SCLK REQUIRED READING

Navigation Ancillary Information Facility

Prepared by
N. J. Bachman

NAIF Document No. 222.02
Last Modified: 06 OCT 1999

Table of Contents

SCLK

Revisions

April 12, 1999

The document differs from the previous version of April 20, 1992 in that it documents the new capability of the SCLK software to convert between ET and continuous ticks. Examples involving Mars Observer have been updated to refer to Mars Global Surveyor. The quotation style has been changed from British to American. The program example showing use of the SCLK system together with the CK reader CKGP has been corrected. Miscellaneous minor changes of wording have been made throughout the text.

Introduction

The spacecraft clock is the onboard time-keeping mechanism that triggers most spacecraft events, such as shuttering of a camera. Since telemetry data are down-linked with this clock's time attached to it, spacecraft clock time (SCLK–pronounced "s-clock") is the fundamental time measurement for referencing many spacecraft activities.

It is natural, then, that SCLK have an important role in the SPICE system. In fact, all C-kernel pointing data are referenced to SCLK. SPICELIB contains routines to convert between SCLK and other standard time systems, such as Ephemeris Time (ET) and Universal Time Coordinated (UTC).

References

1. SFOC SIS "SFOC-2-SYS-Any-TimeForms," 02/06/90.

Supported Missions

The set of missions for which NAIF currently supports SCLK time conversions includes, but is not limited to, the following:

- Cassini
- Galileo
- Mars Climate Orbiter
- Mars Global Surveyor
- Mars Polar Lander
- Near Earth Asteroid Rendezvous
- Stardust
- Voyager I and II

The suite of SCLK routines has been designed to easily accommodate future missions. A later section describes how the system might be easily expanded to incorporate new spacecraft clocks.

The Basics

In this section, we present a minimal subset of facts about the SPICE SCLK system that you can get by with and still use the system successfully.

SCLK rates

Most of the complexity of dealing with SCLK time values arises from the fact that the rate at which any spacecraft clock runs varies over time. As a consequence, the relationship between SCLK and ET or UTC is not accurately described by a linear function; usually, a piecewise linear function is used to model this relationship.

The mapping that models the relationship between SCLK and other time systems is updated as a mission progresses. While the change in the relationship between SCLK and other systems will usually be small, you should be aware that it exists; it may be a cause of discrepancies between results produced by different sets of software.

SCLK kernels

To use any of the SCLK conversion routines, your program must first load a SCLK kernel file. The code fragment

```
CALL LDPOOL ( <name of the SCLK kernel file goes here> )
```

accomplishes this. You must supply the actual name of the kernel file you want to load.

In addition, you will usually need to load a leapseconds kernel. For some missions, conversions between SCLK and ET will require that both an SCLK and a leapseconds kernel be loaded. The code fragment

```
CALL LDPOOL ( <name of the LEAPSECONDS kernel file goes here> )
```

loads a leapseconds kernel. Leapseconds kernels are described in the TIME required reading document.

Normally, you will load these kernels at just one point in your application program, prior to using any time conversion routines.

Details concerning the kernel pool are covered in the KERNEL required reading document.

Partitions, briefly

The lifetime of each mission is divided into intervals called "partitions." Partitions are time intervals during which the spacecraft clock advances continuously. Every time that a discontinuity in a spacecraft clock's readout values occurs, a new partition is started. Discontinuities may consist of positive jumps, in which the spacecraft clock's readout "skips" ahead, or negative jumps, in which the spacecraft clock regresses.

The fact that a spacecraft clock may regress raises the possibility that the clock may give the same reading at two or more different times. For this reason, SCLK strings in SPICELIB are prefaced with partition numbers.

The partition number is a positive integer followed by a forward slash, for example

```
4 /
```

Any number of blanks are allowed on either side of the slash.

An example of a Galileo SCLK string with a partition number is

```
1/100007:76:1
```

Partition numbers serve to ensure that spacecraft clock readings can be interpreted unambiguously.

Converting between SCLK strings and ET or UTC

The time known as "spacecraft event time" (SCET) is usually UTC. You must verify that this is the case for your spacecraft.

To convert a SCLK string to a double precision ET value, you can use the subroutine call

```
CALL SCS2E  ( SC, CLKSTR, ET  )
```

To convert a SCLK string to a UTC string, you can use the code fragment

```
CALL SCS2E  ( SC, CLKSTR, ET  )
CALL TIMOUT ( ET, PICTUR, UTC )
```

where

`SC`	is the NAIF spacecraft ID code for your spacecraft.
`CLKSTR`	is a SCLK string.
`ET`	is an ET time.
`PICTUR`	is a format picture for TIMOUT.
`UTC`	is the UTC time equivalent to SCLK.

See these routines for details concerning their arguments.

The inverse conversion is performed by the code fragment

```
CALL STR2ET ( UTC, ET          )
CALL SCE2S  ( SC,  ET,  CLKSTR )
```

Using encoded SCLK

The SPICELIB C kernel (CK) system tags CK data with SCLK times. Within the CK system, these time tags are encoded as double precision numbers. To look up CK data, you will need to supply encoded SCLK time tags to the CK reader routines.

You can obtain encoded SCLK values from SCLK strings via the routine SCENCD. The code fragment

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

encodes the SCLK string CLKSTR as the double precision value SCLKDP.

Encoded SCLK values can be converted to strings using the code fragment

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

You can obtain continuous encoded SCLK values from ET via the routine SCE2C. The code fragment

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

encodes the ephemeris time ET as the double precision value SCLKDP. SCLKDP need not be integral; even though non-integral tick values do not represent SCLK readings, they are permitted to avoid truncation error when representing ET as encoded SCLK.

A parallel routine SCE2T converts ET to encoded SCLK, rounding the result to the nearest integral tick.

The inverse conversion is provided by the routine SCT2E, which is called as follows:

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

SCT2E handles integral or continuous tick values as inputs.

There is a special routine that is used for encoding "tolerance" values for the CK readers. (See the CK Required Reading document for a discussion of the CK readers.)

The code fragment

```
CALL SCTIKS ( SC, CLKSTR, TICKS )
```

produces an encoded tolerance value. SCTIKS takes SCLK strings WITHOUT partition numbers as inputs; this is because the strings indicate a delta time rather than an absolute time.

All of the concepts used in this section are discussed in greater detail in the following sections of this document.

Encoded SCLK

The fundamental representation of SCLK in the SPICE system is a double precision numeric encoding of each multi-component count. Encoding SCLK provides the following advantages:

- Encoding makes for a more compact representation. Direct (unencoded) representation of spacecraft clock counts usually requires multiple numbers for the separate components making up the SCLK count.
- Having a single numeric equivalent for each count makes it much easier to compare SCLK times (Is time t1 greater than time t2? Is time t1 closer to time t2 than time t3? And so on.)

For these reasons, encoded SCLK is the time representation that is associated with pointing data in the C-kernel. Encoded SCLK is the basis by which conversions are made from SCLK to other time systems.

To convert a character representation of an SCLK count SCLKCH to its double precision encoding SCLKDP, use the routine SCENCD (Encode SCLK):

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

The routine SCDECD (Decode SCLK) recovers the character representation of spacecraft clock 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, –32 for Voyager 2). Each spacecraft may have a different format for its clock counts, so the encoding scheme may be different for each.

Later chapters describing clock types give complete details on clock string formats for spacecraft clocks supported by the SPICE Toolkit.

Ticks

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

An analogy can be drawn with a standard wall clock, showing hours, minutes, and seconds. One tick for a wall clock would be one second. And a wall clock time of

```
10:05:50
```

would represent

```
10(3600) + 5(60) + 50 = 36350
```

ticks.

As in the case of the wall clock, the length of time associated with a tick varies as the clock rate varies.

Since not all spacecraft clocks are the same, the particular time value for one tick varies from spacecraft to spacecraft. For Mars Global Surveyor, for instance, one tick is equivalent to approximately four milliseconds. For Galileo, it's about 8 1/3 milliseconds.

In addition to representing spacecraft clock readings, ticks can be used to represent arbitrary epochs. In order to minimize discretization error, "continuous" (non-integral) tick values are supported: ephemeris times may be converted to non-integral ticks via the routine SCE2C.

Conversion of spacecraft clock strings to ticks always produces integral tick values.

Partitions

One desirable feature of encoded SCLK is that it increases continuously throughout the course of the mission. Unfortunately, real spacecraft clocks do not always behave so nicely. A clock may reset to a lower value, rendering certain counts ambiguous. This might happen if the clock has reached its maximum expression, or because of a power surge. A clock may also jump ahead.

Any time one of these discontinuities occurs, we say that SCLK time has entered a new partition. The partitions must be accounted for when encoding and decoding SCLK.

To continue our analogy, say our wall clock was being used to keep time throughout an entire day. Then 10:05:50 is ambiguous, because we don't know if it falls in the morning or evening "partition." So we append the indicators "a.m." or "p.m." to be clear.

We handle SCLK similarly. Instead of just converting a clock count to ticks (10:05:50 to 36350), we take into account the partition that the count falls in, and compute the number of ticks since clock start (10:05:50 a.m. to 36350; 10:05:50 p.m. to 36350 + 12(60)(60) = 79550).

When you pass a SCLK string to SCENCD, it is normally prefixed with a number indicating the partition in which the count falls. Sample SCLK strings for Voyager 2, including partition numbers, are given in an example program later in this document.

The presence of the partition number is not always required. If it is missing, SCENCD will assume the partition to be the earliest one possible that contains the clock string being encoded. It's good practice to always include the partition number in SCLK strings.

To convert to ticks since clock start, SCENCD processes the partition number. It has to know how many ticks were in all preceding partitions, and what the start and stop clock values were for each. This information is stored in a SCLK kernel file for that spacecraft. The SCLK kernel file is described in detail in a later section.

New partitions may occur at any time throughout the course of active missions. The responsible mission operations team must update the SCLK kernel file to include new partitions as they occur.

In converting encoded SCLK back to an equivalent clock string, SCDECD must also use the SCLK kernel file. Note, however, that you only have to load the SCLK kernel file once in your program, no matter how many calls to SCENCD and SCDECD are made afterwards. See the KERNEL required reading file for information about "loading" miscellaneous kernel files into the kernel pool.

SCDECD always returns a clock string prefixed by a partition number and the '/' character, for example

```
2/2000:83:12
```

If you want to read partition start and stop times for yourself, use the routine SCPART:

```
CALL SCPART ( SC, NPARTS, PSTART, PSTOP )
```

SCLK Conversion Routines

In order to correlate data obtained from different components of the SPICE system, for example pointing and ephemeris data, it is necessary to be able to convert between SCLK time and representations of time in other systems, such as UTC and ephemeris time (also referred to as "ET," "barycentric dynamical time," and "TDB").

SPICELIB contains the following subroutines to convert between encoded and character SCLK, ET and UTC. Note that the names of the the routines involving SCLK are all prefixed with SC, for Spacecraft Clock.

`ET2UTC`	`(ET, FORMAT, PREC, UTC)`	(Convert ET to a UTC string)
`UTC2ET`	`(UTC, ET)`	(Convert a UTC string to ET)
`SCENCD`	`(SC, SCLKCH, SCLKDP)`	(Encode SCLK)
`SCDECD`	`(SC, SCLKDP, SCLKCH)`	(Decode SCLK)
`SCT2E`	`(SC, SCLKDP, ET)`	(Convert encoded SCLK ticks to ET)
`SCS2E`	`(SC, SCLKCH, ET)`	(Convert SCLK string to ET)
`SCE2C`	`(SC, ET, SCLKDP)`	(Convert ET to continuous ticks)
`SCE2T`	`(SC, ET, SCLKDP)`	(Convert ET to encoded SCLK ticks)
`SCE2S`	`(SC, ET, SCLKCH)`	(Convert ET to SCLK string)

It takes at most two subroutine calls to convert between any two of the four representations.

SPICELIB also contains two routines that can encode and decode relative, or "delta" SCLK times. These are SCLK strings without partition numbers that represent time increments rather than total time since clock start. Such strings are encoded as tick counts. The routines are:

`SCTIKS`	`( SC, CLKSTR, TICKS )`	(Convert delta SCLK to ticks)
`SCFMT`	`( SC, TICKS, CLKSTR)`	(Convert ticks to delta SCLK)

Distinguishing Between Different Clocks

The algorithms used to encode and decode SCLK, and convert between SCLK and other time systems are are not necessarily the same for each spacecraft.

The differences are handled by the SCLK software at two levels: High level differences are managed in the code itself through "clock types." More detailed spacecraft-specific differences are handled using parameters in a SCLK kernel.

Clock Types

A clock type is a general clock description which may encompass several separate spacecraft clocks. Each clock type is identified in the SCLK subroutines by an integer code. At the release date of the current revision of this document, all supported missions use spacecraft clock type 1.

A spacecraft clock data type has two components: a format defining the set of acceptable spacecraft clock (SCLK) strings, and a method of converting SCLK strings to a standard time representation, such as ephemeris or UTC seconds past J2000.

For example, a type 1 clock consists of some number of cascading integer counters. An individual counter can increment only when the immediately preceding counter reaches its maximum expression and "rolls over." Our wall clock is an example: the counters are hours, minutes and seconds. One tick for a type 1 clock is defined to be the value of the least-significant component increment. Clock type 1 uses a piecewise-linear interpolation process to convert between SCLK and other time systems.

The chapter "SLCK01" describes clock type 1 in detail. It includes the specific SCLK string formats for each of the type 1 spacecraft clocks supported by the SPICE Toolkit.

SCLK routines determine the clock type for a particular spacecraft from the SCLK kernel file (described in the next section).

Clock type-specific routines

Each clock type is supported in the encoding and decoding process by the routine SCcc, where cc is the number of the clock type. SCcc contains two entry points:

`SCTKcc`	`(SC, CLKSTR, TICKS)`	(SCLK string to ticks, type cc)
`SCFMcc`	`(SC, TICKS, CLKSTR)`	(Ticks to SCLK string, type cc)

SCTKcc and SCFMcc do not process any partition information; that work is handled at a higher level by SCENCD and SCDECD, and is the same for all spacecraft clocks.

SCTKcc and SCFMcc are called by SCTIKS and SCFMT, respectively.

Each clock type is supported in the time conversion process by two routines:

`SCTEcc`	`(SC, SCLKDP, ET)`	(Encoded SCLK ticks to ET, type cc)
`SCECcc`	`(SC, ET, SCLKDP)`	(ET to continuous ticks, type cc)

Spacecraft-Specific Parameters

Once the clock type has been determined, SCLK routines need parameters that uniquely distinguish each spacecraft within the same SCLK type. For instance, for type 1, they need to know: How many components make up this particular clock? What are the modulus values for each of the components? What are the coefficients defining the mapping from SCLK to a “parallel” time system, such as ET? Spacecraft-specific parameters such as these are read from the SCLK kernel file at run-time (see below).

The SCLK Kernel File

NAIF SCLK kernel files supply SPICELIB SCLK conversion routines with information required to convert between SCLK values and other representations of time. Typically, a NAIF SCLK kernel will describe the clock of a single spacecraft.

Before calling any of the routines to encode or decode SCLK, or convert between SCLK and other time systems, an application program must load the contents of the SCLK kernel file into the kernel pool, using the subroutine LDPOOL (load pool):

```
CALL LDPOOL ( 'name_of_SCLK_kernel_file' )
```

An application must also load the leapseconds kernel file if there are any conversions to be performed between ET and UTC. This is typically done in the initialization section of your program.

The SCLK kernel file you use should contain values for the particular spacecraft you are dealing with. The variables expected to be found in the file are all prefixed with the string

```
SCLK_
```

These variables include partition boundaries, clock type, and several other parameters associated with the clock type. These are described below.

Partition boundaries

The tick values for the beginning and end of each partition are given by:

```
SCLK_PARTITION_START_ss = ( .....
                            .....
                            .....
                            ..... )

SCLK_PARTITION_END_ss   = ( .....
                            .....
                            .....
                            ..... )
```

where –ss is the spacecraft ID code. These variables are arrays containing one element per partition. The nth element of

```
SCLK_PARTITITION_END_ss
```

is considered to be the "first tick" of the (n+1)st partition. Mathematically speaking, partitions may be thought of as intervals that are closed on the left and open on the right.

Clock type assignment

If –ss is the NAIF ID code of a spacecraft, the associated clock type for that spacecraft is given by the assignment

```
SCLK_DATA_TYPE_ss = ( cc )
```

where cc is the clock type. New clock types will be developed as needed.

Note that multiple spacecraft ID codes can be associated with the type 1 SCLK data type at one time. Since the spacecraft codes are included in the SCLK variable names, there will be no naming conflicts. (We don't expect this feature to be used much, if at all, but it's there should you need it.)

Clock type-specific parameters

Each spacecraft clock type has its own set of parameters that the SPICELIB SCLK routines require in order to convert SCLK values of that type. A complete list and description of these parameters, and their variable names for the kernel pool, is given for type 1 in the chapter "SCLK01."

Expanding the system: What NAIF must do

Accommodating new spacecraft clocks may involve no code changes to the SCLK subroutines whatsoever.

If a new clock fits into the framework of clock type 1, then the clock can be accommodated simply by producing a new kernel file for that spacecraft clock. For the new clock, a new set of kernel variables corresponding to those described above, and those in the chapter "SCLK01," could be added to an existing SCLK kernel file. Alternatively, an entirely new SCLK kernel file containing the new parameters could be created — this is the more likely approach. Once this is done, all existing SCLK routines will work without modfication with the new spacecraft ID.

If a new clock does not fit into the clock type 1 framework, then NAIF will design a new clock type. This will involve writing new versions of the four clock type-specific routines described earlier:

```
SCTKcc
SCFMcc
SCTEcc
SCECcc
```

where cc is the new clock type number.

New cases will have to be added to the code of the following higher-level SCxxx conversion routines to call the new, type-specific routines:

```
SCFMT
SCTIKS
SCT2E
SCS2E
SCE2C
```

```
SCE2T
SCE2S
```

It will probably be necessary to design new SCLK kernel file variables to accommodate the new type, and augment the standard variables described above.

Adding a new clock type does not change the calling sequence of any of the high level conversion routines. Thus, once you've learned how to use the SCLK conversion routines, you won't have to re-learn just because a new spacecraft clock has been introduced.

An Example Using SCLK Routines

The following example shows how some of the SCLK routines might be used in a typical application program. This one reads pointing data from a C-kernel file. In this example, a set of four input clock times are hard-coded in the program for the purpose of demonstration: A real application written by you would likely get input times from some external source, such as a file or through interactive user input.

```
C
C     Request pointing from a C-kernel file for a sequence of
C     pictures obtained from the Voyager 2 narrow angle camera.
C     Use an array of character spacecraft clock counts as input.
C
C     Decode the output clock counts and print the input and
C     output clock strings. Also print the equivalent UTC time
C     for each output clock time.
C
C     Note that the SCLK kernel file must contain VGR 2 clock
C     information.
C
      INTEGER             NPICS
      PARAMETER         ( NPICS = 4 )

      CHARACTER*(80)      CK
      CHARACTER*(80)      SCLKKR
      CHARACTER*(80)      LSK
      CHARACTER*(25)      SCLKIN    ( NPICS )
      CHARACTER*(25)      SCLKOUT
      CHARACTER*(25)      CLKTOL
      CHARACTER*(25)      UTC
      CHARACTER*(25)      REF

      DOUBLE PRECISION    CMAT      ( 3, 3 )
      DOUBLE PRECISION    ET
      DOUBLE PRECISION    TIMEIN
      DOUBLE PRECISION    TIMOUT
      DOUBLE PRECISION    TOL
```

```
      INTEGER             HANDLE
      INTEGER             I
      INTEGER             INST
      INTEGER             SC

      LOGICAL             FOUND

      DATA  CK        / 'VGR2NA.BC'         /
      DATA  LSK       / 'LSK.KER'           /
      DATA  SCLKKR    / 'SCLK.KER'          /

      DATA  SCLKIN    / '2/20538:39:768',
     .                  '2/20543:21:768',
     .                  '2/20550:37',
     .                  '2/20564:19'        /

      DATA  CLKTOL   /  '        0:01:001'  /

C
C     The instrument we want pointing for is the Voyager 2
C     narrow angle camera.  The reference frame we want is
C     J2000. The spacecraft is Voyager 2.
C
      INST = -32001
      REF  = 'J2000'
      SC   = -32

C
C     Load the appropriate files. We need
C
C     1) A CK file containing pointing data.
C     2) The SCLK kernel file, for the SCLK conversion routines.
C     3) A leapseconds kernel, for ET-UTC conversions.
C
      CALL CKLPF  ( CK,    HANDLE )
      CALL LDPOOL ( SCLKKR        )
      CALL LDPOOL ( LSK           )

C
C     Convert the tolerance string to ticks.
C
      CALL SCTIKS ( SC, CLKTOL, TOL )

      DO I = 1, NPICS

         CALL SCENCD ( SC, SCLKIN( I ),  TIMEIN )

         CALL CKGP   ( INST, TIMEIN, TOL, REF, CMAT, TIMOUT,
     .                 FOUND )

         IF ( FOUND ) THEN

            CALL SCDECD ( SC, TIMOUT, SCLKOUT         )
            CALL SCT2E  ( SC, TIMOUT, ET              )
            CALL ET2UTC (             ET, 'D', 3, UTC )

            WRITE (*,*)
            WRITE (*,*) 'Input  s/c clock count: ', SCLKIN (I)
            WRITE (*,*) 'Output s/c clock count: ', SCLKOUT
```

```
         WRITE (*,*) 'Output UTC:                ', UTC
         WRITE (*,*) 'Output C-Matrix:           ', CMAT
         WRITE (*,*)

      ELSE

         WRITE (*,*) 'Input  s/c clock count: ', SCLKIN (I)
         WRITE (*,*) 'No pointing found.'

      END IF

   END DO
   END
```

The output from this program looks like this:

```
Input  s/c clock count:  2 / 20538:39:768
Output s/c clock count:  2/20538.39.768
Output UTC:              79-186/21:50:23.000
Output C-Matrix:  <first C-matrix>

Input  s/c clock count:  2 / 20543:21:768
Output s/c clock count:  2/20543.22.768
Output UTC:              79-187/01:35:57.774
Output C-Matrix:  <second C-matrix>

Input  s/c clock count:  2 / 20550:37
Output s/c clock count:  2/20550.36.768
Output UTC:              79-187/07:23:57.774
Output C-Matrix:  <third C-matrix>

Input  s/c clock count:  2 / 20564:19
Output s/c clock count:  2/20564.19.768
Output UTC:              79-187/18:22:21.774
Output C-Matrix:  <fourth C-matrix>
```

SCLK01

This chapter describes the type 1 SCLK format and conversion algorithms in detail. Also, the SCLK formats for supported spacecraft whose clocks conform to the type 1 specification are described.

Conforming spacecraft clocks

The following spacecraft have SCLK formats that conform to the type 1 specification:

- Cassini
- Galileo Orbiter
- Mars Global Surveyor
- Mars Climate Orbiter
- Mars Polar Lander
- NEAR
- Stardust
- Voyager 1
- Voyager 2

The spacecraft clock encoding and conversion functionality described in this document is fully supported by the SPICE Toolkit for these spacecraft.

Type 1 SCLK format

The first standard NAIF spacecraft clock data type has two components: a format defining the set of acceptable spacecraft clock (SCLK) strings, and a method of converting SCLK strings to any of a set of standard time systems such as UTC or ET.

Type 1 SCLK strings have the form

```
pp/<time string>
```

where pp is a partition number, and

```
<time string>
```

is a time representation that conforms to the type 1 SCLK format. The partition specification (number and slash character) is optional; SCLK strings without partition numbers are assumed to refer to times in the first partition in which the specified clock count occurred. It's good practice to always include the partition number.

An example of a type 1 SCLK string (for Galileo) is

```
3 / 10110007:09:6:1
```

The number "3" is the partition number, the slash is a delimiter, and the rest of the string is a "time string." With this example in hand, we're ready to define the type 1 SCLK format.

The partition number is a positive integer followed by a forward slash, for example

```
4 /
```

Zero or more blanks are allowed on either side of the slash.

A type 1 SCLK time string consists of a series of one or more fields, each of which contains an integer. All fields but the leftmost are optional. The fields of a time string represent modular counts of time units. (A "mod n" count increments from zero to n-1, and then cycles back to zero.) The values for a given field may be offset by some fixed integer, so that they range from m to m+n, where m is non-negative. The moduli of the various fields are not necessarily the same. The time unit associated with a given field, multiplied by the modulus for that field, gives the time unit for next field to the left.

For each field but the first, values may exceed the modulus for the field. For example, the modulus of the fourth field of a Galileo SCLK string is 8, but the digit "9" is allowed in that field. So

```
0:0:0:9
```

is a valid Galileo SCLK string and represents the same time as

```
0:0:1:1
```

On input to SPICELIB routines, the fields of a type 1 SCLK string may be separated by any of the delimiter characters

```
-   .   ,   :  <blank>
```

On output from SPICELIB routines, the delimiter characters will be those defined by a parameter in the SCLK kernel, described later.

Consecutive delimiters containing no intervening digits are treated as if they delimit zero values.

Note that all fields in time strings represent integers, not decimal fractions. So, the strings

```
11000687:9

11000687:90
```

do not represent the same time value: in the former, the second field indicates a count of 9; in the latter, 90.

Galileo SCLK format

An example of a valid time string (without a partition number) for the Galileo spacecraft clock is:

```
16777214:90:9:7
```

Numbering the fields from left to right, the time units and moduli of the fields are:

```
Field             Time unit              Modulus
-----     ---------------------------    --------
  1       60 2/3   sec.                  16777215
  2          2/3   sec.  (666 2/3 ms)          91
  3          1/15  sec.  ( 66 2/3 ms)          10
  4          1/120 sec.  (   8 1/3 ms)          8
```

Fields 1–4 are known as: "Real time image count" (RIM), "mod 91 count," "mod 10 count" or "real time interrupt count" (RTI), and "mod 8 count." The values in all fields normally range from zero to the modulus of the field, minus one.

The maximum time value that the Galileo spacecraft clock can represent (16777214:90:9:7) is approximately 32 years.

Mars Global Surveyor SCLK format

An example of a valid time string (without a partition number) for the Mars Global Surveyor spacecraft clock is:

```
4294967295.255
```

Numbering the fields from left to right, the time units and moduli of the fields are:

Field	Time unit	Modulus
1	approximately 1 sec.	4294967296
2	1/256 sec.	256

Field 1 is known as the "sclk_secs count." Field 2 is known as the "sclk_fine word." The values in the first and second fields normally range from zero to the modulus of the field, minus 1.

The maximum time value that the Mars Global Surveyor spacecraft clock can represent (4294967295.255) is approximately 136 years.

Voyager SCLK clock format

An example of a valid time string (without a partition number) for both the Voyager 1 and Voyager 2 spacecraft clocks is:

```
65535.59.800
```

Numbering the fields from left to right, the time units and moduli of the fields are:

Field	Time unit	Modulus
1	2880 sec.	65536
2	48 sec.	60
3	0.06 sec.	800

Fields 1–3 are known as: "Mod 16 count" (actually mod 2**16), "mod 60 count," and "mod 800 count." The values in the first and second fields normally range from zero to the modulus of the field, minus 1. The range of the third field is from 1 to 800. The "offset" for the third field is 1, so values in this field normally range from 1 to 800 rather than from 0 to 799; values above 800 are allowed and treated as described above.

The maximum time value that the Voyager 1 and Voyager 2 spacecraft clocks can represent (65535:59:800) is approximately six years.

Type 1 SCLK conversion

SPICELIB contains subroutines that convert between type 1 clock strings and the following representations of time:

- ET
- encoded SCLK

The routines that carry out these conversions are described above in the chapter "SCLK Conversion Routines."

Since SPICELIB also contains subroutines that convert between any of a variety of standard time systems, including ET, UTC, Terrestrial Dynamical Time (TDT), TAI, TDB Julian date, TDT Julian Date, and UTC Julian Date, conversion between SCLK strings and any other time system supported by SPICELIB requires at most two subroutine calls.

Conversion algorithms

For every type 1 spacecraft clock, encoded SCLK values are converted to ephemeris time as follows: first, encoded SCLK values are mapped to equivalent time values in a standard time system such as ET or TDT. If the standard time system is not ET, values from this system are mapped to ET.

The standard time system used for the conversion is referred to here and in the SPICELIB SCLK routines as the "parallel" time system. Normally, the SPICE Toolkit will use only one parallel time system for any given spacecraft clock.

Conversion from ET to encoded SCLK follows the reverse path: first, ET values are converted, if necessary, to equivalent values in the parallel time system; next, those parallel time values are converted to encoded SCLK.

For each type 1 spacecraft clock, encoded SCLK is related to the parallel time system for that clock by a piecewise linear function. The function is defined by a set of pairs of encoded SCLK values and corresponding values in the parallel time system, and by a set of "rate" values that apply to the intervals between the pairs of time values. The rate values give the rate at which "parallel time" increases with respect to encoded SCLK time during the interval over which the rate applies.

The specific method by which pairs of time values and rates are used to map encoded SCLK to parallel time values is explained in detail below. In the following discussion, for simplicity, we'll assume that the parallel time system is ET. We'll also assume that the rate units are ephemeris seconds per the number of ticks in the "most significant SCLK count." In general, the rates in a type 1 SCLK kernel will be in units of

```
    parallel time units
---------------------------
most significant clock count
```

For example, for the GLL orbiter spacecraft clock, the rate unit is "ephemeris seconds per RIM."

We can represent the data that define the SCLK-to-ET mapping as a set of ordered triples of encoded SCLK values (in units of ticks since spacecraft clock start), their equivalents in ephemeris time, and the rates corresponding to each pair of times:

```
( s/c_clock(1),  et(1),  rate(1) )
                   .
                   .
                   .
( s/c_clock(n),  et(n),  rate(n) )
```

The mapping of SCLK values to ephemeris times is carried out as follows: If the sclk time clock satisfies

```
sclk(i)  <  clock  <  sclk(i+1)
         -
```

then the corresponding ephemeris time is

```
et(i)    +    rate(i) *  ( clock - sclk(i) )
```

If

```
clock  >  clock(n)
       -
```

the formula still applies, with i = n.

To convert ephemeris time values to SCLK, we use an analogous method. If "time" is the value to be converted, and

```
et(i)  <  time  <  et(i+1)
       -
```

then the corresponding continuous encoded SCLK value is

```
                time -  et(i)
sclk(i)    +    -------------
```

```
                    rate(i)
```

If

```
time >  et(n)
     -
```

the formula still applies, with i = n.

Note that this method will not handle rate values of 0 seconds per tick.

When the function described by the pairs of time values and rates is continuous, then all rates except for the last one are redundant, since

```
                 et(i+1)   -   et(i)
rate(i)  =   ----------------------
              sclk(i+1)   -   sclk(i)
```

If the mapping from encoded SCLK to the parallel time system is not continuous, then the mapping will not be strictly invertible: mapping an encoded SCLK value to a parallel time value, then mapping that parallel time value to encoded SCLK may not yield the original encoded SCLK value. However, the capability of supporting non-continuous mappings is provided in case it is needed to implement the mapping provided by a flight project.

In order for SPICELIB SCLK conversion routines to work, the information represented by the ordered triples described above must be loaded via the kernel pool. See the section "The spacecraft clock kernel file" below for details.

Type 1 SCLK routines

Type 1 SCLK routines are normally called by the higher-level SCLK routines SCENCD, SCDECD, SCS2E, SCT2E, SCE2C, SCE2T, SCE2S, SCTIKS, and SCFMT; you should not need to call these routines directly, though direct calls to these routines are not prohibited.

The type 1 SCLK routines are

SCFM01	(SC, TICKS, CLKSTR)	(Convert ticks to a type 1 SCLK string)

SCTK01	(SC, CLKSTR, TICKS)	(Convert a type 1 SCLK string to ticks)
SCEC01	(SC, ET, SCLKDP)	(ET to continuous ticks, type 1)
SCET01	(SC, ET, SCLKDP)	(Convert ET to ticks, type 1)
SCTE01	(SC, SCLKDP, ET)	(Convert ticks to ET, type 1)
SCLD01	(NAME, SC, MAXNV, N, DVAL)	(SCLK look up of double precision data, type 1)
SCLI01	(NAME, SC, MAXNV, N, IVAL)	(SCLK look up of integer data, type 1)
SCLU01	(NAME, SC, MAXNV, N, IVAL, DVAL)	(SCLK lookup, type 1)
SC01	(SC, CLKSTR, TICKS, SCLKDP, ET)	(SCLK conversion, type

The last two routines SC01 and SCLU01 are "umbrella" routines which exist for the purpose of allowing their entry points to share data. These routines should not be called directly.

The type 1 SCLK kernel file

Before any SPICELIB routines that make use of type 1 SCLK values can be used, a SCLK kernel file must be loaded into the kernel pool. Regardless of the clock type, an SCLK kernel assigns values to variables that define:

- The clock type for a spacecraft
- The format of SCLK strings, for one or more spacecraft.
- The mapping between encoded SCLK values and a parallel time system.

Variables that are used for all clock types have names that start with the string

```
SCLK_
```

Variables that are applicable only to type 1 spacecraft clocks start with the string

```
SCLK01_
```

An SCLK kernel file makes the following assignments for each spacecraft whose clock values are to be treated as "type 1" by the SPICELIB SCLK routines:

Kernel ID assignment

Each SCLK kernel must assign a identifier to the kernel variable

```
SCLK_KERNEL_ID
```

This identifier is normally a UTC time string, preceded by the character '@', for example,

```
@04-SEP-1990
```

If you have loaded multiple SCLK kernels into the kernel pool, the identifiers for these kernels should be distinct.

Parallel time system code assignment

If –ss is the NAIF ID code of a spacecraft, this ID is associated with a parallel time system by the assignment

```
SCLK01_TIME_SYSTEM_ss  = ( nnn )
```

where nnn is a numeric code designating the time system that the coefficients in the kernel map encoded SCLK to. The time systems and codes currently in use are:

```
Barycentric dynamical time (TDB)     1
Terrestrial dynamical time (TDT)     2
```

This assignment is optional; if absent, the parallel time system is assumed to be barycentric dynamical time.

SCLK type assignment

If –ss is the NAIF ID code of a spacecraft, this ID is associated with a SCLK type by the assignment

```
SCLK_DATA_TYPE_ss = ( 1 )
```

Note that multiple mission ID codes can be associated with the type 1 SCLK data type at one time. Since the mission codes are included in the SCLK variable names, there will be no naming conflicts.

Format constant assignments

All of the format constants start with the string

```
SCLK01
```

and end with the string

```
_ss
```

where –ss is the NAIF mission ID code. This allows the type 1 SCLK routines to find the correct constants for each mission ID associated with the first SCLK data type.

The format constants that must be assigned are

```
SCLK01_N_FIELDS_ss
SCLK01_MODULI_ss
SCLK01_OFFSETS_ss
SCLK01_OUTPUT_DELIM_ss
```

Here are sample assignments of values to the variables describing the format of type 1 SCLK strings. The values shown apply to the Galileo SCLK format.

Number of fields:

```
SCLK01_N_FIELDS_77  =  ( 4 )
```

Modulus of each field:

```
SCLK01_MODULI_77  =  ( 16777215 91 10 8 )
```

Offsets for field values. Offsets are listed for each field in left-to-right order:

```
SCLK01_OFFSETS_77 = ( 0 0 0 0 )
```

Code for delimiter to be used in output strings. The codes and corresponding delimiters are:

```
Code          Delimiter

  1               .

  2               :

  3               -

  4               ,

  5             <space>
```

For Galileo, the code assignment would be:

```
SCLK01_OUTPUT_DELIM_77 = ( 2 )
```

Time coefficients

The data that define the mapping between SCLK and the parallel time system are called "time coefficients." This name is used because the data are coefficients of linear polynomials; as a set, they define a piecewise linear function that maps SCLK to the parallel time system.

The time coefficients are assigned to the variable

```
SCLK01_COEFFICIENTS_ss
```

where –ss is the spacecraft ID code. The assigned values are triplets of SCLK values, corresponding parallel time values, and rates. The SCLK values are expressed in total ticks since clock start. The parallel time values may be expressed in a variety of units. The rate values have units that depend on the units used for the parallel time values: if we call these units PARALLEL_TIME_UNITS, then the rate units are

```
   PARALLEL_TIME_UNITS
----------------------------
most significant clock count
```

The term most "significant clock count" shown in the denominator refers to the length of time associated with one count of the most significant (leftmost) field of the formatted spacecraft clock string. For example, for Voyager 2, the most significant field of a formatted SCLK string is the "mod 16" field. For Galileo, the most significant field is the "RIM count." For Mars Global Surveyor, the most significant field is the "sclk_secs count."

Partition boundaries

In order to convert between SCLK strings and their encoded form of ticks since spacecraft clock start, it is necessary to know the initial and final SCLK readouts for each partition. These values are given by:

```
PARTITION_START_ss
PARTITION_END_ss
```

where –ss is the spacecraft ID code. These variables are arrays containing one element per partition.

Sample SCLK kernels

The following is a sample SCLK kernel for Galileo:

```
\begindata
SCLK_KERNEL_ID            = ( @04-SEP-1990//4:23:00 )

SCLK_DATA_TYPE_77         = ( 1                 )
SCLK01_N_FIELDS_77        = ( 4                 )
SCLK01_MODULI_77          = ( 16777215 91 10 8 )
SCLK01_OFFSETS_77         = (        0  0  0 0 )
SCLK01_OUTPUT_DELIM_77    = ( 2                 )

SCLK_PARTITION_START_77   = ( 0.0000000000000E+00
                              2.5465440000000E+07
                              7.2800001000000E+07
                              1.3176800000000E+08 )

SCLK_PARTITION_END_77      = ( 2.5465440000000E+07
                              7.2800000000000E+07
                              1.3176800000000E+08
                              1.2213812519900E+11 )

SCLK01_COEFFICIENTS_77    = (

0.0000000000000E+00  -3.2287591517365E+08  6.0666283888000E+01
7.2800000000000E+05  -3.2286984854565E+08  6.0666283888000E+01
1.2365520000000E+06  -3.2286561063865E+08  6.0666283888000E+01
1.2365600000000E+06  -3.2286558910065E+08  6.0697000438000E+01
1.2368000000000E+06  -3.2286557090665E+08  6.0666283333000E+01
1.2962400000000E+06  -3.2286507557565E+08  6.0666283333000E+01
2.3296480000000E+07  -3.2286507491065E+08  6.0666300000000E+01
2.3519280000000E+07  -3.2286321825465E+08  5.8238483608000E+02
2.3519760000000E+07  -3.2286317985565E+08  6.0666272281000E+01
2.4024000000000E+07  -3.2285897788265E+08  6.0666271175000E+01
2.5378080000000E+07  -3.2284769395665E+08  6.0808150200000E+01
2.5421760000000E+07  -3.2284732910765E+08  6.0666628073000E+01
2.5465440000000E+07  -3.2284696510765E+08  6.0666628073000E+01
3.6400000000000E+07  -3.2275584383265E+08  6.0666627957000E+01
7.2800000000000E+07  -3.2245251069264E+08  6.0666628004000E+01
1.0919999900000E+08  -3.2214917755262E+08  6.0666628004000E+01
1.2769119900000E+08  -3.2199508431761E+08  6.0665620197000E+01
1.3085799900000E+08  -3.2196869477261E+08  6.0666892494000E+01
1.3176799900000E+08  -3.2196111141061E+08  6.0666722113000E+01
1.3395199900000E+08  -3.2194291139361E+08  6.0666674091000E+01
1.3613599900000E+08  -3.2192471139161E+08  6.0666590261000E+01
1.4341599900000E+08  -3.2186404480160E+08  6.0666611658000E+01
1.5069599900000E+08  -3.2180337818960E+08  6.0666611658000E+01
1.7253599900000E+08  -3.2162137835458E+08  6.0666783566000E+01
1.7515679900000E+08  -3.2159953831258E+08  6.0666629213000E+01
1.7777759900000E+08  -3.2157769832557E+08  6.0666629213000E+01
3.3451599900000E+08  -3.2027154579839E+08  6.0666505193000E+01
3.3713679900000E+08  -3.2024970585638E+08  6.0666627480000E+01
3.3975759900000E+08  -3.2022786587038E+08  6.0666627480000E+01
5.6601999900000E+08  -3.1834234708794E+08  6.0666396876000E+01
5.6733039900000E+08  -3.1833142713693E+08  6.0666626282000E+01
```

```
5.6864079900000E+08  -3.1832050714393E+08  6.0666626282000E+01
8.9797999900000E+08  -3.1557601563707E+08  5.9666626282000E+01
8.9798727900000E+08  -3.1557595597007E+08  6.0666626282000E+01
8.9799455900000E+08  -3.1557589430307E+08  6.0666626282000E+01 )

\begintext
```

Below is a sample SCLK kernel file for Mars Global Surveyor. Note that the text prior to the first

```
\begindata
```

directive is treated as a group of comment lines by the SPICELIB kernel readers. The labels shown in this comment area are examples and should not be construed as a correct specification.

```
KPL/SCLK

Status
-----------------------------------------------

   This file is a SPICE spacecraft clock (SCLK) kernel containing
   information required for Mars Global Surveyor spacecraft
   on-board clock to ET conversion.


Production/History of this SCLK files
-----------------------------------------------

   This file was generated by the NAIF utility program MAKCLK,
   version 3.3, from the most recent Mars Global Surveyor
   spacecraft SCLK SCET file.


Usage
-----------------------------------------------

   This file must be loaded into the user's program by a call to
   the LDPOOL subroutine

         CALL LDPOOL( 'this_file_name' )

   in order to use the SPICELIB SCLK family of subroutines to
   convert MGS spacecraft on-board clock to ET and vice versa and
   to use MGS frames defined below as reference frames for
   geometric quantities being returned by high-level SPK and
   CK subroutines.


References
-----------------------------------------------

   1. SCLK Required Reading file, NAIF document number 222
   2. MAKCLK User's Guide, NAIF document number 267
```

```
Inquiries
--------------------------------------------------------

   If you have any questions regarding this file contact

   MGS Spacecraft Operations Team (SCOPS)
   Lockheed/Martin, Denver

   Boris Semenov - NAIF/JPL
   (818) 354-8136
   bsemenov@spice.jpl.nasa.gov


SCLK DATA
--------------------------------------------------------

\begindata

SCLK_KERNEL_ID           = ( @1999-02-07/03:51:29.00 )

SCLK_DATA_TYPE_94        = ( 1 )
SCLK01_TIME_SYSTEM_94    = ( 2 )
SCLK01_N_FIELDS_94       = ( 2 )
SCLK01_MODULI_94         = ( 4294967296 256 )
SCLK01_OFFSETS_94        = ( 0 0 )
SCLK01_OUTPUT_DELIM_94   = ( 1 )

SCLK_PARTITION_START_94  = ( 1.3611133440000E+11 )

SCLK_PARTITION_END_94    = ( 1.0995116277750E+12 )

SCLK01_COEFFICIENTS_94   = (

 0.0000000000000E+00  -9.9510252675000E+07  9.9999996301748E-01
 8.3066265600000E+08  -9.6265476795000E+07  9.9999994844682E-01
 1.9330583040000E+09  -9.1959244017000E+07  9.9999994927604E-01
 2.7708477440000E+09  -8.8686629183000E+07  9.9999994213351E-01
 4.0538009600000E+09  -8.3675093473000E+07  9.9999993609973E-01
 4.7829370880000E+09  -8.0826905655000E+07  9.9999993275158E-01
 5.2473643520000E+09  -7.9012736777000E+07  9.9999993064539E-01
 5.4909818880000E+09  -7.8061105843000E+07  9.9999992770059E-01
 6.7515176960000E+09  -7.3137138199000E+07  9.9999992410889E-01
 7.9017973760000E+09  -6.8643858540000E+07  9.9999992038548E-01
 8.9854187520000E+09  -6.4410962877000E+07  9.9999991689249E-01
 9.9588085760000E+09  -6.0608659193000E+07  9.9999991330346E-01
 1.1222619136000E+10  -5.5671899621000E+07  9.9999990916047E-01
 1.2448517120000E+10  -5.0883236056000E+07  9.9999990447344E-01
 1.3831336704000E+10  -4.5481597572000E+07  9.9999990051645E-01
 1.5223486464000E+10  -4.0043513113000E+07  9.9999989497162E-01
 1.7390367488000E+10  -3.1579135002000E+07  9.9999988993180E-01
 1.7567130624000E+10  -3.0888654078000E+07  9.9999989100000E-01 )
\begintext
```