   CALL SCDECD ( SC, SCLKDP, SCLKCH )
```

The first argument to both routines, SC, is the NAIF integer ID for the spacecraft whose clock count is being encoded or decoded (for example, -77 for Galileo).

Each spacecraft may have a different format for its clock counts, so the encoding scheme may be different for each. The SCLK Required Reading indicates the expected clock string formats for each mission.

To convert from ET to continuous encoded SCLK, use SCE2C (ET to continuous SCLK):

```
   CALL SCE2C ( SC, SCLKCH, SCLKDP )
```

To convert continuous encoded SCLK to ET, use SCT2E (Ticks to ET):

```
   CALL SCT2E ( SC, SCLKDP, ET )
```

Ticks and Partitions

The units of encoded SCLK are ``ticks since clock start at launch," where a ``tick" is defined to be the shortest time increment expressible by a particular spacecraft clock.

The problem of encoding SCLK is complicated by the fact that spacecraft clocks do not always advance continuously. A discontinuity may occur if a clock resets to a different value. This occurs when a clock reaches its maximum value, but it can also happen due to other reasons which will not be discussed here. Anytime this occurs, we say that the clock has entered a new ``partition."

SCLK strings should normally include a partition number prefixed to the rest of the clock count with a ``/". The partition number uniquely separates a count from identical counts in other partitions.

The presence of the partition number is not required. If it is missing, SCENCD will assume the partition to be the earliest possible one containing the clock string.

SCLK and other time systems

SPICELIB contains subroutines that convert between both the encoded and character form of spacecraft clock time and two other time systems.

The first is ephemeris time (ET), which is specified as some number of ephemeris seconds past a reference epoch. Within the SPICE system, state vectors of spacecraft and target bodies are referenced to ET seconds past the J2000 epoch.

The other is Coordinated Universal Time (UTC), which is also called Greenwich Mean Time. Two subroutine calls are necessary to convert between UTC and SCLK. One routine converts from SCLK to ET, and another from ET to UTC.

See Appendix A for a list of high level subroutines involved in spacecraft clock time conversions.

The SCLK kernel file

Before calling any of the SCLK conversion routines mentioned above, you have to load the contents of the SCLK kernel file into the kernel pool, using the routine LDPOOL (Load kernel pool).

The SCLK kernel file contains spacecraft specific parameters needed to perform the conversions. Included are such things as clock format definitions, partition start and stop times, and time interpolation constants. You should make sure that the kernel file you are using contains information for the particular spacecraft you are working with.

You also have to load the leapseconds kernel file into the kernel pool if you are going to convert between ET and UTC.

Basics

This chapter will present the easiest way to use C-kernel software to obtain pointing data from a CK file for a particular instrument. The mechanism for doing so is a ``reader,'' a subroutine which reads data from the C-kernel. The highest level readers will be discussed here; one that returns the C-matrix, and another that returns the C-matrix and angular velocity vector.

A later chapter will present lower level subroutines that allow the programmer to exert the highest amount of control in reading CK files.

Appendix B contains an example showing how some of the routines presented in this chapter fit together in a typical application program.

The CK File Reader CKGP

Below is a code fragment illustrating the use of the C-kernel file reader CKGP (Get Pointing). The example finds the C-matrix for the Voyager 2 narrow angle camera at a particular epoch during the Jupiter encounter. The C-matrix returned is a transformation from the J2000 frame to instrument-fixed coordinates.

Each of the subroutines used is briefly described below. See the individual subroutine headers for a complete description.

A complete description of how CKGP searches for pointing is provided in the ``Details'' chapter of this document.

```
      INTEGER                SC
      INTEGER                INST
      INTEGER                HANDLE

      DOUBLE PRECISION       SCLKDP
      DOUBLE PRECISION       TOL
      DOUBLE PRECISION       CLKOUT
      DOUBLE PRECISION       CMAT    ( 3, 3 )

      CHARACTER*(10)         REF

      LOGICAL                FOUND


C
C     NAIF ID numbers for the
C
C        1. Voyager 2 spacecraft
C        2. Voyager 2 narrow angle camera
C

      SC   = -32
      INST = -32001
C
C     The C-matrix should transform from J2000 to camera-fixed
C     coordinates.
C
      REF = 'J2000'

C
C     Load the spacecraft clock partition kernel file into the
C     kernel pool, for SCLK encoding and decoding.
C
      CALL LDPOOL ( 'vgr2_sclk.tsc' )

C
C     Load the C-kernel pointing file.
C
      CALL CKLPF ( 'vgr2_jup_inbound.bc', HANDLE )

C
C     We want pointing at a spacecraft clock time appearing in
C     the third spacecraft clock partition.
C
      CALL SCENCD ( SC, '3/20556:17:768', SCLKDP )
C
C     The Voyager 2 clock is of the form xxxxx yy www, where
C     yy is a modulus 60 counter.  Pictures were not shuttered
C     at intervals smaller than one mod 60 count.  Therefore,
C     use this as the tolerance.  ( Notice that no partition
C     number is used when specifying a tolerance )
C
```

```
            CALL SCTIKS ( SC, '0:01:000', TOL  )
      C
      C
      C     Get the pointing for the narrow angle camera.
      C
            CALL CKGP(INST, SCLKDP, TOL, REF, CMAT, CLKOUT, FOUND)
```

LDPOOL

LDPOOL loads the kernel pool with the contents of the specified file, which, in this case is the SCLK kernel file. (LDPOOL is not used to load S- and P-kernel files or C-kernel files.)

SCENCD (below) and SCDECD require the contents of the SCLK kernel file in order to properly encode and decode clock values. (See section on Spacecraft Clock Time).

CKLPF

CKLPF loads a CK file for processing by other CK routines. It takes as input the name of the C-kernel file to be used, in this example

```
   'vgr2_jup_inbound.bc'
```

It returns an integer called a ``handle,'' which is used much like a logical unit number in FORTRAN. All CK routines use handles instead of file names to refer to a specific file because handles contain implicit information about how a file may be accessed. You would need to use the handle to close the file, for example, or for use with some of the other DAF routines. But after loading you will not have to refer to the file, by name or handle, in your calls to CKGP.

Once loaded, a file is ready for any number of reads, so it needs to be loaded only once, typically in the initialization section of your program. Among other things, CKLPF opens the file with all the appropriate options, relieving you of that responsibility.

SCENCD and SCE2C

SCENCD encodes a character representation of spacecraft clock time such as

```
   '3/20556:17:768'
```
into a double precision number (SCLKDP). The value returned by SCENCD is a discrete tick count. When starting with an ET value, a continuous tick count may be obtained by calling SCE2C.

You must use encoded SCLK when calling CK reader routines.

SCTIKS

SCTIKS converts a clock string without partition number to units of ``ticks,'' which are the units of encoded SCLK returned by SCENCD.

The distinction between SCENCD and SCTIKS is important. The result of calling SCENCD is a relative measurement: ticks since the start of the clock at launch. The result of calling SCTIKS is an absolute measurement: ticks. It's like the difference between the times 3:55 p.m. (a specific time of the day) and 3:55 (three hours and fifty-five minutes - a length of time).

CKGP

CKGP looks through files loaded by CKLPF to find the data needed to compute the C-matrix for a specified spacecraft instrument at a particular time. It uses the following inputs and outputs.

Inputs are:

INST

The NAIF instrument ID. In this example, we want pointing for the Voyager 2 narrow angle camera (NAIF code -32001).

SCLKDP

Encoded SCLK time. Units are ``ticks since clock start at launch'' May be discrete or continuous.

TOL

SCLK time tolerance. TOL is measured in units of ``ticks.''
The pointing returned by CKGP will be for a time within TOL ticks of SCLKDP. In general, TOL should be smaller than the typical spacecraft clock time interval between instrument observations.

REF

The NAIF mnemonic for the base reference frame. The output C-matrix, if found, will be a transformation from REF to instrument-fixed coordinates.

See the FRAMES Required Reading for a list of those frames supported by the SPICE system, along with the accepted mnemonics for those frames.

Outputs are:

CMAT

The C-matrix. CMAT is a transformation matrix from the base frame REF to the instrument-fixed frame at the time CLKOUT.

CLKOUT

Continuous encoded spacecraft clock time for which CMAT is valid. This will be within TOL ticks of SCLKDP.

FOUND

Found flag. FOUND will be true if it was possible to return a C-matrix for INST for a time within TOL ticks of SCLKDP. FOUND will be false otherwise.

The CK File Reader CKGPAV

CKGPAV (Get Pointing and Angular Velocity) is almost identical to CKGP, except that it returns an angular velocity vector in addition to a C-matrix.

The calling sequence for CKGPAV is:

```
   CALL CKGPAV ( INST, SCLKDP, TOL, REF, CMAT, AV, CLKOUT, FOUND )
```

The angular velocity vector AV is a double precision array of size three. The components of AV are given relative to the base reference frame REF.

All of the other arguments are identical to those of CKGP. And, just as with CKGP, you must load a CK file by calling CKLPF before calling CKGPAV.

The behavior of CKGPAV is, however, slightly different from that of CKGP, and these differences will be explained in the ``Details'' chapter of this document.

Multiple Files and the C-kernel

There will probably be occasions when you will want to access pointing that is contained in more than one CK file. For instance, you may have several files describing pointing for several disjoint time periods, or for different instruments. Or you may have one file containing a partially updated version of another file's pointing.

In both cases, you would like to be able to get the pointing you want without having to run your application on each file separately. C-kernel software allows you to do this through the file loading and unloading process.

The file loading routine CKLPF was introduced in the last section. It was mentioned that you have to load the CK file before you try to access it, that you have to load it only once during program execution, and that in subsequent calls to CKGP, you don't have to refer to the file at all.

What was not mentioned was that multiple pointing files may be loaded and that CKGP will automatically search through as many of the files as necessary to satisfy the request.

If you have multiple files describing pointing for different time periods or different instruments, you can simply load them all at the beginning of your program, and then forget about which file covered what period or instrument. There is a hierarchy for searching, however, that you need to understand in case you happen to load files that have redundant coverage.

A request for pointing is satisfied by searching through the last loaded files first. Thus if we ran

```
   CALL CKLPF ( 'ckfile_1.bc', HANDL1 )
   CALL CKLPF ( 'ckfile_2.bc', HANDL2 )
   CALL CKLPF ( 'ckfile_3.bc', HANDL3 )
```

and then later made a request for pointing, the software would search through ckfile_3 first, ckfile_2 second, and ckfile_1 last.

This scheme is consistent with the fact that within an individual file, the data that were inserted last supersede those before them. In essence, loaded files are treated like one big file.

What if you have files representing different versions of the same pointing? This is a likely scenario considering there are tools (such as NAIF's C-smithing program) to update and ``improve'' pointing results.

For example, suppose you have one file containing predicted pointing values, and another containing improved, updated values. One approach would be to load the files in the following order:

```
   CALL CKLPF ( 'predict.bc', HANDL1 )
   CALL CKLPF ( 'update.bc',  HANDL2 )
```

This way, the ``better'' (updated) pointing file always gets searched first.

If, on the other hand, you want to be explicit about which file to search, you need a way of telling C-kernel software to stop looking in one file, and start looking in another. CKLPF accomplishes the latter by loading a file for processing. To tell C-kernel software to stop looking through a file, then, you need to unload it, with CKUPF (unload pointing file):

```
      INTEGER          HANDL1
      INTEGER          HANDL2

C
C     Load the first version.
C
      CALL CKLPF ( 'predict.bc', HANDL1 )
           .
           .  process pointing from first file.
           .
C
C     Unload the first version.
C
      CALL CKUPF ( HANDL1 )

C
C     Load the second version.
C
      CALL CKLPF ( 'update.bc', HANDL2 )
           .
           .  process pointing from the second file.
           .
```

Notice that to unload the file, you need to use that file's handle, as returned by CKLPF, and not the file's name.

Details

In the previous chapter, we introduced the two CK readers, CKGP and CKGPAV, which return C-matrices and angular velocity vectors from CK files.

In this chapter we introduce the concept of a CK file segment, and explain how these segments are organized into CK files. We then show exactly how CKGP and CKGPAV go about searching through files and segments to obtain the data that they need.

File Structure and Implementation

Each C-kernel file is made up of a number of ``segments.'' A segment is a set of logical records containing double precision numbers. When evaluated, each record gives a C-matrix and optionally, an angular velocity vector, of some spacecraft structure for some time within an interval. The segments in a file are ordered from beginning to end, with new segments added to the end of a file. The C-kernel readers use this ordering to check segments at the end of the file first.

Notice that the definition of a segment does not specify what type of record it contains. This vagueness is intentional. One of the primary features of the C-kernel is to provide a framework in which to store pointing data in any form, without users having to worry about that form when reading the data. Thus, different segments may contain different implementations of discrete or continuous data, but the same high-level readers are used to access all types.

In fact, there are only a couple of routines that are concerned with the internal data type of a segment. Other routines obtain all the information they need about a segment from two fields which precede each segment: ``descriptors'' and ``identifiers.'' Their formats are identical from segment to segment, and provide important information about the data contained inside.

Segment Descriptors

The C-kernel reader subroutines begin addressing the question, ``Can the request for pointing be satisfied by this segment?'' by looking at the descriptor.

A descriptor tells what instrument's pointing is being described, the interval of time for which the segment is valid, the reference frame of the internally stored data, and the segment data type.

Each segment descriptor contains two double precision components (DCD) and six integer components (ICD).

```
        -----------------------------------
DCD(1)  |  Initial SCLK                   |
        -----------------------------------
DCD(2)  |  Final SCLK                     |
        -----------------------------------
ICD(1)  |  Instrument    |
        -----------------
```

```
   ICD(2)  |  Reference     |
           ------------------
   ICD(3)  |  Data type     |
           ------------------
   ICD(4)  |  Rates Flag    |
           ------------------
   ICD(5)  |  Begin Address |
           ------------------
   ICD(6)  |  End   Address |
           ------------------
```

DCD(1),
DCD(2)

> The initial and final encoded spacecraft clock times for the segment.

ICD(1)

> The integer code of the instrument whose pointing is being described.

ICD(2)

> The NAIF integer ID of the base reference frame for the segment data. (For example, J2000, B1950, and so on --- to see which ID represents which coordinate system, see the Frames Required Reading.)

ICD(3)

> The data type of the segment. This indicates how the data is stored internally. The reader subroutines will use it to evaluate the data records. Typically, users will not have to know this code.

ICD(4)

> The angular rates flag. This indicates whether or not the segment is capable of producing angular velocity data. If ICD(4) = 0, then the segment contains pointing data only. If ICD(4) = 1, then the segment contains angular velocity data as well.

ICD(5),
ICD(6)

> Initial and final addresses of the segment data within the file. Users will typically not want or need to know about these addresses. They tell the readers where to go within a file to get the records needed to satisfy a particular request.

The descriptor is stored as a double precision array, with pairs of integer components equivalenced to double precision numbers. We say that the descriptor is ``packed'' into a double precision array. The size of a packed descriptor is five double precision numbers.

In the ``Looking at Descriptors'' section, you will be shown how to get a descriptor from a particular segment and ``unpack'' it into its double precision and integer components. You can then view the individual components.

Segment Identifiers

The idea behind a segment identifier is to provide a character field which allows a user to determine the exact origin of the segment.

For the most part, it will be up to the institution that creates a particular C-kernel segment to determine what goes in this free-format 40 character memory cell. However, it should be possible for users to look at a segment identifier and determine who knows the details about the creation of the segment.

For example, if a particular identifier looked like

```
   NAIF CSMITHING RET LOGA151
```

then a user should be able to contact NAIF to locate the right people to give the history of that segment: ephemerides used, source of pointing, assumptions, constraints, and so on.

Forty characters is not enough space to store all source information for every segment that might be built. Instead, the idea is to provide a pointer to the people or documents that will have all of the details about the source of the data.

Comment Area

In addition to segment identifiers, every binary CK file has a ``Comment Area'' for storing free-format textual information about the pointing data in the file. Ideally, each CK file will contain internal documentation that describes all of the details about the source of the data, its recommended use, and any other pertinent information. For example, the beginning and ending epochs for the file, the names and NAIF integer codes of the instruments included, an accuracy estimate, the date the file was produced, the names of the ephemeris files used, and any assumptions or constraints could be included. Comments about a particular segment in the file could refer to the segment by its identifier.

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, 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.

A CK file is a DAF

Each CK file is one implementation of a NAIF construct called a Double Precision Array File (DAF). DAFs are described in detail in reference [1]. Each CK segment is an instance of the DAF double precision array. The descriptor is an instance of a DAF ``summary''; the identifier is an instance of a DAF ``name.''

DAF routines are used at the lowest level to open, close, read, write and search CK files. As such, they allow for maximum flexibility in, for instance, examining a particular number within a segment, or searching for a particular segment within a file. Therefore, if the CK routines presented in this document do not allow you the control you want in looking through files, the DAF routines certainly will.

How the CK Readers Work

There are basically two steps to reading data from the C-kernel: locating the segment applicable to the request made, and evaluating the data contained inside the segment to return the C-matrix and angular velocity vector. In this section you'll see how these steps are implemented by CKGP and CKGPAV.

The General Search Algorithm

The CK readers search through files loaded by CKLPF to satisfy a pointing request. The files are searched in the reverse order from which they were loaded. Thus the last-loaded file is searched first, then the second to last, and so forth. The contents of individual files are also searched in backwards order, giving priority to segments that were added to a file later than the others.

The search ends when a segment is found that can give pointing for the specified instrument at a time falling within the specified tolerance on either side of the request time. Within that segment, the instance closest to the input time is located and returned.

The time for which pointing is being returned is not always the closest to the request time in all of the loaded files. The returned time is actually the closest time within the tolerance of the request time from the first segment that can satisfy the request. The algorithm works like this because it assumes that the last loaded files contain the highest quality pointing. Because segments are prioritized in this way users should not make their tolerance argument larger than the minimum spacing between the data in the files they are reading.

The following example illustrates this search procedure. Segments A and B are in the same file, with segment A located closer to the end of the file than segment B. Both segments A and B contain discrete pointing data.

```
                         SCLKDP    TOL
                              \   /
                               | |
                               |/ \
Request 1                  [---+---]
                           .   .   .
                           .   .   .
Segment A          (0-----------------0--------0--0-----0)
                           .   .   .
                           .   .   .
Segment B         (-0--0--0--0--0--0--0--0--0--0--0--0--0)
                                ^
                                |
                  CK reader returns this instance


                              SCLKDP
                                   \   TOL
                                    | /
                                    |/\
Request 2                        [--+--]
                                 .  .  .
                                 .  .  .
Segment A          (0-----------------0--------0--0-----0)
                                      ^
                                      |
                        CK reader returns this instance

Segment B         (0-0--0--0--0--0--0--0--0--0--0--0--0-0)
```

Segments that contain continuous pointing data are searched in the same manner as discrete segments. For request times that fall within the bounds of continuous intervals, the CK reader will return pointing at the request time. When the request time does not fall within an interval, then a time at an endpoint of an interval may be returned if it is the closest time in the segment to the user request time and also within the tolerance.

In the following examples segment A contains discrete pointing data and segment C contains continuous data. Segment A is located closer to the end of the file than segment C.

```
                    SCLKDP
                         \  TOL
```

```
                                  | /
                                  |/\
   Request 3                   [--+--]
                               .  .  .
                               .  .  .
   Segment A          (0-----------------0--------0--0-----0)
                               .  .  .
                               .  .  .
   Segment C          (--[=============]---[====]------[=]--)
                                  ^
                                  |
                     CK reader returns this instance
```

In the next example assume that the order of segment A and C in file are reversed.

```
                                   SCLKDP
                                        \   TOL
                                         | /
                                         |/\
   Request 4                          [--+--]
                                      .  .  .
                                      .  .  .
   Segment C          (--[=============]---[====]------[=]--)
                                           ^
                                           |
                             CK reader returns this instance

   Segment A          (0-----------------0--------0--0-----0)
```

The next few sections will go into greater detail about how CKGP and CKGPAV search through segments.

The Difference Between CKGP and CKGPAV

The only significant difference between the search algorithms of CKGP and CKGPAV is in which segments they search through to satisfy a request for pointing data. Recall that segments in a CK file only optionally contain angular velocity data. Since CKGP does not return an angular velocity vector, it is free to consider all segments when satisfying a request, because all segments will contain the data for constructing C-matrices. CKGPAV, on the other hand, will consider only those segments which also contain angular velocity data.

Because of this difference, it is possible that on the exact same set of inputs, CKGP and CKGPAV could return different values for the C-matrix. This could occur if a CK file contained two segments covering the same time period for the same instrument, one with angular rates and one without. CKGP might use the C-matrix only segment, whereas CKGPAV would ignore that segment and use the one containing angular velocity data.

To avoid this situation, NAIF advises users not to place segments with and without angular velocity data in the same file.

Locating the Applicable Segment

Within CKGP and CKGPAV, finding the right segment is the job of CKBSS (Begin a Search for a Segment), and CKSNS (Select the Next Segment).

CKBSS and CKSNS are both entry points to the routine CKBSR (Buffer Segments for Readers).

CKBSS establishes a search for segments. It records the desired instrument (INST), SCLK time (SCLKDP), and SCLK tolerance (TOL) for the search. It also records the need for angular velocity --- NEEDAV is true if angular velocity data is being requested, false otherwise.

CKSNS then uses DAF routines to search through loaded files to find a segment matching the criteria established in the call to CKBSS. Last-loaded files get searched first, and within a single file, segments get checked starting from the end of the file and going backwards.

When an applicable segment is found, the descriptor and identifier for that segment, and the handle of the file containing the segment, are returned, and the readers output logical flag FOUND is set to true. If no applicable segment is found, FOUND is false.

If a segment is found, but is subsequently found to be inadequate, CKSNS can be called again to find the next applicable segment using the searching order described above.

CKSNS can be called any number of times after a search has been started by CKBSS, and will just return a false value for FOUND whenever applicable segments have run out.

Because CKSNS is called every time a request is made, an internal buffer of segment descriptors is maintained by CKBSR to keep from performing superfluous file reads. You can adjust the size of the buffer by changing the parameter STSIZE in CKBSR.

Looking at Descriptors

The descriptor and handle returned by CKSNS are used by other CK routines to locate and evaluate the pointing records. In order to do so, those routines have to unpack a descriptor into its double precision and integer parts, using the DAF routine DAFUS (Unpack Summary).

Evaluating the Records --- the Reader CKPFS

After locating an appropriate segment via CKSNS, CKGP and CKGPAV evaluate pointing records with a call to CKPFS (Pointing From Segment), a low level CK reader.

CKPFS takes as input the handle and descriptor of the applicable file and segment, along with the time specifications and angular velocity flag.

CKPFS returns the C-matrix and, if requested, the angular velocity vector for the time in the segment closest to SCLKDP and within TOL ticks of it. If CKPFS can't locate a time close enough in the segment, then FOUND is set to false. (If FOUND is false, then CKGP and CKGPAV will try another segment by calling CKSNS again, then CKPFS again, and so on.)

The output data are referenced to the base frame indicated by the descriptor. In other words, at this point, CMAT is a transformation from the base frame specified by ICD(2) to instrument-fixed coordinates, and the coordinates of AV lie in that same base frame.

Transforming the Results

The final task performed by CKGP and CKGPAV is to transform the returned data from their stored reference frame to that requested by the calling program.

First, the routines compare the NAIF ID for the requested frame with that of the stored frame. If the requested frame matches the segment frame, there is nothing to be done. Otherwise, the C-matrix and angular velocity vector have to be transformed.

Recall that the C-matrix returned by CKPFS is a rotation matrix from a base frame (call it REFSEG) to instrument-fixed coordinates:

```
   [          ] I-fixed
   |          |
   |   CMAT   |
   |          |
   [          ] REFSEG
```

What we want is a rotation matrix from the requested frame (call it REFREQ) to instrument-fixed coordinates:

```
   [          ] I-fixed
   |          |
   |    CMAT    |
   |            |
   [            ] REFREQ
```
So all we have to do is multiply the returned C-matrix by a rotation matrix, call it RMAT, from the requested frame to the one specified in the segment:

```
   [            ] I-fixed      [            ] I-fixed  [            ] REFSEG
   |            |              |            |          |            |
   |    CMAT    |          =   |    CMAT    |          |    RMAT    |
   |            |              |            |          |            |
   [            ] REFREQ       [            ] REFSEG   [            ] REFREQ
```
Once you have RMAT, it is a trivial matter to transform the angular velocity vector. Its coordinates, upon return from CKPFS, are in the frame REFSEG.

Data Types

The C-kernel framework for providing pointing data has been designed for flexibility. Different methods of storing and evaluating the data can be implemented independently of the high-level routines used to read the data. The only real restriction is that each segment must be stored as an array of double precision numbers.

Each method of storing and evaluating the data contained in a segment defines a different ``data type.'' The data type of a segment is specified by the third integer component of the the segment descriptor. The integer code for a data type is equal to the number of that type. For example, a segment of data type 1 would have the third integer component of its descriptor equal to 1. A data type need not accommodate angular velocity data. If it can't, all segments of that data type would have the value of the fourth integer component of the descriptor set equal to zero, which indicates that the segment does not contain angular velocity data.

The CK reader that makes a distinction between segments of different data types is the low level reader CKPFS. The main body of CKPFS consists of a case statement of the form:

```
         IF       ( TYPE .EQ. 1 ) THEN
            .
            .
            .
         ELSE  IF ( TYPE .EQ. 2 ) THEN
            .
            .
            .
```

```
ELSE  IF ( TYPE .EQ. N ) THEN
   .
   .
   .
ELSE

  CALL SETMSG( 'The data type # is not currently supported.')
  CALL ERRINT( '#', TYPE                         )
  CALL SIGERR( 'SPICE(CKUNKNOWNDATATYPE)' )

END IF
```

Once CKPFS determines the data type of a segment, two type-specific routines are called. The first, CKRxx, reads a segment of type xx and returns the information from the segment necessary to evaluate pointing at a particular time. The second routine CKExx evaluates the information returned by CKRxx, producing a C-matrix, and if requested, an angular velocity vector.

There are currently three supported CK data types in SPICELIB and they are described in detail in the sections that follow.

Data Type 1

The following method of storing and evaluating discrete pointing and angular rate values defines C-kernel data type 1.

Each pointing instance is stored as a four-tuple called a ``quaternion.'' Quaternions are widely used to represent rotation matrices. They require less than half the space of 3x3 matrices and finding the rotation matrix given by a quaternion is faster and easier than finding it from, say, RA, Dec, and Twist. In addition, other computations involving rotations, such as finding the rotation representing two successive rotations, may be performed on the quaternions directly.

The four numbers of a quaternion represent a unit vector and an angle. The vector represents the axis of a rotation, and the angle represents the magnitude of that rotation. If the vector is U = (u1, u2, u3), and the angle is T, then the quaternion Q is given by:

```
Q = ( q0, q1, q2, q3 )
  = ( cos(T/2), sin(T/2)*u1, sin(T/2)*u2, sin(T/2)*u3 )
```

The details of quaternion representations of rotations, and the derivations of those representations are documented in the SPICELIB Required Reading file ROTATIONS.

Data type 1 provides the option of including angular velocity data. If such data is included, the angular velocity vector A = (a1, a2, a3) corresponding to each pointing instance will be stored as itself. The coordinates of the vector will be in the same base reference frame as that of the C-matrix quaternions.

A type 1 pointing record consists of either four or seven double precision numbers; four for the C-matrix quaternion, and, optionally, three for the angular velocity vector.

```
+--------+--------+--------+--------+--------+--------+--------+
|  q     |  q     |  q     |  q     |  a     |  a     |  a     |
|   0    |   1    |   2    |   3    |   1    |   2    |   3    |
+--------+--------+--------+--------+--------+--------+--------+
```

Every type 1 segment has four parts to it:

```
+----------------------------------------------------------+
|                                                          |
|                                                          |
|                         Pointing                         |
|                                                          |
|                                                          |
+----------------------------------------------------------+
|                 |
|                 |
|    SCLK times   |
|                 |
|                 |
+-----------------+
|                 |
|  SCLK directory |
|                 |
+-----------------+
|      NPREC      |
+-----------------+
```

The final component, NPREC, gives the total number of pointing instances described by the segment.

Preceding it, starting from the top, are NPREC pointing records, ordered with respect to time, each consisting of the four or seven double precision numbers described above.

Following the pointing section are the NPREC encoded spacecraft clock times corresponding to the pointing records. These must be in strictly increasing order.

Following the SCLK times is a very simple SCLK directory. The directory contains INT((NPREC-1) / 100) entries. The Ith directory entry contains the midpoint of the (I*100)th and the (I*100 + 1)st SCLK time. Thus,

```
Directory(1) = ( SCLKDP(100) + SCLKDP(101) )   / 2

Directory(2) = ( SCLKDP(200) + SCLKDP(201) )   / 2
```

and so on.

If there are 100 or fewer entries, there is no directory. The directory is used to narrow down searches for pointing records to groups of 100 or less. Midpoints of adjacent times are used so that if an input time falls on one side of the directory time, then the group represented by that side is guaranteed to contain the time closest to the input time.

Type 1 subroutines

There are several CK subroutines that support data type 1. Their names and functions are:

CKW01

writes a type 1 segment to a file.

CKR01

reads a pointing record from a type 1 segment that satisfies a request for pointing at a given time.

CKE01

evaluates the record supplied by CKR01.

CKNR01

gives the number of pointing instances in a type 1 segment.

CKGR01

gets the Ith pointing instance from a type 1 segment.

Data Type 2

The following method of storing and evaluating continuous pointing data for a spacecraft structure defines C-kernel data type 2.

A type 2 segment consists of disjoint intervals of time during which the angular velocity of the spacecraft is constant. Thus, throughout an interval, the spacecraft structure rotates from its initial position about a fixed right-handed axis defined by the direction of the angular velocity vector at a constant rate equal to the magnitude of that vector.

A type 2 CK segment contains the following information for each interval:

1. The encoded spacecraft clock START and STOP times for the interval.
2. The quaternion representing the C-matrix associated with the start time of the interval.
3. The constant angular velocity vector, in radians per second, for the interval.
4. A factor which relates seconds and encoded SCLK ticks. This is necessary to convert the difference between the requested and interval start times from SCLK to seconds.

The orientation of a spacecraft structure may be determined from the above information at any time that is within the bounds of one of the intervals.

Every type 2 segment is organized into four parts.

```
+------------------------------------------------------------------+
|                                                                  |
|                                                                  |
|                            Pointing                              |
|                                                                  |
|                                                                  |
+------------------------------------------------------------------+
|                    |
|                    |
|  SCLK start times  |
|                    |
|                    |
+--------------------+
|                    |
|                    |
|  SCLK stop times   |
|                    |
|                    |
+--------------------+
|                    |
|  SCLK directory    |
|                    |
+--------------------+
```

The first part of a segment contains pointing records which are ordered with respect to their corresponding interval start times. A type 2 pointing record contains eight double precision numbers in the following form:

```
+-------+-------+-------+-------+-------+-------+-------+------+
|       |       |       |       |       |       |       |      |
|  q0   |  q1   |  q2   |  q3   |  a1   |  a2   |  a3   | rate |
|       |       |       |       |       |       |       |      |
+-------+-------+-------+-------+-------+-------+-------+------+
```

The first four elements are the components of the quaternion Q = (q0,q1,q2,q3) that is used to represent the C-matrix associated with the start time of the interval. Next are the three components of the angular velocity vector A = (a1,a2,a3) which are given with respect to the base reference frame specified in the segment descriptor.

The last element is a rate which converts the difference between the requested and interval start time from encoded SCLK ticks to seconds.

For segments containing predict data, this factor will be equal to the nominal amount of time represented by one tick of the particular spacecraft's clock. The nominal rate is given here for several spacecraft.

```
   spacecraft                     seconds / tick ( sec )
   ---------------------          ----------------------
   Galileo                        1 / 120
   Mars Global Surveyor           1 / 256
   Voyager I and II               0.06
```

For segments based on real rather than predicted pointing, the rate at which the spacecraft clock runs relative to ephemeris time will deviate from the nominal rate. The creator of the segment will need to determine an average value for this rate over the time period of the interval.

Located after the pointing data are the interval START times followed by the STOP times.

The START and STOP times should be ordered and in encoded SCLK form. The intervals should be disjoint except for possibly at the endpoints. If an input request time falls on an overlapping endpoint then the interval used will be the one corresponding to the start time. Degenerate intervals in which the STOP time equals the START time are not allowed.

Following the STOP times is a very simple directory of spacecraft clock times containing INT((NPREC-1)/100) entries, where NPREC is the number of pointing intervals. The Ith directory entry contains the midpoint of the (I*100)th STOP and the (I*100 + 1)st START SCLK time.

```
   Thus,

   Directory(1) = ( STOP(100) + START(101) )   / 2

   Directory(2) = ( STOP(200) + START(201) )   / 2

   .
   .
   .
```

If there are 100 or fewer entries then there is no directory. The directory is used to narrow down searches for pointing records to groups of 100 or less.

Type 2 subroutines

There are several CK subroutines that support data type 2. Their names and functions are:

CKW02

writes a type 2 segment to a file.

CKR02

reads a pointing record from a type 2 segment that satisfies a request for pointing at a given time.

CKE02

evaluates the record supplied by CKR02.

CKNR02

gives the number of pointing records in a type 2 segment.

CKGR02

gets the Ith pointing record from a type 2 segment.

Data Type 3

The following method of storing and evaluating discrete pointing data for a spacecraft structure defines C-kernel data type 3.

A type 3 segment consists of discrete pointing instances that are partitioned into groups within which linear interpolation between adjacent pointing instances is valid. Since the pointing instances in a segment are ordered with respect to time, these groups can be thought of as representing intervals of time over which the pointing of a spacecraft structure is given continuously. Therefore, in the description that follows, these groups of pointing instances will be referred to as interpolation intervals.

All of the pointing instances in the segment must be ordered by encoded spacecraft clock time and must belong to one and only one interpolation interval. The intervals must begin and end at times for which there are pointing instances in the segment. The CK software that evaluates the data in the segment does not extrapolate pointing past the bounds of the intervals.

A user's view of the time coverage provided by a type 3 segment can be viewed pictorially as follows:

```
   pointing instances:     0-0-0-0-0----0-0-0-0-0-----0------0-0-0-0
                           |       |    |       |     |      |     |
   interval bounds:       BEG      |   BEG      |    BEG    BEG    |
                                  END          END   END          END
```

In the above picture, the zeros indicate the times associated with the discrete pointing instances and the vertical bars show the bounds of the interpolation intervals that they are partitioned into. Note that the intervals begin and end at times associated with pointing instances. Also note that intervals consisting of just a single pointing instance are allowed.

When pointing is desired for a time that is within the bounds of one of the intervals, the CK reader routines return interpolated pointing at the request time. In the example below, the pointing request time is indicated by SCLKDP and the user supplied tolerance is given by TOL. In this example the tolerance argument of the CK readers could be set to zero and pointing would still be returned.

```
                                  SCLKDP    TOL
                                       \   /
                                        | |
                                        |/ \
                                    [---+---]
                                    .   .   .
                                    .   .   .
pointing instances:     0-0-0-0-0----0-0-0-0-0-----0------0-0-0-0
                        |       |    |  ^    |     |      |     |
interval bounds:       BEG      |   BEG |    |    BEG    BEG    |
                               END      |   END   END          END
                                        |
         CK reader returns interpolated pointing at this time.
```

When a request time falls in a gap between intervals, no extrapolation is performed. Instead, pointing is returned for the interval endpoint closest to the request time, provided that time is within the user supplied tolerance. In this example if the tolerance were set to zero no pointing would be returned.

```
                                         SCLKDP
                                              \   TOL
                                               | /
                                               |/\
                                           [---+---]
                                           .   .   .
                                           .   .   .
pointing instances:     0-0-0-0-0----0-0-0-0-0-----0------0-0-0-0
                        |       |    |       |     |      |     |
interval bounds:       BEG      |   BEG      |    BEG    BEG    |
                               END          END   END          END
                                             ^
                                             |
                             CK reader returns this instance.
```

The physical structure of the data stored in a type 3 segment is as follows:

```
+-----------------------------------------------------------------+
|                                                                 |
|                                                                 |
|                          Pointing                               |
|                                                                 |
|                                                                 |
+-----------------------------------------------------------------+
|                        |
|  SCLK times            |
|                        |
+------------------------+
|                        |
|  SCLK directory        |
|                        |
+------------------------+
|                        |
|  Interval start times  |
|                        |
+------------------------+
|                        |
|  Start times directory |
|                        |
+------------------------+
|                        |
|  Number of intervals   |
|                        |
+------------------------+
|                        |
|  Number of pointing    |
|      instances         |
|                        |
+------------------------+
```
In the discussion that follows let NPREC be the number of pointing instances in the segment and let NUMINT be the number of intervals into which the pointing instances are partitioned.

The first part of a segment contains NPREC pointing records which are ordered with respect to increasing time. Depending on whether or not the segment contains angular velocity data, a type 3 pointing record contains either four or seven double precision numbers in the following form:

```
+--------+--------+--------+--------+--------+--------+--------+
|        |        |        |        |        |        |        |
|   q0   |   q1   |   q2   |   q3   |   a1   |   a2   |   a3   |
|        |        |        |        |        |        |        |
+--------+--------+--------+--------+--------+--------+--------+
```
The first four elements are the components of the quaternion Q = (q0,q1,q2,q3) that is used to represent the pointing of the instrument or spacecraft structure to which the segment applies. Next are the three components of the angular velocity vector AV = (a1,a2,a3) which are given with respect to the base reference frame specified in the segment descriptor. These components are optional and are present only if the segment contains angular velocity data as specified by the fourth integer component of the segment descriptor.

Following the pointing data are the NPREC times associated with the pointing instances. These times are in encoded SCLK form and should be strictly increasing.

Immediately following the last time is a very simple directory of the SCLK times. The directory contains INT((NPREC-1) / 100) entries. The Ith directory entry contains the (I*100)th SCLK time. Thus,

```
   Directory(1) = SCLKDP(100)

   Directory(2) = SCLKDP(200)

    .
    .
    .
```

If there are 100 or fewer entries, there is no directory. The directory is used to narrow down searches for pointing records to groups of 100 or less.

Next are the NUMINT start times of the intervals that the pointing instances are partitioned into. These times are given in encoded spacecraft clock and must be strictly increasing. They must also be equal to times for which there are pointing instances in the segment. Note that the interval stop times are not stored in the segment. They are not needed because the stop time of the Ith interval is simply the time associated with the pointing instance that precedes the start time of the (I+1)th interval.

Following the interval start times is a directory of these times. This directory is constructed in a form similar to the directory for the times associated with the pointing instances. The start times directory contains INT ((NUMINT-1) / 100) entries and contains every 100th start time. Thus:

```
   Directory(1) = START(100)

   Directory(2) = START(200)

    .
    .
    .
```

Finally, the last two words in the segment give the total number of interpolation intervals (NUMINT) and the total number of pointing instances (NPREC) in the segment.

A segment writer routine is provided which calls the low level DAF routines necessary to write a type 3 segment to a C-kernel. However, the creator of the segment is responsible for determining whether or not it is valid to interpolate between adjacent pointing instances, and thus how they should be partitioned into intervals. See the header of the routine CKW03 for a complete description of the inputs required to write a segment.

Linear Interpolation Algorithm

The linear interpolation performed between adjacent pointing instances by the CK software is defined by the following algorithm:

1. Let t be the time for which pointing is desired and let CMAT1 and CMAT2 be C-matrices associated with times t1 and t2 such that:

```
                    t1 <= t <= t2,  where t1 < t2.
```

2. Assume that the spacecraft frame rotates about a fixed axis at a constant angular rate from time t1 to time t2. Then the rotation axis and angle can be derived from the rotation matrix ROT12 where:

```
                         T                       T
                    CMAT2   =  ROT12    *   CMAT1

         or
                                    T
                    ROT12   =  CMAT2    *   CMAT1
```

3. Obtain the axis and angle of the rotation from the matrix ROT12. Let the axis vector of the rotation be AXIS and the rotation angle be ANGLE.

4. To obtain pointing information at time t, rotate the spacecraft frame about the vector AXIS from its orientation at time t1 by the angle THETA where:

```
                                               ( t  - t1 )
                    THETA  =  ANGLE  *   -----------
                                               ( t2 - t1 )
```

5. Thus if ROT1t is the matrix that rotates vectors by the angle THETA about the vector AXIS, then the desired C-matrix is given by:

```
                        T                    T
                    CMAT  =  ROT1t   *   CMAT1

                                             T
                    CMAT  =  CMAT1   *   ROT1t
```

6. The angular velocity is treated independently of the C-matrix. If it is requested, then the AV at time t is the weighted average of the angular velocity vectors at time t1 and time t2:

```
                          ( t  - t1 )
                    W  =  -----------
                          ( t2 - t1 )


                    AV  = ( 1 - W ) * AV1   +   W * AV2
```

Type 3 subroutines

There are several CK subroutines that support data type 3. Their names and function are:

CKW03

writes a type 3 segment to a file.

CKR03

reads a pointing record from a type 3 segment that satisfies a request for pointing at a given time.

CKE03

evaluates the record supplied by CKR03.

CKNR03

gives the number of pointing instances in a type 3 segment.

CKGR03

gets the Ith pointing instance from a type 3 segment.

Appendix A --- Summary of C-kernel Subroutines

Summary of Mnemonics

Each C-kernel subroutine name consists of a mnemonic which translates into a short description of the routine's purpose. Those beginning with ``CK'' are names of routines that deal solely with C-kernel files. The other routines provide support that is not necessarily C-kernel specific.

Many of the routines listed below are entry points to another subroutine. If they are, the parent routine's name will be listed inside brackets preceding the mnemonic translation.

```
       C-kernel Routines


CKBSS  [CKBSR] ( C-kernel, begin search for segment                )
CKCLS          ( C-kernel, close a pointing file                   )
CKE01          ( C-kernel, evaluate pointing record, data type 1  )
CKE02          ( C-kernel, evaluate pointing record, data type 2  )
CKE03          ( C-kernel, evaluate pointing record, data type 3  )
CKGP           ( C-kernel, get pointing                            )
CKGPAV         ( C-kernel, get pointing and angular velocity       )
CKGR01         ( C-kernel, get record, data type 1                 )
CKGR02         ( C-kernel, get record, data type 2                 )
CKGR03         ( C-kernel, get record, data type 3                 )
CKLPF  [CKBSR] ( C-kernel, load pointing file                      )
CKNR01         ( C-kernel, number of records, data type 1          )
CKNR02         ( C-kernel, number of records, data type 2          )
CKNR03         ( C-kernel, number of records, data type 3          )
CKOPN          ( C-kernel, open a new pointing file                )
CKPFS          ( C-kernel, pointing from segment                   )
CKR01          ( C-kernel, read pointing record, data type 1       )
CKR02          ( C-kernel, read pointing record, data type 2       )
CKR03          ( C-kernel, read pointing record, data type 3       )
CKSNS  [CKBSR] ( C-kernel, select next segment                     )
CKUPF  [CKBSR] ( C-kernel, unload pointing file                    )
CKW01          ( C-kernel, write segment to C-kernel, data type 1 )
CKW02          ( C-kernel, write segment to C-kernel, data type 2 )
CKW03          ( C-kernel, write segment to C-kernel, data type 3 )

       SCLK conversion routines

SCDECD         ( Decode spacecraft clock                 )
SCENCD         ( Encode spacecraft clock                 )
SCPART         ( Spacecraft clock partitions             )
SCFMT          ( Spacecraft clock format                 )
SCTIKS         ( Spacecraft clock ticks                  )
SCT2E          ( Convert encoded SCLK Ticks to ET        )
SCS2E          ( Convert SCLK String to ET               )
SCE2C          ( Convert ET to continuous SCLK Ticks     )
SCE2T          ( Convert ET to encoded SCLK Ticks        )
SCE2S          ( Convert ET to SCLK String               )

UTC2ET         ( UTC to Ephemeris Time                   )
ET2UTC         ( Ephemeris Time to UTC                   )




       Reference frame routines

IRFROT [CHGIRF] ( Inertial reference frame, rotate      )
IRFNUM [CHGIRF] ( Inertial reference frame number       )
IRFNAM [CHGIRF] ( Inertial reference frame name         )
IRFDEF [CHGIRF] ( Inertial reference frame, default     )
```

Summary of Calling Sequences

```
         C-kernel Routines

  CKLPF  ( FNAME,  HANDLE                                          )
  CKUPF  (         HANDLE                                          )
  CKBSS  ( INST,   SCLKDP, TOL,    NEEDAV                          )
  CKSNS  ( HANDLE, DESCR,  SEGID,  FOUND                           )
  CKGP   ( INST,   SCLKDP, TOL,    REF,   CMAT,   CLKOUT, FOUND   )
  CKGPAV ( INST,   SCLKDP, TOL,    REF,   CMAT,   AV,     CLKOUT,
           FOUND                                                   )
  CKPFS  ( HANDLE, DESCR,  SCLKDP, TOL,   NEEDAV, CMAT,   AV,
           CLKOUT, FOUND                                           )
  CKOPN  ( FNAME,  IFNAME, NCOMCH, HANDLE                          )
  CKCLS  ( HANDLE                                                  )
  CKR01  ( HANDLE, DESCR,  SCLKDP, TOL,   NEEDAV, RECORD, FOUND   )
  CKE01  ( NEEDAV, RECORD, CMAT,   AV,    CLKOUT                   )
  CKW01  ( HANDLE, BEGTIM, ENDTIM, INST,  REF,    AVFLAG, SEGID,
           NPREC,  SCLKDP, QUATS,  AVVS                            )
  CKNR01 ( HANDLE, DESCR,  NPREC                                   )
  CKGR01 ( HANDLE, DESCR,  RECNO,  RECORD                          )
  CKR02  ( HANDLE, DESCR,  SCLKDP, TOL,   RECORD, FOUND            )
  CKE02  ( NEEDAV, RECORD, CMAT,   AV,     CLKOUT                  )
  CKW02  ( HANDLE, BEGTIM, ENDTIM, INST,  REF,    SEGID,  NPREC,
           START,  STOP,   QUATS,  AVVS,  RATES                    )
  CKNR02 ( HANDLE, DESCR,  NPREC                                   )
  CKGR02 ( HANDLE, DESCR,  RECNO,  RECORD                          )
  CKR03  ( HANDLE, DESCR,  SCLKDP, TOL,   NEEDAV, RECORD, FOUND   )
  CKE03  ( NEEDAV, RECORD, CMAT,   AV,    CLKOUT                   )
  CKW03  ( HANDLE, BEGTIM, ENDTIM, INST,  REF,    AVFLAG, SEGID,
           NPREC,  SCLKDP, QUATS,  AVVS,  NINTS,  STARTS           )
  CKNR03 ( HANDLE, DESCR,  NPREC                                   )
  CKGR03 ( HANDLE, DESCR,  RECNO,  RECORD                          )

         SCLK conversion routines

  SCDECD ( SC,     SCLKDP, SCLKCH        )
  SCENCD ( SC,     SCLKCH, SCLKDP        )
  SCPART ( SC,     NPARTS, PSTART, PSTOP )
  SCFMT  ( SC,     TICKS,  CLKSTR        )
  SCTIKS ( SC,     CLKSTR, TICKS         )
  SCT2E  ( SC,     SCLKDP, ET            )
  SCS2E  ( SC,     SCLKCH, ET            )
  SCE2C  ( SC,     ET,     SCLKDP        )
  SCE2T  ( SC,     ET,     SCLKDP        )
  SCE2S  ( SC,     ET,     SCLKCH        )

  UTC2ET ( UTCSTR, ET                    )
  ET2UTC ( ET,     FORMAT, PREC,  UTCSTR )
```

```
       Reference frame routines

IRFROT ( REFA,   REFB,   ROTAB )
IRFNUM ( NAME,   INDEX         )
IRFNAM ( INDEX,  NAME          )
IRFDEF ( INDEX                 )
```

Appendix B --- Example Program PLANET_POINT

The following program shows how C-kernel subroutines fit together with other SPICELIB routines to solve a typical problem requiring pointing data.

All of the subroutines used here are part of SPICELIB.

```
      PROGRAM PLANET_POINT
      IMPLICIT NONE

C
C     Compute the planetocentric latitude, longitude and radius
C     coordinates of the point at which the optic axis of an
C     instrument intersects the surface of a target planet.
C     Assume that the axis of the instrument is along the Z-axis
C     of the instrument fixed reference frame.
C
C     The following files are required:
C
C        1) Kernel file containing planetary constants.
C        2) Kernel file containing spacecraft clock (SCLK) data.
C        3) SPK file containing planetary and spacecraft
C           ephemeris data.
C        4) CK file containing instrument pointing data.
C
C     The following quantities are required:
C
C        1) NAIF integer spacecraft ID
C        2) NAIF integer planet ID
C        3) NAIF integer instrument ID
C        4) SCLK time string
C        5) SCLK tolerance.
C
C     The following steps are taken to locate the desired point:
C
```

```
C        1) The inertial pointing (VPNT) of the instrument at
C           the input SCLK time is read from the CK file.
C
C        2) The apparent position (VTARG) is computed for the
C           center of the target body as seen from the spacecraft,
C           at the ephemeris time (ET) corresponding to SCLK.
C
C           The one-way light time (TAU) from the target to the
C           spacecraft is also computed.
C
C        3) The transformation (TIBF) from inertial to body-fixed
C           coordinates is computed for the epoch ET-TAU, using
C           quantities from the planetary constants kernel.
C
C        4) The radii (R) of the tri-axial ellipsoid used to model
C           the target body are extracted from the planetary
C           constants kernel.
C
C        5) The position of the observer, in body-fixed coordinates
C           is computed using VTARG and TIBF.
C
C        6) VPNT is converted to body-fixed coordinates using TIBF.
C
C        7) The routine SURFPT computes the point of intersection,
C           given the two body-fixed positions, and tri-axial
C           ellipsoid radii.
C
C$ Particulars
C
C     1) The instrument boresight is assumed to define the z-axis
C        of the instrument-fixed reference frame. This is reflected
C        in the choice of ( 0, 0, 1 ) as the boresight pointing
C        vector (VPNT) in instrument-fixed coordinates.
C
C$ Declarations

      INTEGER               FILEN
      PARAMETER           ( FILEN  = 128 )

      INTEGER               TIMLEN
      PARAMETER           ( TIMLEN =  30 )

      INTEGER               FRMLEN
      PARAMETER           ( FRMLEN =  20 )


      CHARACTER*(FILEN)     FILE
      CHARACTER*(TIMLEN)    SCLKCH
      CHARACTER*(TIMLEN)    TOLCH
      CHARACTER*(FRMLEN)    REF

      INTEGER               HANDL1
      INTEGER               HANDL2
      INTEGER               SC
      INTEGER               INST
      INTEGER               TARG
      INTEGER               N
      DOUBLE PRECISION      SCLKDP
      DOUBLE PRECISION      ET
      DOUBLE PRECISION      TOL
      DOUBLE PRECISION      CMAT    ( 3, 3 )
      DOUBLE PRECISION      CLKOUT
      DOUBLE PRECISION      VTARG   ( 6 )
      DOUBLE PRECISION      TAU
      DOUBLE PRECISION      TIBF    ( 3, 3 )
      DOUBLE PRECISION      R       ( 3 )
      DOUBLE PRECISION      VPOS    ( 3 )
      DOUBLE PRECISION      VSURF   ( 3 )
      DOUBLE PRECISION      VPNT    ( 3 )
      DOUBLE PRECISION      RADIUS
      DOUBLE PRECISION      LONG
      DOUBLE PRECISION      LAT

      LOGICAL               FOUND

C
C     Initial values
C

C
C     The inertial reference frame for all output.
C
      DATA   REF     / 'J2000' /

C
C     The boresight vector is assumed to define the z-axis of the
C     instrument-fixed frame.
C
      DATA   VPNT    / 0.D0, 0.D0, 1.D0 /

C
C     Get all of the files, and load them.
C

      WRITE (*,*) 'Enter the name of the kernel file containing'//
     .            ' planetary constants:'
      READ (*,FMT='(A)') FILE

      CALL LDPOOL ( FILE )

      WRITE (*,*)
      WRITE (*,*) 'Enter the name of the kernel file containing'//
     .            ' SCLK coefficients:'
      READ (*,FMT='(A)') FILE

      CALL LDPOOL ( FILE )

      WRITE (*,*)
      WRITE (*,*) 'Enter the name of the SPK file containing' //
     .            ' planetary and spacecraft ephemerides:'
      READ (*,FMT='(A)') FILE

      CALL SPKLEF ( FILE, HANDL1 )
      WRITE (*,*)
      WRITE (*,*) 'Enter the name of the CK file containing' //
     .            ' instrument pointing:'
      READ (*,FMT='(A)') FILE

      CALL CKLPF ( FILE, HANDL2 )

C
C     Get the ID codes for spacecraft, instrument, and target body.
C
      WRITE (*,*)
      WRITE (*,*) 'Enter NAIF integer spacecraft ID:'
      READ  (*,*)  SC
      WRITE (*,*)
      WRITE (*,*) 'Enter NAIF integer instrument ID:'
      READ  (*,*)  INST
      WRITE (*,*)
      WRITE (*,*) 'Enter NAIF integer ID for the target body:'
      READ  (*,*)  TARG

C
C     Determine the input epoch.
C
      WRITE (*,*)
      WRITE (*,*) 'Enter SCLK string (blank line to quit):'
      READ (*,FMT='(A)') SCLKCH

      DO WHILE ( SCLKCH .NE. ' ' )

C
C        Convert the input clock string to ticks.
C
         CALL SCENCD ( SC, SCLKCH, SCLKDP )
C
C        Determine the time tolerance.
C
         WRITE (*,*) 'Enter the tolerance as a SCLK string'
         READ  (*,FMT='(A)') TOLCH
C
C        Convert the tolerance to ticks.
C
         CALL SCTIKS ( SC, TOLCH, TOL )
C
C        Search the CK file for pointing data at the time SCLKDP.
C
         CALL CKGP ( INST, SCLKDP, TOL, REF, CMAT, CLKOUT, FOUND )

         IF ( .NOT. FOUND ) THEN
            WRITE (*,*)
            WRITE (*,*) 'The C-kernel file does not contain ' //
     .                  'data for that time.'
            STOP
         END IF

C
C        Compute the inertial pointing vector for the instrument
```

```
C        boresight.
C
C        The C-matrix is a transformation from inertial to
C        instrument-fixed coordinates. The transpose rotates
C        the other way --- what we want.
C
         CALL MTXV ( CMAT,  VPNT, VPNT )


C
C        For all other computations, use the ET time corresponding
C        to the input SCLK.
C
         CALL SCT2E ( SC, SCLKDP, ET )
C
C        Compute the target state vector (position and velocity).
C
         CALL SPKEZ ( TARG, ET, REF, 'LT+S', SC, VTARG, TAU )

C
C        Get TIBF matrix and radii of target ellipsoid model.
C
C        We need TIBF for the target as it appeared when the
C        instrument took its measurement at time ET. The target
C        was at its apparent location TAU seconds earlier.
C
C        BODMAT and BODVAR will read constants from the planetary
C        constants kernel file.
C
         CALL BODMAT ( TARG,  ET-TAU, TIBF )
         CALL BODVAR ( TARG, 'RADII', N, R )

C
C        The position of the observer is just the negative of the
C        position part of the spacecraft-target vector, VTARG.
C        Note that this is NOT the same as the apparent position of
C        the spacecraft as seen from the target.
C
         CALL VMINUS ( VTARG, VPOS )

C
C        Put both vectors in body-fixed coordinates.
C
         CALL MXV ( TIBF,  VPOS,  VPOS )
         CALL MXV ( TIBF,  VPNT,  VPNT )

C
C        Compute the point of intersection, if any.
C

         CALL SURFPT ( VPOS, VPNT, R(1), R(2), R(3), VSURF, FOUND )

         IF ( .NOT. FOUND ) THEN
            WRITE (*,*)
            WRITE (*,*) 'The line-of-sight pointing vector '   //
     .                  'does not intersect the target '
            WRITE (*,*) 'at this epoch.'
            ELSE

C
C              Convert intersection point from rectangular to lat-lon-
C              radius coordinates.
C
               CALL RECLAT ( VSURF, RADIUS, LONG, LAT )

               WRITE (*,*)
               WRITE (*,*) 'Radius:    ', RADIUS
               WRITE (*,*) 'Longitude: ', LONG
               WRITE (*,*) 'Latitude:  ', LAT

            END IF

C
C           Input next epoch.
C
            WRITE (*,*)
            WRITE (*,*) 'Enter SCLK string (blank line to quit):'
            READ (*,FMT='(A)') SCLKCH

         END DO

         END
```

Appendix C --- An Example of Writing a Type 1 CK Segment

The following example shows how one might write a program to create or add to a C-kernel file.

The program creates a single type 1 segment for the scan platform of the Galileo spacecraft. Assume that C-matrices, angular velocity vectors, and the associated SCLK time strings are contained in time-ordered arrays assumed to have been initialized elsewhere (by the subroutine GET_GLL_PNT --- not part of SPICELIB). The program provides the option of adding the segment to an existing file, or creating a new file.

```
         PROGRAM WRTCK1
         IMPLICIT NONE


         INTEGER              FILEN
```

```
      PARAMETER           ( FILEN  = 128 )

      INTEGER               TIMLEN
      PARAMETER           ( TIMLEN =  30 )

      INTEGER               SIDLEN
      PARAMETER           ( SIDLEN =  40 )

      INTEGER               FRMLEN
      PARAMETER           ( FRMLEN =  20 )

      INTEGER               MAXREC
      PARAMETER           ( MAXREC =      10000 )


      DOUBLE PRECISION      CMATS   ( 3, 3, MAXREC )
      DOUBLE PRECISION      QUATS   ( 4,    MAXREC )
      DOUBLE PRECISION      AVVS    ( 3,    MAXREC )
      DOUBLE PRECISION      SCLKDP  (       MAXREC )
      DOUBLE PRECISION      BEGTIM
      DOUBLE PRECISION      ENDTIM

      CHARACTER*(TIMLEN)    SCLKCH  (       MAXREC )
      CHARACTER*(SIDLEN)    SEGID
      CHARACTER*(FILEN)     FILE
      CHARACTER*(1)         ANSWR
      CHARACTER*(FRMLEN)    REF

      INTEGER               INST
      INTEGER               NPREC
      INTEGER               HANDLE

      LOGICAL               AVFLAG

C
C     Can either add to an existing CK file or create a brand
C     new one.
C
      WRITE (*,*)
      WRITE (*,*) 'You may either add to an existing CK file, or'//
     .            ' create a new one.'
      WRITE (*,*) 'Enter the name of the file:'
      READ (*,FMT='(A)') FILE

      WRITE (*,*)
      WRITE (*,*) 'Is this an existing or new file? (Type E or N):'
      READ (*,FMT='(A)') ANSWR

C
C     To convert SCLK times from clock string to encoded SCLK,
C     we need to load the Galileo spacecraft clock kernel file into
C     the kernel pool.  Assume that the file is called GLL_SCLK.TSC
C
      CALL LDPOOL ( 'GLL_SCLK.TSC' )

C
C     To open a new file use CKOPN, and for an existing file use
```

```
C     DAFOPW.
C
C     For a new file, set the internal file name ( 2nd argument in
C     CKOPN ) equal to the file name.
C
      IF ( ANSWR .EQ. 'N' ) THEN

         CALL CKOPN ( FILE, FILE, 0, HANDLE )

      ELSE IF ( ANSWR .EQ. 'E' ) THEN

         CALL DAFOPW ( FILE, HANDLE )

      END IF

C
C     Get the pointing information to go in the C-kernel segment.
C
C        1) Number of pointing instances returned
C        2) Array of SCLK times
C        3) Array of C-matrices
C        4) Array of angular velocity vectors
C
      CALL GET_GLL_PNT ( NPREC, SCLKCH, CMATS, AVVS )

C
C     Enter the information to go in the segment descriptor.
C
C     The NAIF instrument ID code for the Galileo scan platform
C     is -77001.
C
      INST = -77001

C
C     The inertial reference frame is B1950.
C
      REF = 'B1950'

C
C     This segment will contain angular velocity.
C
      AVFLAG = .TRUE.

C
C     The segment identifier provides a 40 character label for
C     the segment.
C
      SEGID = 'GLL SCAN PLT - NAIF - 18-NOV-90'

C
C     Now convert the times to encoded SCLK.
C
      DO I = 1, NPREC
         CALL SCENCD ( -77, SCLKCH(I), SCLKDP(I) )
      END DO

C
```

```
C     Set the segment boundaries equal to the first and last
C     time in the segment.
C
      BEGTIM = SCLKDP(    1)
      ENDTIM = SCLKDP(NPREC)

C
C     The C-matrices are represented by quaternions in a type 1 CK
C     segment.  The SPICELIB routine M2Q converts C-matrices to
C     quaternions.
C
      DO I = 1, NPREC
         CALL M2Q ( CMATS(1,1,I), QUATS(1,I) )
      END DO

C
C     That is all the information that we need. Write the segment.
C
      CALL CKW01 ( HANDLE, BEGTIM, ENDTIM, INST,  REF, AVFLAG,
     .             SEGID,  NPREC,  SCLKDP, QUATS, AVVS        )

C
C     Close the file.
C
      CALL CKCLS ( HANDLE )

      END
```

Appendix D --- An Example of Writing a Type 2 CK Segment

This example program creates a single type 2 segment of predict pointing for the scan platform of the Galileo spacecraft.

This program will use data type 2 to store pointing information for time intervals during which the pointing of the scan platform is constant. It is assumed that a routine called GLL_CONST_PNT will provide ordered arrays of C-matrices and interval start and stop times. The Ith C-matrix represents the fixed platform pointing during the Ith interval. Assume that the start and stop times are given in Galileo clock string form so that they must be converted into encoded SCLK for use in the C-kernel.

```
      PROGRAM WRTCK2
      IMPLICIT NONE
      INTEGER               FILEN
      PARAMETER           ( FILEN  = 128 )

      INTEGER               TIMLEN
      PARAMETER           ( TIMLEN =  30 )

      INTEGER               SIDLEN
      PARAMETER           ( SIDLEN =  40 )

      INTEGER               FRMLEN
      PARAMETER           ( FRMLEN =  20 )

      INTEGER               MAXREC
      PARAMETER           ( MAXREC =      10000 )


      DOUBLE PRECISION      CMATS   ( 3, 3, MAXREC )
      DOUBLE PRECISION      QUATS   (    4, MAXREC )
      DOUBLE PRECISION      AVVS    (    3, MAXREC )
      DOUBLE PRECISION      START   (       MAXREC )
      DOUBLE PRECISION      STOP    (       MAXREC )
      DOUBLE PRECISION      RATES   (       MAXREC )
      DOUBLE PRECISION      BEGTIM
      DOUBLE PRECISION      ENDTIM
      DOUBLE PRECISION      SECTIK

      CHARACTER*(SIDLEN)    SEGID
      CHARACTER*(TIMLEN)    BEGCH   (       MAXREC )
      CHARACTER*(TIMLEN)    ENDCH   (       MAXREC )
      CHARACTER*(FILEN)     FILE
      CHARACTER*(1)         ANSWR
      CHARACTER*(FRMLEN)    REF

      INTEGER               INST
      INTEGER               NPREC
      INTEGER               HANDLE

C
C     Can either add to an existing CK file or create a brand
C     new one.
C
      WRITE (*,*)
      WRITE (*,*) 'You may either add to an existing CK file, or'//
     .            ' create a new one.'
      WRITE (*,*) 'Enter the name of the file:'
      READ (*,FMT='(A)') FILE

      WRITE (*,*)
      WRITE (*,*) 'Is this an existing or new file? (Type E or N):'
      READ (*,FMT='(A)') ANSWR


C
C      es from clock strings to encoded SCLK,
C     we need to load the Galileo spacecraft clock kernel file into
C     the kernel pool.  Assume that the file is called GLL_SCLK.TSC
```

```
C
      CALL LDPOOL ( 'GLL_SCLK.TSC' )

C
C     To open a new file use CKOPN, and for an existing file use
C     DAFOPW.
C
C     For a new file, set the internal file name ( 2nd argument in
C     CKOPN ) equal to the file name.
C
      IF ( ANSWR .EQ. 'N' ) THEN

         CALL CKOPN ( FILE, FILE, 0, HANDLE )

      ELSE IF ( ANSWR .EQ. 'E' ) THEN

         CALL DAFOPW ( FILE, HANDLE )

      END IF

C
C     Get the pointing information to go in the C-kernel segment.
C
C        1) Number of pointing intervals returned
C        2) Interval start times in clock string form
C        3) Interval stop times in clock string form
C        4) Array of C-matrices
C
      CALL GLL_CONST_PNT ( NPREC, BEGCH, ENDCH, CMATS )

C
C     Need to convert the times to encoded SCLK.
C
      DO I = 1, NPREC
         CALL SCENCD ( -77, BEGCH(I), START(I) )
         CALL SCENCD ( -77, ENDCH(I), STOP (I) )
      END DO

C
C     Determine the information to go in the segment descriptor.
C
C     The NAIF instrument ID code for the Galileo scan platform
C     is -77001.
C
      INST = -77001

C
C     The inertial reference frame is B1950.
C
      REF = 'B1950'

C
C     Set the segment boundaries equal to the START time of the
C     first interval and the STOP time of the last interval.
C
      BEGTIM = START(    1)
      ENDTIM = STOP (NPREC)
```

```
C
C     The segment identifier provides a 40 character label for the
C     segment.
C
      SEGID = 'GLL SCAN PLT - NAIF - TYPE 2 PREDICT '

C
C     The C-matrices are represented by quaternions in a type 2 CK
C     segment.  The SPICELIB routine M2Q converts C-matrices to
C     quaternions.
C
      DO I = 1, NPREC
         CALL M2Q ( CMATS(1,1,I), QUATS(1,I) )
      END DO

C
C     Since the pointing is constant over each interval the angular
C     velocity vector is always zero.
C
      DO I = 1, NPREC
         CALL CLEARD ( 3, AVVS(1,I) )
      END DO

C
C     Since this is a predict segment the number of seconds
C     represented by one tick during each of the intervals will
C     be set equal to the nominal amount of time represented by
C     the least significant field of the Galileo clock: 1/120 sec.
C
      SECTIK = 1.D0 / 120.D0

      DO I = 1, NPREC
         RATES(I) = SECTIK
      END DO

C
C     That is all the information that we need. Write the segment.
C
      CALL CKW02 ( HANDLE, BEGTIM, ENDTIM, INST,  REF,  SEGID,
     .             NPREC,  START,  STOP,   QUATS, AVVS, RATES  )

C
C     Close the file.
C
      CALL CKCLS ( HANDLE )

      END
```

Appendix E --- An Example of Writing a Type 3 CK Segment

The following example program shows how one might write a type 3 C-kernel segment to a new file.

The program creates a single type 3 segment for a two hour time period for the Mars Global Surveyor spacecraft bus. The program calculates the pointing instances directly from the spacecraft and planet (SPK) ephemeris file.

The names of the input ephemeris, leapseconds, spacecraft clock, and planetary constants kernel files are fictitious.

```
      PROGRAM MGS_TYPE03
      IMPLICIT NONE

C
C     This program creates a predict type 3 CK segment for the
C     Mars Global Surveyor spacecraft when it is in orbit around
C     Mars.
C

C
C     Assign the NAIF body id codes for the Mars Global Surveyor
C     spacecraft and Mars.
C
      INTEGER               MGS
      PARAMETER           ( MGS  = -94 )

      INTEGER               MARS
      PARAMETER           ( MARS = 499 )

C
C     The reference frame of the segment is J2000.
C
      CHARACTER*(10)        REF
      PARAMETER           ( REF = 'J2000' )

C
C     We will need about 2000 pointing instances.
C
      INTEGER               MAXREC
      PARAMETER           ( MAXREC = 2000 )

C
C     Variables
C
      CHARACTER*(30)        UTCBEG
      CHARACTER*(30)        UTCEND
      CHARACTER*(60)        CKFILE
      CHARACTER*(60)        INFNAM
      CHARACTER*(40)        SEGID
      CHARACTER*(5)         CONT

      DOUBLE PRECISION      ETBEG
      DOUBLE PRECISION      ETEND
      DOUBLE PRECISION      EPOCH
      DOUBLE PRECISION      BEGTIM
      DOUBLE PRECISION      ENDTIM
      DOUBLE PRECISION      SCBEG
      DOUBLE PRECISION      SCEND
      DOUBLE PRECISION      SCLK
      DOUBLE PRECISION      CMAT    ( 3, 3      )
      DOUBLE PRECISION      DCMAT   ( 3, 3      )
      DOUBLE PRECISION      OMEGA   ( 3, 3      )
      DOUBLE PRECISION      SCLKDP  (    MAXREC )
      DOUBLE PRECISION      QUAT    ( 4, MAXREC )
      DOUBLE PRECISION      AV      ( 3, MAXREC )
      DOUBLE PRECISION      START   (    MAXREC )

      INTEGER               HANDLE
      INTEGER               NREC
      INTEGER               NINT
      INTEGER               INST
      INTEGER               I

      LOGICAL               AVFLAG

C
C     Load the binary SPK file that provides states for MGS with
C     respect to Mars for the time period of interest.
C
      CALL SPKLEF ( 'naf0000c.bsp', HANDLE )
C
C     Load the text leapseconds, spacecraft clock ( sclk ), and
C     planetary constants ( pck ) files into the kernel pool.
C
      CALL LDPOOL ( 'leap.tls' )

      CALL LDPOOL ( 'mgs.sc'   )

      CALL LDPOOL ( 'mgs.pck'  )
C
C     The segment begin and end times.
C
      UTCBEG = '1994 JAN 21 00:00:00'
      UTCEND = '1994 JAN 21 02:00:00'

      CALL UTC2ET ( UTCBEG, ETBEG )
      CALL UTC2ET ( UTCEND, ETEND )

      CALL SCE2C  ( MGS, ETBEG, SCBEG )
      CALL SCE2C  ( MGS, ETEND, SCEND )

C
C     Calculate the quaternions and angular velocity vectors at
C     roughly four second intervals from the segment start time
```

```
C     until the end.
C
      I = 1

      SCLK = SCBEG

      DO WHILE ( ( SCLK .LE. SCEND ) .AND. ( I .LE. MAXREC ) )
C
C        The times stored in the C-kernel are always in encoded
C        spacecraft clock form.  SPK takes ET as the input time.
C
         SCLKDP(I) = SCLK

         CALL SCT2E ( MGS, SCLK, EPOCH )

C
C        Find the C-matrix using the MGSSPICE routine LOCVRT_M.
C        LOCVRT_M returns the 3x3 matrix that transforms vectors
C        from a specified inertial reference frame to the `Local
C        Vertical Frame' for a specified observer and target body.
C        For Mars Global Surveyor, this frame is also known as the
C        "A-frame" and the "Orbital Reference Coordinate System".
C
         CALL LOCVRT_M ( MARS, MGS, EPOCH, REF, 'NONE', CMAT )

         CALL M2Q ( CMAT, QUAT(1,I) )
C
C        Calculate the angular velocity vector using the following
C        formula:
C
C        Let the angular velocity vector be  AV = ( a1, a2, a3 )
C        and let the matrix OMEGA be:
C
C                    +--              --+
C                    |   0   -a3   a2   |
C                    |                  |
C           OMEGA =  |  a3    0    -a1  |
C                    |                  |
C                    |  -a2   a1    0   |
C                    +--              --+
C
C         Then the derivative of a C-matrix C is given by
C
C                                 t
C                     t       d [ C ]
C            OMEGA * C   =    -------
C                               dt
C
C         Thus, given a C-matrix and its derivative, the angular
C         velocity can be calculated from
C
C                          t
C                        dC
C             OMEGA  =  { -- }  *  C
C                        dt
C
C
```

```
C
C        GET_DERVRT is a non SPICELIB routine that will calculate
C        the derivative of the C-matrix calculated by LOCVRT_M.
C
         CALL GET_DERVRT ( EPOCH, DCMAT )

         CALL MTXM ( DCMAT, CMAT, OMEGA )

         AV(1,I)  = OMEGA (3,2)
         AV(2,I)  = OMEGA (1,3)
         AV(3,I)  = OMEGA (2,1)
C
C        Increase the counter and encoded SCLK time for the next
C        pointing instance.
C
         I = I + 1

         SCLK = SCLK + 1024.D0

      END DO

      NREC = I - 1

C
C     Unload the SPK file.
C
      CALL SPKUEF ( HANDLE )

C
C     The process of determining how to partition the pointing
C     instances into interpolation intervals varies with respect
C     to the means by which the pointing instances are obtained.
C
C     For this example program it is acceptable to interpolate
C     between all of the adjacent pointing instances because:
C
C     1) The pointing was calculated at every 4 seconds so there
C        are no gaps in the data.
C
C     2) The pointing was calculated directly from the spacecraft
C        and planetary ephemeris so that the functions for the
C        spacecraft axis and angular velocity vectors will change
C        "slowly" and continuously.
C
C     Therefore there is only one interpolation interval for the
C     entire segment.
C
      NINT        = 1

      START ( 1 ) = SCLKDP (1)


C
C     Now that the pointing instances have been calculated the
C     segment can be written to a C-kernel file.
C
C     Open a new file.
```

```
C
      CKFILE   = 'mgs_predict_ck.bc'

      INFNAM   = 'mgs_predict_ck.bc'

      CALL DAFONW ( CKFILE, 'CK', 2, 6, INFNAM, 0, HANDLE )

C
C     Set the values of the components of the segment descriptor.
C
C     The NAIF id code for the MGS spacecraft bus is:
C
      INST     = -94000
C
C     This segment contains angular velocity data.
C
      AVFLAG   = .TRUE.
C
C     The segment begins and ends with the first and last
C     pointing instances.
C
      BEGTIM   = SCLKDP ( 1    )
      ENDTIM   = SCLKDP ( NREC )

C
C     The reference frame was specified above as J2000.
C
C     The segment identifier is:
C
      SEGID    = 'MGS PREDICT TYPE 3 SEGMENT'

C
C     Write the segment to the file attached to HANDLE.
C
      CALL CKW03 ( HANDLE, BEGTIM, ENDTIM, INST, REF, AVFLAG,
     .             SEGID,  NREC,   SCLKDP, QUAT, AV,  NINT,
     .             START                                     )

C
C     Close the file.
C
      CALL DAFCLS ( HANDLE )

      END
```