# Time Routines in CSPICE

## Revisions

### 22 July 1997

This edition of TIME Required Reading documents the routine ET2LST. This routine allows user's to easily convert Ephemeris Time (Barycentric Dynamical Time) to the local solar time at a user specified longitude on the surface of an object.

In addition to the new routine ET2LST, we document a slight extension of the set of time strings that are recognized by the SPICE time software. This extension is documented in Appendix B.

### 15 October 1996

This edition of TIME Required Reading is a substantial revision to the previous edition; this reflects a major enhancement of the SPICE time software. This version describes the new time related software that was included in version N0046 of SPICE . We also draw distinctions between the various levels of time conversion software that are available to Toolkit users.

The following routines are new as of version N0046 of SPICELIB.

```
   STR2ET      TSETYR      TTRANS      JUL2GR
```

```
    TIMOUT      TIMDEF      TPARTV      GR2JUL
    TPICTR      TCHCKD      TCHECK      TEXPYR
```

### 30 June 1994

This version differs substantially from the previous version of 13 April 1992. Much of the description of the time software has been redone and sections added to describe how to modify time string parsing behavior and the conversion between uniform time systems.

### 13 April 1992

This version differs from the previous version of 10 April 1991 in that it discusses the new routine, UNITIM, for converting between additive numeric time systems.

## References

---

The formulation and the values used in this document are taken from the following sources

1. Moyer, T.D., Transformation from Proper Time on Earth to Coordinate Time in Solar System Barycentric Space-Time Frame of Reference, Parts 1 and 2, Celestial Mechanics 23 (1981), 33-56 and 57-68.

2. Moyer, T.D., Effects of Conversion to the J2000 Astronomical Reference System on Algorithms for Computing Time Differences and Clock Rates, JPL IOM 314.5--942, 1 October 1985.

3. The Explanatory Supplement to the Astronomical Almanac (1992) Edited by P. Kenneth Seidelmann, University Science Books, Mill Valley, California 94941

4. SCLK Required Reading.

The variable names used are consistent with notations used in the Astronomical Almanac.

For a general and very accessible discussion of time we recommend:

> 5. James Jespersen and Jane Fitz-Randolph ``From Sundials to Atomic Clocks---Understanding Time and Frequency'' (Dover Publications, Inc. 1977) ISBN 0-486-24265-X

# Introduction

This document describes the software available in the SPICE Toolkit for manipulating various representations of time. It is your main source for general information about calendar based and continuous time systems in SPICE . For specifics of a particular routine you should consult the header of that routine.

In addition to the discussion of time software, there are two appendices to this document. The first provides basic background material on various time systems. The second discusses the details of how time strings are parsed in the SPICE system.

The Toolkit also supports conversion between spacecraft clock (SCLK) and Barycentric Dynamical Time (TDB). However, spacecraft clock conversion is mentioned only in the context of background information in Appendix A. SPICE routines dealing with spacecraft clock are discussed in SCLK Required Reading.

## Intended Audience

This document is intended for all SPICE users.

# Overview

SPICE contains a versatile set of time conversion routines designed to simplify conversions between several time systems. The basic time systems supported are: Coordinated Universal Time (UTC), Barycentric Dynamical Time (TDB) and Terrestrial Dynamical Time (TDT). In addition, most common time formats are supported including: calendar, day of year, and Julian Date.

A brief description of the various time systems is given in Appendix A.

## If You're in a Hurry

We'll discuss things in more detail in a moment, but in case you are just looking for the right name of the routine to perform some time transformation task, here is a classification of the time routines in SPICE. We touch on only the most important routines in the remainder of this overview.

Loading a Leapseconds Kernel

```
   LDPOOL ( FILE )
```

Converting strings to ET

```
   STR2ET ( STRING, ET )

   UTC2ET ( UTCSTR, ET )

   TPARSE ( STRING, SP2000, ERROR )
```

Converting ET to a string

```
   TIMOUT ( ET, PICTUR, STRING )

   ET2UTC ( ET, FORMAT, PREC, UTCSTR )

   ETCAL  ( ET, STRING )
```

Converting between numeric representations of time

```
   UNITIM ( DPTIME, INSYS, OUTSYS )
```

Runtime modification of behavior

```
   TIMDEF ( ACTION, ITEM, VALUE )

   TSETYR ( YEAR )

   TPARCH ( YESNO )
```

Formatting aid

```
   TPICTR ( SAMPLE, PICTUR, OK, ERROR )
```

Converting ET to local solar time on the surface of an object.

```
    ET2LST ( ET, BODY, LONG, TYPE, HR, MN, SC, TIME, AMPM )
```

Foundation routines

```
    TTRANS ( INTYP, OUTTYP, TIMVEC )

    TPARTV ( STRING, TVEC,   NTVEC, TYPE,
   .                 MODIFY, MODS,  YABBRV, SUCCES,
   .                 PICTUR, ERROR )
```

Utilities

```
    DELTET ( EPOCH, EPTYPE, DELTA )

    TEXPYR ( YEAR )

    TCHCKD ( YESNO )

    JUL2GR ( YEAR, MONTH, DAY, DOY )

    GR2JUL ( YEAR, MONTH, DAY, DOY )

    TCHECK ( TVEC, TYPE, MODS, MODIFY, OK, ERROR )
```

Time constants

```
    B1900 ()
    B1950 ()
    J1900 ()
    J1950 ()
    J2000 ()
    J2100 ()
    JYEAR ()
    SPD   ()
    TYEAR ()
```

## The J2000 Epoch

The basic spatial reference system for SPICE is the J2000 system. This is an inertial reference frame in which the equations of motion for the solar system may be integrated. This reference frame is specified by the orientation of the earth's mean equator and equinox at a particular epoch --- the J2000 epoch. This epoch is Greenwich noon on January 1, 2000 Barycentric Dynamical Time. Throughout the SPICE documentation, you will see the expressions: ``seconds past 2000''; ``seconds past J2000''; or ``seconds past the J2000 epoch.'' In all cases, the reference epoch is noon January 1, 2000 on a particular time scale.

(As we've just seen ``J2000'' is used to name the fundamental inertial frame and a particular epoch. This can sometimes be confusing if you are not careful to distinguish the context in which the term ``J2000'' is used.)

## Leapseconds

In almost all cases, before converting between different representations of time you must ``load'' a leapseconds kernel (LSK) into memory. The leapseconds kernel is a text kernel and is loaded via the routine LDPOOL.

```
    LDPOOL ( '<file name of leapseconds kernel>' )
```

The leapseconds kernel is discussed in more detail later in this document.

## Converting Time Strings to Numeric Representations

If you are starting with a representation of time in the form of a string such as ``Mon Sep 30 09:59:10 PDT 1996'' you will normally need to get this into a numeric representation before you can work with it. The basic routine for converting strings to a numeric representation is STR2ET (``String to ET'').

```
    STR2ET ( STRING, ET )
```

STR2ET computes the ephemeris epoch corresponding to an input string. The ephemeris epoch is represented as seconds past the epoch of the J2000 reference frame in the time system known as Barycentric Dynamical Time (TDB). This time system is also referred to as Ephemeris Time (ET) throughout the SPICE Toolkit.

The variety of ways people have developed for representing times is enormous. It is unlikely that any single subroutine can accommodate all of the custom time formats that have arisen in various computing contexts. However, we believe that STR2ET correctly interprets most time formats used throughout the planetary science community. For example STR2ET supports ISO time formats, UNIX `date` output formats. VMS time formats, MS-DOS formats, epochs in both the A.D. and B.C. eras, time zones, etc.

If you've been using the Toolkit for a while you are probably familiar with the routine UTC2ET.

```
    UTC2ET ( UTCSTR, ET )
```

UTC2ET provides a subset of the capabilities contained in STR2ET. It does not recognize time zones or time systems other than the UTC system. However, it has been the work horse for time conversion within the Toolkit for many years. In version N0046 of the Toolkit it was upgraded to support ISO time formats.

If you are writing new code, we recommend that you use the routine STR2ET. There is no need to upgrade any of your existing code that calls UTC2ET. However, you may want to replace calls to UTC2ET with calls to STR2ET due to the greater flexibility of STR2ET.

## Converting Numeric Representations to Time Strings

If you need to examine an epoch given as some double precision number of seconds past J2000, you will normally want to convert it to some more meaningful representation. There are two routines normally used for this task. They offer varying degrees of flexibility in the output strings they can produce. The more general of these is TIMOUT.

```
    TIMOUT ( ET, PICTUR, STRING )
```

Given an epoch ET expressed as double precision seconds past J2000 and a format picture pictur that you would like to use as a model for the output time strings, TIMOUT produces a string representing the input ET in a format that matches the one specified by pictur with the length of the string lenout. Using TIMOUT you can produce a time string in almost any format you desire (including many that cannot be recognized by any of the SPICE software). To assist in creating a format picture the routine TPICTR is provided. TPICTR takes a sample time string and produces the format picture that corresponds to the sample. By using TPICTR and TIMOUT together you can easily produce strings in the format you are used to seeing.

Less flexible, but slightly easier to use, ET2UTC has been the standard SPICE time formatting routine for many years.

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

This routine supports several fixed formats: calendar, Julian Date (UTC), day-of-year, ISO year/month/day, and ISO year/day-of-year. You may adjust the number of digits that follow the decimal point in the seconds component (or day in the Julian Date format).

## Converting between Different Numeric Formats

You may need to convert between different numeric representations of time such as TDT, Julian Date TDB, TAI seconds past J2000, etc. The routine UNITIM is available for such conversions.

```
    UNITIM ( DPTIME, INSYS, OUTSYS )
```

# Initialization

## Leapseconds Kernel

Most SPICE time routines make use of the information contained in a leapseconds kernel. Specifically, all of the following routines make use of the leapseconds kernel.

**STR2ET**

Converts strings to ET.

**UTC2ET**

Converts UTC strings to ET

**TIMOUT**

Converts ET to strings

**ET2UTC**

Converts ET to a UTC string.

**UNITIM**

Converts between numeric time systems

**TTRANS**

Converts between different parsed representations of time

Before any of these routines can be used you must ``load'' a leapseconds kernel into the ``kernel pool.'' This is done by calling the routine LDPOOL, whose calling sequence is:

```
   LDPOOL ( KERNEL )
```

KERNEL is the name of a ``leapseconds kernel.'' Leapseconds kernels are text based kernels containing the epochs of leap seconds and other constants required by the time conversion routines.

The leapseconds kernel needs to be loaded just once per program run; normally, the leapseconds kernel is loaded in a program's initialization section.

The precise contents of the leapseconds kernel are discussed in the section ``Computing Delta ET'' below. Text kernels and the routine LDPOOL are discussed in more detail in KERNEL Required Reading.

### SPK and PCK kernels

The routine ET2LST converts ephemeris time (ET) to the local solar time for a point at a user specified longitude on the surface of a body. This computation is performed using the bodyfixed location of the sun. Consequently, to use ET2LST you must first load SPK and PCK files that contain sufficient position and orientation data for the computation of the bodyfixed location of the sun.

SPK files are loaded using the routine SPKLEF.

```
    SPKLEF ( '<spk file name>', HANDLE )
```

PCK files are usually text based. Text based kernels are loaded by calling LDPOOL.

```
    LDPOOL ( KERNEL )
```

Occasionally, PCK, files are binary (DAF based) files that contain the orientation of an object with respect to an inertial frame. Binary PCK files are loaded with the routine PCKLOF.

```
    PCKLOF ( '<binary pck file name>', HANDLE )
```

As with the leapseconds kernel, SPK and PCK files need to be loaded just once per program run---usually at program initialization.

## Input String Conversion

---

We normally represent epochs as a combination of a date and time of day. In C the simplest means of specifying an epoch as a date and time is to create a string such as:

```
    STRING = 'Oct 1, 1996 09:12:32'
```

However, arithmetic is most easily performed with numeric representations of time. In SPICE we represent epochs as some number of double precision seconds past the J2000 epoch.

SPICE contains three routines for converting strings directly to ``seconds past 2000.'' They are STR2ET, UTC2ET, and TPARSE. All of these routines take a string as input and produce a double precision number that gives the number of seconds past the J2000 epoch corresponding to the input string. The method of analyzing the input string and assigning meaning to its various components is identical for all three routines. This analysis is called ``parsing'' the string. All three routines, STR2ET, UTC2ET and TPARSE, use the ``foundation'' routine TPARTV to parse the input string. Each then interprets the results of TPARTV to assign meaning to the string. Below are a number of examples of strings and the interpretation assigned to the various components.

ISO (T) Formats.

```
   String                         Year Mon  DOY DOM  HR Min Sec
   ----------------------------   ---- ---  --- ---  -- --- ------
   1996-12-18T12:28:28            1996 Dec   na  18  12  28 28
   1986-01-18T12                  1986 Jan   na  18  12  00 00
   1986-01-18T12:19               1986 Jan   na  18  12  19 00
   1986-01-18T12:19:52.18         1986 Jan   na  18  12  19 52.18
   1995-08T18:28:12               1995  na  008  na  18  28 12
   1995-18T                       1995  na  018  na  00  00 00
```

Calendar Formats.

```
   String                         Year   Mon DOM  HR Min  Sec
   ----------------------------   ----   --- ---  -- ---  ------
   Tue Aug  6 11:10:57  1996      1996   Aug  06  11  10  57
   1 DEC 1997 12:28:29.192        1997   Dec  01  12  28  29.192
   2/3/1996 17:18:12.002          1996   Feb  03  17  18  12.002
   Mar 2 12:18:17.287 1993        1993   Mar  02  12  18  17.287
   1992 11:18:28  3 Jul           1992   Jul  03  11  18  28
   June 12, 1989 01:21            1989   Jun  12  01  21  00
   1978/3/12 23:28:59.29          1978   Mar  12  23  28  59.29
   17JUN1982 18:28:28             1982   Jun  17  18  28  28
   13:28:28.128 1992 27 Jun       1992   Jun  27  13  28  28.128
   1972 27 jun 12:29              1972   Jun  27  12  29  00
   '93 Jan 23 12:29:47.289        1993*  Jan  23  12  29  47.289
   27 Jan 3, 19:12:28.182         2027*  Jan  03  19  12  28.182
   23 A.D. APR 4, 18:28:29.29     0023   Apr  04  18  28  29.29
   18 B.C. Jun 3, 12:29:28.291    -017   Jun  03  12  29  28.291
   29 Jun  30 12:29:29.298        2029+  Jun  30  12  29  29.298
   29 Jun '30 12:29:29.298        2030*  Jun  29  12  29  29.298
```

Day of Year Formats

```
   String                         Year  DOY HR Min Sec
   ----------------------------   ----  --- -- --- ------
   1997-162::12:18:28.827         1997  162 12  18 28.827
   162-1996/12:28:28.287          1996  162 12  28 28.287
   1993-321/12:28:28.287          1993  231 12  28 28.287
   1992 183// 12 18 19            1992  183 12  18 19
   17:28:01.287 1992-272//        1992  272 17  28 01.287
   17:28:01.282 272-1994//        1994  272 17  28 01.282
   '92-271/ 12:28:30.291          1992* 271 12  28 30.291
   92-182/ 18:28:28.281           1992* 182 18  28 28.281
   182-92/ 12:29:29.192           0182+ 092 12  29 29.192
   182-'92/ 12:28:29.182          1992  182 12  28 29.182
```

Julian Date Strings

```
   jd 28272.291                   Julian Date   28272.291
   2451515.2981 (JD)              Julian Date 2451515.2981
   2451515.2981 JD               Julian Date 2451515.2981
```

Abbreviations Used in Tables

```
   na    --- Not Applicable
   Mon   --- Month
   DOY   --- Day of Year
   DOM   --- Day of Month
   Wkday --- Weekday
   Hr    --- Hour
   Min   --- Minutes
   Sec   --- Seconds
```

*

> The default interpretation of a year that has been abbreviated with a leading quote as in 'xy (such as '92) is to treat the year as 19xy if xy is more than 49 and to treat it is 20xy otherwise. Thus '52 is interpreted as 1952 and '47 is treated as 2047.

+

> When a day of year format or calendar format string is input and neither of the integer components of the date is greater than 1000, the first integer is regarded as being the year.

## Parsing Time Strings

A time string is parsed by first scanning the string from left to right and identifying recognizable substrings. (integers, punctuation marks, names of months, names of weekdays and time systems, time zones, etc.) These recognizable substrings are called the tokens of the input string. The meaning of some tokens are immediately determined. For example named months, weekdays and time systems have clear meanings. However, the meanings of numeric components must be deciphered from their magnitudes and location in the string relative to the immediately recognized components of the input string.

The following substrings are immediately recognizable.

1. All months (January, February, ... ) or any abbreviation of at least 3 letters;

2. All weekdays (Sunday, Monday, ... ) or any abbreviation of at least 3 letters;

3. Standard abbreviations of U.S. time zones: 'EST', 'EDT', 'CST', 'CDT', 'MST', 'MDT', 'PDT', 'PST'.

4. The abbreviations for eras: 'B.C.', 'BC', 'A.D.', and 'AD';

5. Time systems: 'TDT', 'TDB', 'UTC' (Note that 'ET' is not a recognized time system);

6. Julian Date Label: 'JD' (Note that JED is not a recognized Julian Date Label);

7. The 12-hour clock labels: 'A.M.', 'AM', 'P.M.' and 'PM';

8. Time Zones expressed as UTC offsets: UTC+HR:MN, UTC-HR:MN where HR is an unsigned integer between 0 and 12 inclusive; MN is an unsigned integer between 0 and 59 inclusive.

With the exception of months, all items above may be enclosed in parentheses. For example 'TDB' and '(TDB)' are both recognized as the same time system.

The case of the letters in these substrings does not matter. For example all of the various ways of writing 'TDB' ( 'TDB', 'tDB', ... 'tdb') are recognized as 'TDB'.

It is not necessary to leave space between the various substrings. For example JDTDT and JDUTC are recognized as 'JD' followed by 'TDT' and 'JD' followed by 'UTC' respectively.

To determine the meaning of the numeric tokens in the input string, a set of transformation rules are applied to the full set of tokens in the string. These transformations are repeated until the meaning of every token has been determined or until further transformations yield no new clues into the meaning of the numeric tokens. Here is an overview of the rules that are applied to the various tokens in the string.

1. Unless the substring JD or jd is present the string is assumed to be a calendar format (day-month-year or year and day of year). If the substring JD or jd is present, the string is assumed to represent a Julian date.

2. If the Julian date specifier is not present, any integer greater than 999 is regarded as being a year specification.

3. A dash `-' can represent a minus sign only if it precedes the first digit in the string and the string contains the Julian date specifier (JD). (No negative years, months, days, etc are allowed).

4. Numeric components of a time string must be separated by a character that is not a digit or decimal point. Only one decimal component is allowed. For example 1994219.12819 is sometimes interpreted as the 219th day of 1994 + 0.12819 days. The SPICE time parsing software does not support such strings.

5. No exponential components are allowed. For example you can't input 1993 Jun 23 23:00:01.202E-4. You have to explicitly list all zeros that follow the decimal point: i.e. 1993 Jun 23 23:00:00.0001202

6. The single colon (:) when used to separate numeric components of a string is interpreted as separating Hours, Minutes, and Seconds of time.

7. If a double slash (//) or double colon (::) follows a pair of integers, those integers are assumed to represent the year and day of year.

8. A quote followed by an integer less than 100 is regarded as an abbreviated year. For example: '93 would be regarded as the 93rd year of the reference century. See TEXPYR for further discussion of abbreviated years.

9. An integer followed by 'B.C.' or 'A.D.' is regarded as a year in the era associated with that abbreviation.

10. All dates are regarded as belonging to the extended Gregorian Calendar (the Gregorian calendar is the calendar currently used by western society).

11. If the ISO date-time separator (T) is present in the string, only ISO allowed token patterns are examined for a match with the current set of tokens. If no match is found the search is abandoned and appropriate diagnostic messages are generated.

12. If two delimiters are found in succession in the time string, the time string is diagnosed as an erroneous string. (Delimiters are comma, white space, dash, slash, period, day of year mark)

Note the delimiters do not have to be the same. The pair of characters ``,-" counts as two successive delimiters.

13. White space and commas serve only to delimit tokens in the input string; they do not affect the meaning of any of the tokens.

14. When the sizes of the integer components do not clearly specify a year but the name of a month is present (for example 'APR') the following patterns are assumed

```
Year Month Day
Month Day Year
Year Day Month
```

15. When integer components are separated by slashes (/) as in 3/4/5. The integers are assumed to be Month, Day, Year. Thus in our example '3/4/5' is assumed to mean 4th of March in the year '05.

16. If a day of year marker is present (// or ::) and the size of the integer components does not clearly specify the year (as in 45-33//) the string is interpreted as Year Day-of-Year. Thus 45-33// is interpreted as the 33rd day of the year '45.

Once the various tokens have been determined and a meaning attached to them, the routines STR2ET, UTC2ET, and TPARSE, use the tokens to construct the double precision number giving the number of seconds past J2000 that corresponds to input string. However, not all tokens or token combinations are allowed by the routines.

## STR2ET

The routine STR2ET is the most flexible of the three time transformation routines. STR2ET accepts the widest variety of time strings. To illustrate the various features of STR2ET we begin by considering the string

```
   1988 June 13, 3:29:48
```

There is nothing in this string to indicate what time system the date and time belong to. Moreover, there is nothing to indicate whether the time is based on a 24-hour clock or twelve hour clock.

In the absence of such indicators, the default interpretation of this string is to regard the time of day to be a time on a 24-hour clock in the UTC time system. The date is a date on the Gregorian Calendar (this is the calendar used in nearly all western societies).

## Labels (A.M. and P.M.)

If you add more information to the string, STR2ET can then make a more informed interpretation of the time string. For example:

```
   1988 June 13, 3:29:48 P.M.
```

is still regarded as a UTC epoch. However, with the addition of the ``P.M.'' label it is now interpreted as the same epoch as the unlabeled epoch 1988 June 13, 15:29:48. Similarly

```
   1988 June 13, 12:29:48 A.M.
```

is interpreted as

```
   1988 June 13, 00:29:48
```

on the 24-hour clock.

## For the Record

12:00 A.M. corresponds to Midnight (00:00 on the 24-hour clock). 12:00 P.M. corresponds to Noon (12:00 on the 24-hour clock).

## Labels (Time Zones)

You may add still further indicators to the string. For example

```
   1988 June 13, 3:29:48 P.M. PST
```

is interpreted as an epoch in the Pacific Standard Time system. This is equivalent to

```
   1988 June 13, 23:29:48 UTC
```

All of the standard abbreviations for U.S. time zones are recognized by the time parser.

```
   EST   --- Eastern Standard Time  ( UTC-5:00 )
   CST   --- Central Standard Time  ( UTC-6:00 )
   MST   --- Mountain Standard Time ( UTC-7:00 )
   PST   --- Pacific Standard Time  ( UTC-8:00 )

   EDT   --- Eastern Daylight Time  ( UTC-4:00 )
   CDT   --- Central Daylight Time  ( UTC-5:00 )
   MDT   --- Mountain Daylight Time ( UTC-6:00 )
   PDT   --- Pacific Daylight Time  ( UTC-7:00 )
```

In addition, any other time zone may be specified by representing its offset from UTC.

To specify an offset from UTC you need to create an offset label. The label starts with the letters `UTC' followed by a `+' for time zones east of Greenwich and `-' for time zones west of Greenwich. This is followed by the number of hours to add or subtract from UTC. This is optionally followed by a colon `:' and the number of minutes to add or subtract to get the local time zone. Thus to specify the time zone of Calcutta (which is 5 and 1/2 hours ahead of UTC) you would specify the time zone to be UTC+5:30. To specify the time zone of Newfoundland (which is 3 and 1/2 hours behind UTC) use the offset notation UTC-3:30.

## For the Record

Leapseconds occur at the same time in all time zones. In other words, the seconds component of a time string is the same for any time zone as is the seconds component of UTC. The following are all legitimate ways to represent an epoch of some event that occurred in the leapsecond

```
   1995 December 31 23:59:60.5  (UTC)


   1996 January   1, 05:29:60.5  (UTC+5:30 --- Calcutta Time)
   1995 December 31, 20:29:60.5  (UTC-3:30 --- Newfoundland)
   1995 December 31  18:59:60.5  (EST)
   1995 December 31  17:59:60.5  (CST)
   1995 December 31  16:59:60.5  (MST)
   1995 December 31  15:59:60.5  (PST)
```

### Labels ( TDT, TDT, and UTC )

In addition to specifying time zones you may specify that the string be interpreted as a formal calendar representation in either the Barycentric Dynamical Time system (TDB) or the Terrestrial Dynamical Time system (TDT).

In these systems there are no leapseconds; every day has exactly 86400 seconds. TDB times are written as

```
   1988 June 13, 12:29:48 TDB
```

TDT times are written as:

```
   1988 June 13, 12:29:48 TDT
```

To add clarity or to override any changes you happen to make to the default behavior of ET2STR (see below) you may add the label ``UTC'' to any time string.

```
   1998 Jun 13, 12:29:48 UTC
```

Note that the system label may be placed anywhere in the time string. All of the following will be understood by the time parsing software:

```
   TDB 1988 June 13, 12:29:48
   1988 June 13, 12:29:48 TDB
   1988 June 13, TDB 12:29:48
```

## UTC2ET

---

The routine UTC2ET can be thought of as a version of STR2ET that allows a narrower range of inputs. It converts strings in the UTC system to TDB seconds past the J2000 epoch. It does not support other time systems or time zones. In addition UTC2ET does not recognize times on a 12-hour clock. Strings such as

```
   1983 June 13, 9:00:00 A.M.
```

are treated as erroneous by UTC2ET.

## TPARSE

The routine TPARSE can be thought of as a narrow version of STR2ET that allows only TDB as input. TPARSE converts strings on a formal time scale to seconds past the J2000 epoch. TPARSE doesn't ``know'' anything about leapseconds. Since TPARSE does not make use of leapseconds, it can be used without first loading a leapseconds kernel.

Like UTC2ET, TPARSE does not recognize other time systems or time zones. Also it does not recognize times on a 12-hour clock.

Unlike STR2ET and UTC2ET, TPARSE does not make use of the SPICE exception handling subsystem. Erroneous strings are diagnosed via a string---ERROR. If the string ERROR is returned empty (blank) no problems were detected in the input string. If ERROR is returned by TPARSE non-blank, it contains a diagnostic message that indicates problems with the input time string.

## Changing Default Behavior

The three time string transformation routines can be adjusted at run time so that various built in defaults can be changed without re-writing any of the code for the routines.

### Abbreviated Years

All three string transformation routines treat abbreviated years in the same fashion. The default behavior is to map any abbreviated year into the range from 1968 to 2067. Thus the year 22 corresponds to 2022; 77 corresponds to 1977. However, you may reset the lower end of this 100 year range via the routine TSETYR. For example if you would like to set the default range to be from 1972 to 2071 issue the following subroutine call:

```
    TSETYR ( 1972 )
```
Note that this change affects the behavior of all three string conversion routines.

## Range of Time String Components

The routines TPARSE and UTC2ET accept time strings whose numeric components are outside of the normal range of values used in time and calendar representations. For example strings such as

```
   1985 FEB 43 27:65:25  (equivalent to 1985 MAR 16 04:05:25)
```
will be accepted as input. You might wish to restrict the range of input strings so that this behavior is not allowed. The routine TPARCH is provided for this purpose. If you place the following subroutine call

```
    TPARCH ( 'YES' )
```
early in your program, prior to any calls to UTC2ET or TPARSE, the components of calendar strings will be restricted so that all calendar components will be in the ``expected'' range. (The exact ranges for the components are spelled out in the header for TPARCH )

STR2ET does not accept time strings whose components are outside the normal range used in conversation. You cannot alter this behavior without re-coding STR2ET.

## Default Time Systems and Time Zone

When a string is presented without a time system or time zone label STR2ET assumes that the string represents a time in a default time zone or time system. If you take no action, the default time system is UTC. (There is no time zone offset; UTC is the same as UTC+00:00) You can override the default by simply including the time zone or time system of interest in the input time string. However, under some circumstances you may find that you almost always use the TDB time system. In such a case you would normally need to include the TDB label in the time string every time you use STR2ET. Hence, the defaults used by STR2ET might be a hindrance rather than a convenience. With this possibility in mind, STR2ET has been designed so that you may alter its default behavior with regard to default time system or time zone. To change the default time system or time zone use the routine TIMDEF.

(Keep in mind that if you specify a time zone or time system label in the input time string the default time zone or system is not used. The label in the string is used to determine the time zone or time system.)

## Changing the Time System

Three time systems are supported: UTC, TDB, TDT. To change the default system to one of these three systems issue the appropriate subroutine call below:

```
    TIMDEF ( 'SET', 'SYSTEM', 'UTC' )
    TIMDEF ( 'SET', 'SYSTEM', 'TDB' )
    TIMDEF ( 'SET', 'SYSTEM', 'TDT' )
```
Note that setting a time system turns off any default time zone you may have set using TIMDEF.

## Time Zones

All time zones are supported by STR2ET. The default time zone is simply Greenwich Mean Time (UTC+00:00). To change the default behavior of STR2ET so that unlabeled strings are assumed to be referenced to a particular time zone (for example Pacific Standard Time) issue the subroutine call below.

```
    TIMDEF ( 'SET', 'ZONE', 'PST' )
```
Note that setting a time zone turns off any default time system you may have set via TIMDEF.

## Calendars

The default calendar used by STR2ET is the Gregorian calendar. However, the Gregorian calendar did not come into existence until October 15, 1582. To complicate matters, many countries did not adopt the Gregorian calendar until centuries later. Prior to adoption of the Gregorian calendar most western societies used the Julian calendar. The generation of successive days is identical on the Julian and Gregorian calendars except for the determination of leap days in the last year of a century such as the year 1900. On the Julian calendar, a leap day is inserted as the last day of February every 4 years. on the Gregorian calendar, a leap day is inserted as the last day of February every 4 years with the possible exception of the last year of a century (such as 1900). The last year of a century is a leap year only if the year is evenly divisible by 400. Thus the year 2000 is a leap year on the Gregorian calendar but 1900 is not.

Both the Gregorian and Julian calendars can be extended forward and backward in time indefinitely. The default behavior of STR2ET is to use the Gregorian calendar for all epochs. However, using TIMDEF you can set the default calendar to one of three: GREGORIAN, JULIAN, or MIXED.

```
    TIMDEF ( 'SET', 'CALENDAR', 'GREGORIAN' )
    TIMDEF ( 'SET', 'CALENDAR', 'JULIAN'    )
    TIMDEF ( 'SET', 'CALENDAR', 'MIXED'     )
```

The ``MIXED'' calendar assumes that calendar strings for epochs prior to October 6, 1582 belong to the Julian Calendar; strings for later epochs are assumed to belong to the Gregorian Calendar. The specification of a calendar, does not affect a previous setting of a time system or time zone. You can change the calendar used by STR2ET only through the routine TIMDEF, there are no labels recognized by STR2ET for the various calendars.

## Output Conversion

Times need to be printed out as well as read in. SPICE contains three routines for accomplishing this task: TIMOUT, ET2UTC, and ETCAL. All three convert a number of ephemeris seconds past J2000 to a time string.

### TIMOUT

```
    TIMOUT ( ET, PICTUR, OUTSTR )
```

where

**ET**

is a double precision number containing the number of TDB seconds past J2000 corresponding to some epoch.

**PICTUR**

is a characters string that describes how the output string should be formatted. It is a ``picture'' of the format for output.

**OUTPUT**

is the string corresponding to ET and PICTUR.

To see how this works, consider the following example time string:

```
   04:29:29.292 Jan 13, 1996
```

The value of PICTUR to use to create time strings that are similar in appearance to the example string is:

```
   PICTUR = 'HR:MN:SC.### Mon DD, YYYY ::RND'
```

Most of this components in PICTUR are fairly obvious. The exception is the substring

```
   '::RND'.
```

This substring tells TIMOUT to round the seconds portion of the output string instead of simply truncating. (Note that the case of the letters is significant in pictur.) TIMOUT can produce strings representing epochs in the time systems (UTC, TDB, TDT) or any time zone, and on either the Julian, Gregorian Calendar or Mixed Calendar. You may round or truncate numeric components.

The rules for constructing pictur are spelled out in the header to TIMOUT. However, you may very well never need to learn these rules. SPICE contains the routine TPICTR that can construct a time format picture for you from a sample time string. Returning to the example above, if the following block of code is executed, pictur will contain the format picture that will yield output strings similar to our example string.

```
    EXAMPL = '04:29:29.292 Jan 13, 1996'

    TPICTR ( EXAMPL, PICTUR, OK, ERROR )
```

The arguments ok and error are outputs from TPICTR. They are present because some strings are not recognized as time strings. TPICTR recognizes the same set of time strings as does STR2ET, UTC2ET and TPARSE. However, if you want your output string to be in a system other than UTC you must supply the label for that system in your example string. TPICTR can construct format pictures for strings that are not accepted by the string conversion routines. For example, if you would like to suppress the year in a calendar output format, you could use the following example string:

```
    EXAMPL = 'Jan 12, 02:28:29.### A.M. (PDT)'
```

Even though this string is ambiguous as an epoch (there's no year specified), it is sufficient for determining a picture that describes its format. If you decide to use TPICTR with inputs like this, be sure to check the output flag OK. Even though you know what is intended, TPICTR may have problems with some ambiguous time strings.

## ET2UTC

The routine ET2UTC is an older time formatting routine. It is not as flexible as TIMOUT. All outputs are UTC outputs and only a limited set of formats are supported. On the other hand it is easier to learn how to use ET2UTC. ET2UTC is an inverse to UTC2ET: that is following the calls

```
    UTC2ET ( UTCIN,  ET             )
    ET2UTC ( ET,    'C',  3, UTCOUT )
```

utcout is identical in content to (although probably formatted differently from) UTCIN. ET2UTC can create time strings in any of the following formats.

```
   Format      Name            Example
   ------      -----------     --------------------------
   'C'         Calendar        '1979 JUL 04 14:19:57.184'
   'D'         Day of Year     '1979-114 // 14:19:57.184'
   'J'         Julian Date     'JD 2433282.529'
   'ISOC'      ISO Calendar    '1987-04-122T16:31:12.814'
   'ISOD'      ISO Day of Year '1987-102T16:31:12.814'
```

In addition, you may specify the number of decimal places in the fractional part of the seconds token or the Julian Date (three, in the examples above). Note that Julian Dates are prefaced with the character string `JD' (and are UTC Julian Dates). This allows strings generated by ET2UTC to be used later as inputs to UTC2ET or STR2ET.

### ETCAL

The routine ETCAL is a utility routine. It can produce outputs in a single format with a fixed number of decimal places. Moreover, the calendar strings it produces are on a formal calendar. There are no leapseconds; each day has exactly 86400 seconds. Since it does not make use of leapseconds, you don't need to load a leapseconds kernel prior to calling ETCAL. This makes it well suited for producing diagnostic messages. Indeed, it was created so that more user friendly diagnostic messages could be produced by those SPICE routines that require ET as an input.

## Converting Between Uniform Time Scales

We use the term uniform time scale to refer to those representations of time that are numeric (each epoch is represented by a number) and additive. A numeric time system is additive if given the representations E1 and E2 of any pair of successive epochs, the time elapsed between the epochs is given by the difference E2 - E1.

Conversion between uniform time scales can be carried out via the double precision function UNITIM. The uniform time scales that are supported by this routine are:

```
   'TAI'     International Atomic Time.
   'TDB'     Barycentric Dynamical Time.
   'TDT'     Terrestrial Dynamical Time.
   'ET'      Ephemeris time
   'JDTDB'   Julian Date relative to TDB.
   'JDTDT'   Julian Date relative to TDT.
   'JED'     Julian Ephemeris date.

   *  In the @SPICE system ET  is synonymous to TDB.
   ** In the @SPICE system JED is synonymous to JDTDB.
```

## Local Solar Time

---

Local solar time is a used to give people an idea of how high the sun is in the sky as seen from a particular site on surface of a planet or satellite. When the Sun is on the zenith meridian, the local solar time is 12:00:00 noon. For points on the equator of a body, the Sun rises around 6:00:00 A.M. local solar time; it sets around 6:00:00 P.M. local solar time.

Formally, the local solar time at a site on a body is the difference between the planetocentric longitude of the site and the planetocentric longitude of the Sun as seen from the center of the body. The angular difference in these two longitudes is measured in hours, minutes, and seconds in the same sense that hours, minutes and seconds are used to measure right ascension--- 24 hours in 360 degrees; 60 minutes in an hour; 60 seconds in a minute. When the sun in on the zenith meridian the hour is defined to be 12. Finally, the hours increase from sunrise to sunset.

Because of these conventions, an hour of local solar time will not be of the same duration as a UTC hour. In the case of a site on Mars, a solar hour will be approximately 62 UTC minutes.

Local solar time for a specific site can be computed using the routine ET2LST (ET to Local Solar Time).

## Foundation Routines and Utilities

At the heart of the SPICE time software subsystem are the ``foundation'' routines TPARTV and TTRANS. TPARTV is used to take apart a time string and convert it to a vector of numeric components. TTRANS serves the role of converting between the various numeric vector representations of time. If you need to build your own time conversion routines, these routines are a good place to begin.

In addition to the foundation routines, you may find helpful the following utility routines.

**TEXPYR**

converts two-digit abbreviated years to full years. You set lower bound of the 100 year mapping interval via the routine TSETYR discussed earlier in this document.

**TCHECK**

takes a numeric vector representing the components of a calendar time and checks that all components are within the normal range used in conversation. Note that TCHECK performs no action until you call TPARCH with an argument of "YES".

**TCHCKD**

allows you to determine if component checking has been enabled in TCHECK via a call to TPARCH.

**JUL2GR**

converts the year, month, and day of an epoch on the Julian Calendar to the corresponding year, month, day and day-of-year on the Gregorian calendar.

**GR2JUL**

converts the year, month, and day of an epoch on the Gregorian Calendar to the corresponding year, month, day and day-of-year on the Julian calendar.

**DELTET**

computes the time difference TDB - UTC.

**B1900**

returns the Julian ephemeris date (TDB) of the epoch of the Besselian date 1900.

**B1950**

returns the Julian ephemeris date (TDB) of the epoch of the Besselian date 1950.

**J1900**

returns the Julian Date of 1899 DEC 31 12:00:00 (TDB)

**J1950**

returns the Julian ephemeris date of the epoch 1 Jan 1950 00:00:00 (TDB).

**J2000**

returns the Julian ephemeris date of the epoch 1 Jan 2000 12:00:00 (TDB).

**J2100**

returns the Julian ephemeris date of the epoch 1 Jan 2100 12:00:00

**JYEAR**

returns the number of seconds in a Julian year (365.25 Julian days).

**SPD**

returns the number of TDB seconds in a Julian day TDB (86400 seconds).

**TYEAR**

returns the number of seconds in a tropical year (approximately the number of seconds from one spring equinox to the next)

## Example

---

The following program demonstrates use of the time conversion routines STR2ET, TPICTR, TIMOUT and ET2UTC.

Note that the data necessary to convert between UTC and ET are loaded into the kernel pool just once---typically during program initialization--- after which the conversion may be performed at any level within the program.

```
      PROGRAM EXAMPLE
C
C     Convert between UTC and ET interactively, and convert ET
C     back to UTC in calendar format, DOY format, and as a
C     Julian date.
C
C     Requires a leapseconds kernel.
C
      INTEGER               FILEN
      PARAMETER           ( FILEN = 128 )

      INTEGER               LNSIZE
      PARAMETER           ( LNSIZE = 60 )


      CHARACTER*(8)         ANSWER
      CHARACTER*(FILEN)     KERNEL
      CHARACTER*(LNSIZE)    DOY
      CHARACTER*(LNSIZE)    ERROR
      CHARACTER*(LNSIZE)    EXAMP1
      CHARACTER*(LNSIZE)    EXAMP2
      CHARACTER*(LNSIZE)    JDUTC
      CHARACTER*(LNSIZE)    PICTR1
      CHARACTER*(LNSIZE)    PICTR2
      CHARACTER*(LNSIZE)    PST
      CHARACTER*(LNSIZE)    STR
      CHARACTER*(LNSIZE)    UTC

      DOUBLE PRECISION      ET

      LOGICAL               OK

C
C     Get the name of the leapseconds kernel file.
C
      WRITE (*,*)  'We need to load a leapseconds kernel.'
      CALL PROMPT ('Kernel Name: ', KERNEL )

C
C     Load the leapseconds kernel into the kernel pool.
C
      CALL LDPOOL ( KERNEL )


C
C     Create pictures for producing strings similar to
C     those below.
C
      EXAMP1 = 'Fri Oct 04, 08:57:28.000 (UTC) 1996'
      EXAMP2 = 'Fri Oct 04, 08:57:28.000 (PST) 1996'

      CALL TPICTR ( EXAMP1, PICTR1, OK, ERROR )
      CALL TPICTR ( EXAMP2, PICTR2, OK, ERROR )


C
C     Compute result for each new UTC epoch.
C
      ANSWER = 'Y'

      DO WHILE (      ( ANSWER(1:1) .EQ. 'Y' )
     .           .OR. ( ANSWER(1:1) .EQ. 'y' )  )

         WRITE (*,*) ' '
         CALL PROMPT ( 'Enter a time: ', STR )

         CALL STR2ET ( STR, ET )

         WRITE (*,*) ' '
         WRITE (*,*) 'Input time converts to ET ' //
     .               '(sec past J2000)', ET


         CALL TIMOUT ( ET, PICTR1,    UTC   )
         CALL TIMOUT ( ET, PICTR2,    PST   )
         CALL ET2UTC ( ET, 'ISOC', 3, DOY   )
         CALL ET2UTC ( ET, 'J',    7, JDUTC )

         WRITE (*,*) ' '
         WRITE (*,*) 'ET converts back to'
         WRITE (*,*) ' '
         WRITE (*,*) UTC
         WRITE (*,*) PST
         WRITE (*,*) ' '
         WRITE (*,*) DOY
         WRITE (*,*) JDUTC

         WRITE (*,*) ' '
         CALL PROMPT ('Do you wish to continue?', ANSWER )

      END DO

      END
```

# Appendix A. Background Material

The Toolkit directly supports three time systems. They are

1. Coordinated Universal Time (UTC)
2. Barycentric Dynamical Time (TDB) also called Ephemeris Time (ET)
3. Spacecraft Clock Time (SCLK---pronounced ``ess clock")

## Coordinated Universal Time (UTC)

### International Atomic Time (TAI)

Before discussing Coordinated Universal Time we feel it is helpful to talk about International Atomic Time (TAI or atomic time). Atomic time is based upon the atomic second as defined by the ``oscillation of the undisturbed cesium atom.'' Atomic time is simply a count of atomic seconds that have occurred since the astronomically determined instant of midnight January 1, 1958 00:00:00 at the Royal Observatory in Greenwich, England. Atomic time is kept by the International Earth Rotation Service (IERS, formally the Bureau International L'Heure---BIH) in Paris, France. The National Bureau of Standards and the U.S. Naval Observatory set their clocks by the clock maintained by the IERS.

## Naming the seconds of TAI --- UTC

Coordinated Universal Time is a system of time keeping that gives a name to each instant of time of the TAI system. These names are formed from the calendar date and time of day that we use in our daily affairs. They consist of 6 components: year, month, day, hour, minutes and seconds. The year, month and day components are the normal calendar year month and day that appear on wall calendars. The hours component may assume any value from 0 through 23. The minutes component may assume any value from 0 to 59. The seconds will usually (but not always) range from 0 to 59.999... . The hour-minute-second string

```
   '00:00:00'
```

is midnight and is the first instant of the calendar day specified by the first three components of the UTC time.

In the SPICE system UTC times are represented by character strings. These strings contain: year, month, day, hour, minute and second separated by delimiters (spaces or punctuation marks). The various delimiters and substrings between the delimiters are called the tokens of the string. A typical time string looks like

```
       '5 OCTOBER 1986 7:20:16.122 (UTC)'
```

The tokens of the string and the associated UTC time components are

```
   '5'       --- day
   'OCTOBER' --- month
   '1986'    --- year
   '7'       --- hours
   '20'      --- minutes
   '16.122'  --- seconds
```

The link between any token and its corresponding UTC component is determined by examining the values of the tokens and comparing them to the other tokens. The precise rules used are spelled out in great detail in appendix 2. For now, simply be assured that the following five strings all mean the same thing and are interpreted in the same way by SPICE Toolkit software.

```
   '5 OCTOBER 1986'
   '1986 OCTOBER 5'
   '1986 5 OCTOBER'
   '1986 10 5'
   '10 5 1986'
```

## Tying UTC to the Earth's Rotation

The names given to TAI instants by the UTC system are governed by the earth's rotation. Ideally, UTC strings having hours, minutes and seconds components all zero should correspond to Greenwich midnight as determined by the observations of the transits of stars (the time system known as UT1). However, since the rotation of the earth is not uniform, this ideal cannot be realized. The difference between Greenwich midnight observed astronomically and UTC midnight is almost never zero. However, to keep the difference from becoming too large, UTC is occasionally adjusted so that the difference between the two midnights never exceeds .9 seconds. Thus from a knowledge of UTC one can always compute UT1 to better than 1 second accuracy.

## Leapseconds

When Greenwich UT1 midnight lags behind UTC midnight by more than 0.7 seconds the IERS will announce that a leap second will be added to the collection of UTC names. This leap second has traditionally been added after the last ``normal'' UTC name of December 31 or June 30. Thus when a UTC second is added the hours-minutes-seconds portion of the UTC name progresses as shown here

```
   ... DECEMBER 31 23:59:57
   ... DECEMBER 31 23:59:58
   ... DECEMBER 31 23:59:59
   ... DECEMBER 31 23:59:60
   ... JANUARY   1 00:00:00
```

instead of the usual progression

```
   ... DECEMBER 31 23:59:57
   ... DECEMBER 31 23:59:58
   ... DECEMBER 31 23:59:59
   ... JANUARY   1 00:00:00
```

Should Greenwich UT1 midnight run ahead of UTC midnight by more than 0.7 seconds the IERS will announce a negative leap second. In this case one of the usual UTC hours-minutes-seconds triples will be missing from the list of UTC names. In this case the progression will be:

```
   ... DECEMBER 31 23:59:57
   ... DECEMBER 31 23:59:58
   ... JANUARY   1 00:00:00
```

Since 1972 when leap seconds and the UTC system were introduced, a negative leap second has not occurred.

### The Leapseconds Kernel (LSK)

The primary difficulty with UTC strings is that it is not possible to predict which atomic times will correspond to times during a UTC leap second. Thus algorithms for converting between UTC and time systems that simply use a continuous set of numeric markers require knowledge of the location of leap seconds in the list of names. This is the purpose of the LEAPSECONDS kernel supplied with the Toolkit. To convert between UTC times and any other system, you must first load the leapseconds kernel via a call to the routine LDPOOL.

## Ephemeris Time (ET)

---

Ephemeris time is the uniform time scale represented by the independent variable in the differential equations that describe the motions of the planets, sun and moon. There are two forms of ephemeris time: Barycentric Dynamical Time (TDB) and Terrestrial Dynamical Time (TDT). Although they represent different time systems, these time systems are closely related.

### Barycentric Dynamical Time (TDB)

Barycentric dynamical time is used when describing the motion of bodies with respect to the solar system barycenter.

## Terrestrial Dynamical Time (TDT)

Terrestrial dynamical time is used when describing motions of objects near the earth. As far as measurements have been able to detect, TDT and TAI change at the same rate. Thus the difference between TDT and TAI is a constant. It is defined to be 32.184 seconds. At the zero point of TAI, TDT has a value of 32.184.

## The Relationship between TDT and TDB

TDB is believed to be in agreement with the time that would be kept by an atomic clock located at the solar system barycenter. A comparison of the times kept by a clock at the solar system barycenter with a TDB clock on earth would reveal that the two clocks are in close agreement but that they run at different rates at different times of the year. This is due to relativistic effects.

At some times in the year the TDT clock appears to run fast when compared to the TDB clock, at other times of the year it appears to run slow. Let TDB0 be some fixed epoch on the TDB clock and TDT0 be a fixed epoch on the TDT clock (TDB0 and TDT0 do not necessarily have to be the same epoch). Any epoch, EPOCH, can be represented in the following ways: as the number of seconds TDB(EPOCH), that have elapsed since TDB0 on the TDB clock; or as the number of seconds, TDT(EPOCH), that have elapsed since TDT0 on the TDT clock. If we plot the differences TDB(EPOCH) - TDT(EPOCH) against TDB(EPOCH) over all epochs, we will find that the graph is very close to a periodic function.

In SPICE the difference between TDT and TDB is computed as follows:

```
   [1]      TDB - TDT =  K * sin (E)
```

where K is a constant, and E is the eccentric anomaly of the heliocentric orbit of the Earth-Moon barycenter. This difference, which ignores small-period fluctuations, is accurate to about 0.000030 seconds. Thus to five decimal places the difference between TDT and TDB is a periodic function with magnitude approximately 0.001658 seconds and period equal to one sidereal year.

The eccentric anomaly E is given by

```
   [2]     E = M + EB sin (M)
```

where EB and M are the eccentricity and mean anomaly of the heliocentric orbit of the Earth-Moon barycenter. The mean anomaly is in turn given by

```
   [3]     M = M0 + M1*t
```

where t is the epoch TDB expressed in barycentric dynamical seconds past the epoch of J2000.

The values K, EB, M0, and M1 are retrieved from the kernel pool. These are part of the leapseconds kernel. They correspond to the ``kernel pool variables'' DELTET/K, DELTET/EB, and DELTET/M. The nominal values are:

```
   DELTET/K                =    1.657D-3
   DELTET/EB               =    1.671D-2
   DELTET/M                = (  6.239996D0   1.99096871D-7 )
```

### In the Toolkit ET Means TDB

When ephemeris time is called for by Toolkit routines, TDB is the implied time system. Software that converts between the various time systems described here use TDB whenever ephemeris time is called for. We call this time ET. (You can convert a UTC time string to TDT times, but you must make two subroutine calls instead of one.)

Ephemeris time is given in terms of seconds past a reference epoch. The reference epoch used throughout the Toolkit is the epoch J2000 (roughly noon on January 1, 2000). Using the Toolkit software, you can find out how many seconds the J2000 epoch is from right now.

## Naming the Seconds of Ephemeris Time

Although ephemeris time is a formal time, within the limits of measurements it coincides with atomic time. As such we should be able to relate it to the expressions of time that we use use everyday.

However, ephemeris time is described as a count of ephemeris seconds past the ephemeris reference epoch (J2000). For most of us the expression

```
   -312819349 seconds past the ephemeris epoch J2000
```

bears little relationship to the time system we use to organize our lives. For this reason, it is common to give names to the various ephemeris seconds in a manner analogous to the UTC naming of the seconds of TAI---as a calendar date and time of day. The above string corresponds to

```
   '1990 FEB 1 21:44:11 (TDB)'
```
There is an important distinction between the names given to ephemeris seconds and the names used by the UTC system. The names assigned to ephemeris times never have leap seconds. The `seconds' component of the name is restricted to and includes all values from 0 to 59.999... . Thus the time string above does not represent the same moment in time as does ``1990 FEB 1 21:44:11 (UTC)'' There are two reasons. First, ephemeris time is ahead of atomic time by 32.184 seconds. Second, when a leap second occurs UTC strings fit an extra name into the sequence of valid UTC names. Thus it appears that UTC names fall behind ET names by a second after each leapsecond. At the present time UTC time strings appear to be 62.184 seconds behind ET time strings. This appearance is due to the fact that the two naming conventions are not the same. They simply have a lot of names in common.

It is both fortunate and unfortunate that there is a huge set of common names between calendar dates ET and calendar dates UTC. Since there are relatively few leapseconds, a time given by an ET name is always close to the time in the UTC system having the same name. Thus for planning observations, you can know what day the observation will take place, whether or not you are likely to need a coat and how to arrange your daily activities around the observation. But for precise work you must pay attention to the difference between the two times systems. If in planning the observation of a stellar occultation by an asteroid the difference between the two naming systems is neglected, it is likely that the observation will be missed.

The routine STR2ET will convert an ephemeris calendar date to seconds past the ephemeris epoch J2000.

## Some Consequences of Leapseconds

There is no way of predicting when future leapseconds will occur. Normally you can predict whether there will be a leapsecond in the next few months, but beyond this predictions of leapseconds are not reliable. As a result we cannot say with certainty when a particular future UTC epoch will occur. For example, suppose you have a timer that you can set to ``beep'' after some number of seconds have passed. If this timer counts seconds perfectly without loosing or gaining time over decades, you cannot set it today to beep at midnight (00:00:00) January 1 (UTC) ten years from now---the number of leapseconds that will occur in the next ten years is not known. On the other hand, it is possible to set the timer so that it will beep at midnight January 1 (TDB). The TDB system does not have leapseconds. It is only necessary to know an algorithm (such as STR2ET) for converting calendar epochs TDB to seconds past some reference epoch in order to determine how to set the timer to beep at the correct epoch.

Any given Leapseconds Kernel will eventually become obsolete. Sometime after the creation of any Leapseconds Kernel there will be new leapseconds. When future leapseconds occur the old Leapseconds Kernel will no longer correctly describe the relationship between UTC, TDT and TDB for epochs that follow the new leapsecond. However, for epochs prior to the new leapsecond, the old kernel will always correctly describe the relationship between UTC, TDT and TDB.

## Computing UTC from TDB (DELTET)

---

Below are a few epochs printed out in calendar format in both the TDT and UTC time systems.

```
   1996, Oct 11, 12:01:02.1840  (TDT)
   1996, Oct 11, 12:00:00.0000  (UTC)

   1996, Oct 12, 12:01:02.1840  (TDT)
   1996, Oct 12, 12:00:00.0000  (UTC)

   1996, Oct 13, 12:01:02.1840  (TDT)
   1996, Oct 13, 12:00:00.0000  (UTC)

   1996, Oct 14, 12:01:02.1840  (TDT)
   1996, Oct 14, 12:00:00.0000  (UTC)

   1996, Oct 15, 12:01:02.1840  (TDT)
   1996, Oct 15, 12:00:00.0000  (UTC)
```

At least in October 1996, it's clear that if you have either TDT or UTC you can construct the corresponding representation for the same epoch in the UTC or TDT system by simply subtracting or adding 62.184 seconds.

If you don't worry about what happens during a leapsecond you can express the above idea as:

```
   [4]           DeltaTDT =  TDT - UTC
```

For all epochs except during UTC leapseconds the above expression makes sense. DeltaTDT is simply a step function increasing by one after each leapsecond. Thus DeltaTDT can be viewed as a step function of either UTC or TDT.

If you rearrange this expression, you can get

```
   [5]           UTC = TDT - DeltaTDT
```

Since, TDT can be expressed as seconds past J2000 (TDT), the above expression indicates the UTC can be expressed as some count of seconds. This representation is referred to by the dubious name of ``UTC seconds past J2000." If you write down the UTC calendar time string corresponding to an epoch and count the number of seconds between that calendar expression and the UTC calendar expression ``January 1, 2000 12:00:00'' and ignore leapseconds, you get the value of UTC in the expression above.

In practice this expression is broken down as follows:

```
   [6]           UTC  =  TDT - DeltaTA - DeltaAT
```
where

```
                 DeltaTA =  (TDT - TAI)
```
and

```
                 DeltaAT =  DeltaTDT - DeltaTA
```
The value DeltaTA is a constant, its value is nominally 32.184 seconds. DeltaTA is a step function. These two variables appear in the leapseconds kernel.

If we combine equation [6] above with equation [1] from the section ``The Relationship between TDT and TDB'' we get the following expression

```
   [7]           TDB - UTC =  DeltaTA + DeltaAT + K*sin(E)
```
This last value is called DeltaET and is computed by the SPICE routine DELTET. The various values that are used in the computation of DeltaET are contained in the Leapseconds Kernel. Indeed, a Leapseconds Kernel consists of precisely the information needed to compute DeltaET. Below is a sample Leapseconds kernel.

```
   \begindata

   DELTET/DELTA_T_A       =   32.184
   DELTET/K               =    1.657D-3
   DELTET/EB              =    1.671D-2
   DELTET/M               = (  6.239996D0   1.99096871D-7 )

   DELTET/DELTA_AT        = ( 10,   @1972-JAN-1
                              11,   @1972-JUL-1
                              12,   @1973-JAN-1
                              13,   @1974-JAN-1
                              14,   @1975-JAN-1
                              15,   @1976-JAN-1
                              16,   @1977-JAN-1
                              17,   @1978-JAN-1
                              18,   @1979-JAN-1
                              19,   @1980-JAN-1
                              20,   @1981-JUL-1
                              21,   @1982-JUL-1
                              22,   @1983-JUL-1
                              23,   @1985-JUL-1
                              24,   @1988-JAN-1  )

   \begintext
   DELTET/DELTA_T_A  corresponds to DeltaTA in equation [7].
   DELTET/K          corresponds to K in equation [7].
   DELTET/EB         corresponds to EB in equation [2].
   DELTET/M          corresponds to M0 and M1 of equation [3].
   DELTET/DELTA_AT   corresponds to DeltaAT of equation [7].
                     Note that this expression gives the
                     points on the UTC scale at which
                     DeltaAT changes.
```

Although NAIF recommends against it, you could modify this file to alter the conversion. For example, until 1985 JPL's Orbit Determination Program (ODP) set used a value of 32.1843817 for DeltaTA, and some older CRS tapes were created using this value in the conversion from TAI to TDT. The value returned by DELTET can be made compatible with these tapes by replacing the current value (32.184, exactly) with the older value. Also, JPL'S Optical Navigation Program (ONP) set does not use the periodic term (K sin E) of the difference TDB-TDT. Setting the value of K to zero eliminates this term.

## Problems With the Formulation of DeltaET

As we pointed out above, the expression ( TDT - UTC ) is meaningful as long as you stay away from leapseconds. If you write down the TDT and UTC representations for an epoch that occurs during a leapsecond you will have something like this:

```
   1996 Jan 01, 00:01:01.6840  (TDT)
   1996 Dec 31, 23:59:60.5000  (UTC)
```

Given these two epochs, it is no longer clear what we should assign to the value TDT - UTC. Thus although equation [7] above provides a simple expression for computing the ``difference between UTC and TDB'', the expression fails to tell us how to convert between TDB (or TDT) and UTC during leapseconds. For this reason the SPICE system does not use DeltaET when converting between TDB (or TDT) and UTC. Instead, the table of offsets corresponding to DeltaAT in the leapseconds kernel is converted to an equivalent table as shown below.

```
   Day Number of 1971-DEC-31     TAI seconds past 2000 at
                                 beginning of 1971-DEC-31

   Day Number of 1972-JAN-01     TAI seconds past 2000 at
                                 beginning of 1972-JAN-01

   Day Number of 1972-JUN-30     TAI seconds past 2000 at
                                 beginning of 1972-JUN-30

   Day Number of 1972-JUL-01     TAI seconds past 2000 at
                                 beginning of 1972-JUL-01

   Day Number of 1972-DEC-31     TAI seconds past 2000 at
                                 beginning of 1972-DEC-31

   Day Number of 1973-JAN-01     TAI seconds past 2000 at
                                 beginning of 1973-JAN-01
   Day Number of 1973-DEC-31     TAI seconds past 2000 at
                                 beginning of 1973-DEC-31
              .                           .
              .                           .
              .                           .
```
where the day number associated with a particular calendar date is the integer number of days that have passed since Jan 01, 0001 A.D. (on the extended Gregorian Calendar).

Given an epoch to be converted between UTC and some other time system (call this other system `S'), we decompose the conversion problem into two parts:

1. converting between UTC and TAI,

2. converting between TAI and system S.

To convert between TAI and UTC, we examine the above table to determine whether or not the epoch in question falls on a day containing a leapsecond or during a day that is 86400 seconds in length. Once the length of the day associated with the epoch has been determined, the conversion from UTC to TAI (or from TAI to UTC) is straight forward. (See the routine TTRANS for details.) Having settled the problem of converting between TAI and UTC, the conversion between TAI and system S is carried out using the analytic expressions (equations [1], [2] and [3]) given above.

## Spacecraft Clock (SCLK)

Most spacecraft have an onboard clock. This clock controls the times at which various actions are performed by the spacecraft and its science instruments. Observations are usually tagged with the spacecraft clock time when the observations are taken.

Each spacecraft clock can be constructed differently. For Galileo the SPICE spacecraft clock times looks like

```
   p/rrrrrrrr:mm:t:e

   p - partition number
   r - rim counts
   m - minor frame
   t - real time interrupt
   e - mod eight count
```
When asking for the matrix which describes the pointing for some structure or instrument used to perform an observation, you will usually request this information by supplying the spacecraft

clock string that was used to tag the observation. This string must usually be related to UTC or ET. Consequently it is necessary to load a file of ``spacecraft clock coefficients'' that enables SPICE software to transform the spacecraft clock string into one of the other time systems. This file of spacecraft clock coefficients is loaded with the routine LDPOOL.

A more detailed discussion of Spacecraft Clock is contained in the Required Reading file SCLK.REQ that is included with the SPICE Toolkit.

## Julian Date

---

The Julian date system is a numerical time system that allows you to easily compute the number of days between two epochs. NAIF recognizes two types of Julian dates. Julian Ephemeris Date (JED) and Julian Date UTC (JDUTC). As with calendar dates used for ephemeris time and calendar dates UTC, the distinction between the two systems is important. The names of the two systems overlap, but they correspond to different moments of time.

Julian Ephemeris Date is computed directly from ET via the formula

```
   JED(ET)  = J2000() + ET/SPD()
```
where J2000 is a constant function that returns the Julian Ephemeris Date of the reference epoch for ET, and SPD is a constant function that gives the number or seconds per day.

Julian Date UTC has an integer value whenever the corresponding UTC time is noon.

We recommend against using the JDUTC system as it provides no mechanism for talking about events that might occur during a leapsecond. All of the other time systems discussed can be used to refer to events occurring during a leap second.

### The abbreviation JD

Julian date is often abbreviated as ``JD.'' Unfortunately, the meaning of this string depends upon context. For example, the SPICE routine UTC2ET treats the string ``2451821.1928 JD'' as Julian Date UTC. On the other hand, the SPICE routine TPARSE treats the same string as Julian Date TDB. Consequently, for high accuracy work, you must be sure of the context when using strings labelled in this way. Unless context is clear, it's usually safer to label Julian Date strings with one of the unambiguous labels: JDUTC, JDTDB, or JDTDT.

# Appendix B. Parsing Time Strings

This appendix gives a detailed account of how the routine TPARTV parses time strings. TPARTV is the ``foundation'' routine relied upon by STR2ET, UTC2ET, TPARSE and TPICTR to accomplish the task of analyzing and assigning meaning to the components of a time string.

This appendix is not for everyone. Unless you need to understand in great detail how parsing of strings is performed, you can safely skip this appendix. The discussion below is quite technical and mirrors very closely the code in TPARTV that handles the parsing of time strings.

## An Outline of the Parser

The first step in processing a time string is to scan it from left to right identifying various substrings. If a substring is encountered that cannot be identified, attempts to further process the string are abandoned.

Having identified the components in the string as integers, months, weekdays, time systems, etc. An internal representation of the string is constructed. This representation is simply a list of the identified substrings in the order they are encountered. Each item in the list is called a token.

Working with the list of tokens, various rules are applied to remove some tokens and combine others into new tokens. The process of combination and removal of tokens continues until all tokens belong to a special set of ``meaningful'' tokens or until no further combinations and removals can be performed. If processing stops before all tokens are meaningful, a diagnostic message is created and the string is regarded as un-parsable. If all of the tokens are meaningful, a compatibility check is performed on the tokens to make sure that they unambiguously specify an epoch.

Once it is clear that an unambiguous epoch has been specified, the substrings corresponding to the meaningful tokens are converted into numeric representations or are noted so that the time conversion software can properly interpret the numeric components.

Almost all of the work of manipulating tokens is carried out by SPICE private routines. These routines are not considered part of the SPICE public interface. Feel free to read and copy these routines. However, we strongly recommend that you not call these routines in your own code since we do not guarantee backward compatibility of these routines in future releases of the Toolkit.

## Tokenizing the Input String

---

The first step in parsing a time string is to decompose it into recognizable substring components. This decomposition is done as follows:

Starting with the next unexamined character (on the first pass this is the first character in the string), scan from left to right looking for one of the following classes of substrings:

1. a maximal sequence of digits forming an unsigned integer.
2. a maximal sequence of space characters
3. a tab character
4. a weekday (or abbreviation of a weekday of at least 3 letters)
5. a month name (or abbreviation of a month name of at least 3 letters)
6. a time zone ( standard U.S. abbreviations)
7. a positive UTC offset specifier ( `UTC+' )
8. a negative UTC offset specifier ( `UTC-' )
9. a time system (TDT, TDB, UTC)
10. an era specifier ( `A.D.', `B.C.', `AD', `BC' )
11. a 12-hour clock specifier ( `A.M.', `P.M.', `AM', `PM' )
12. a Julian date specifier ( `JD' )

13. a day of year specifier ( `::' or `//' )

14. a period `.'

15. a dash `-'

16. a slash `/'

17. a colon `:'

18. a left parenthesis `('

19. a right parenthesis `)'

20. a single quote character (')

Once the next substring has been identified, its boundaries and classification are stored in the next available location in the buffer reserved for the tokenized representation of the time string.

The steps above are then repeated until the entire substring has been tokenized or a failure to recognize some substring occurs. If a failure occurs the location in the string is noted and a diagnostic message is created indicating the failure in the attempt to parse the string.

When the tokenization is finished, there will be a list of tokens from which a string can be constructed that lists the class of each token. Each class of token is represented by a single character. By placing these characters in a string a simple list of token classes is maintained. The characters used for the remainder of this discussion are listed below.

```
   Q  stands for the quote character
   [  stands for the left parenthesis character
   ]  stands for the right parenthesis character
   ,  stands for the comma character
   -  stands for the dash character
   .  stands for the decimal point character
   /  stands for the slash character
   :  stands for the colon character
   N  stands for one of the symbols A.M. or P.M.
   O  stands for the symbol UTC+
   Z  stands for a time zone such as PDT, PSD, CDT,
   b  stands for a block of white space (spaces or tabs)
   d  stands for the day of year marker (// or ::)
   e  stands for the era (B.C. or A.D.)
   j  stands for Julian date
   m  stands for a month
   o  stands for the symbol UTC-
   s  stands for a time system (UTC, TDT, TDB)
   t  stands the ISO date-T-time separator
   w  stands for the day of the week
   i  stands for a sequence of digits
```

Thus the list of token classifications corresponding to

```
   '1995 Jan 12 12:28:28'
```
will be

```
   'ibmbibi:i:i'
```

## Combining and Removing Tokens

---

Once an internal tokenized representation of the time string has been created, the internal representation is manipulated so that the meaning of the tokens is gradually discovered.

There are 3 basic operations that can be performed on the tokenized representation:

1. A token can be ``removed'' from the representation based on its classification. This removal can be wholesale as in ``remove all tokens corresponding to the blank character'', or it can be positional as in ``remove the last token classified as a blank.''

2. A sequence of tokens can be combined into a single new token with a potentially new classification. For example you might have a subsequence of token classifications such as `i.i' in the tokenized representation that corresponds to an unsigned integer, a period, and another unsigned integer. Under suitable circumstances this sequence `i.i' might be replaced by `n' (for number).

3. A single token can be reclassified. For example you might have a token whose classification is `i' for `unsigned integer' and have it reclassified as an hour `H'

## Initial Token Processing

---

The first phase of processing the tokenized time discovers any UTC offsets in the input string, abbreviated months, decimal numbers, and removes white space. The process proceeds as follows:

1. Token sequences that represent UTC time offsets are combined to form a single token with a new classification. (The character used for this new kind of token is `Z'.)

2. Months or weekdays that are followed by a period are combined to form a single token (month or weekday respectively). The motivation for this combination is to allow abbreviations such as ``Jan." It also allows strings such as ``January."

3. The right most sequence of tokens of the form ``i.i", (integer-period-integer) or ``i." (integer-period) is combined to form a single token ``n" (number). This combination is performed only once in the token resolution process.

4. All blanks (``b") are removed from the tokenization.

## Julian Dates

The string is now examined to see if the Julian date specifier `JD' is present. If so the following operations are performed. If no Julian date specifier is present, the steps below are skipped and processing resumes under the section ``Calendar Dates."

1. Any token sequence of the form `[s]' ( left parenthesis - time system - right parenthesis) is transformed to the sequence `*s*'. The `*' token is then removed. This leaves just the time system (TDT, TDB, or UTC) specification in the tokenization.

-- Note: Whenever a character in the token classification is replaced by `*', the next step is to remove all tokens classified as `*' from the token list. In the remainder of the discussion, we will not add the sentence describing the removal of all asterisks. It will be implicit that the asterisk is always removed after it is placed in the token list.

2. If the token sequence `[j]' (left parenthesis - Julian date specifier - right parenthesis) is present, it is replaced by `*j*'

3. If no number token, `n', (see above) is present in the tokenization, the left most integer (`i') is reclassified as a number ( `n' ).

4. If the token sequence `-n' ( dash - number ) appears in the token list, it is combined and classified as a number (`n'). This allows for the input of negative Julian dates.

5. The Julian date specifier `j' is noted and removed from the token list.

6. Any system token (`s') present in the token list is noted and removed.

7. The numeric components of the string are converted to double precision values and the token list is checked for unresolved tokens. (The only thing that should be in the token list at this point is a single numeric token.)

8. The parsing process halts. Either the string was successfully parsed and a double precision value for the Julian date has been constructed or there were unresolved tokens in the token list and a diagnostic message has been created.

## Calendar Dates

If the Julian date specifier was not present in the token list, we assume that the string and token list represents some calendar date format. One consequence of this assumption is that the dash `-' is now assumed to be just a punctuation mark and not part of some number. ISO formats are given first priority in the scheme of token resolution. Note that ISO formats do not allow the inclusion of time systems, time zones, eras, or 12-hour clocks.

Any integer class tokens (`i') whose corresponding substrings represent integers greater than or equal to 1000 are reclassified as years (`Y').

## ISO Formats

If the ISO separator token `T' is present, the string is treated as an ISO format string. If the token list matches one of the token patterns in the left column it is transformed to the corresponding item in the right column by removing punctuation and making the indicated transformations.

```
Y-i-iT ........ YmD
Y-i-iTi ....... YmDH
Y-i-iTi:i ..... YmDHM
Y-i-iTi:i:i ... YmDHMS
Y-i-iTi:i:n ... YmDHMS
Y-i-iTi:n ..... YmDHM
Y-i-iTn ....... YmDH
Y-iT .......... Yy
Y-iTi ......... YyH
Y-iTi:i ....... YyHM
Y-iTi:i:i ..... YyHMS
Y-iTi:i:n ..... YyHMS
Y-iTi:n ....... YyHM
Y-iTn ......... YyH
i-i-iT ........ YmD
i-i-iTi ....... YmDH
i-i-iTi:i ..... YmDHM
i-i-iTi:i:i ... YmDHMS
i-i-iTi:i:n ... YmDHMS
i-i-iTi:n ..... YmDHM
```

```
      i-i-iTn ....... YmDH
      i-iT .......... Yy
      i-iTi ......... YyH
      i-iTi:i ....... YyHM
      i-iTi:i:i ..... YyHMS
      i-iTi:i:n ..... YyHMS
      i-iTi:n ....... YyHM
      i-iTn ......... YyH

                      Y  ---  Year
                      m  ---  Month
                      D  ---  Day of Month
                      y  ---  Day of Year
                      H  ---  Hour
                      M  ---  Minute
                      S  ---  Second
```

If the token list contains the ISO separator (`T') but the list does not match one of the patters shown above, the input string is regarded as erroneous.

## Other Calendar Formats

If the ISO separator is not part of the token list, we next do what we can to recognize years and note the presence of modifiers (time zone specification, era, 12-hour clock etc.)

1. If a two digit integer is preceded by the quote character ('), the pair of tokens is combined to a single token and reclassified as a year.

2. The following token transformations are performed:

```
   '[e]'  ---> '*e*' (parenthesized era to era)
   '[w]'  ---> '*w*' (parenthesized weekday to weekday)
   '[N]'  ---> '*N*' (parenthesized AM/PM   to AM/PM)
   '[Z]'  ---> '*Z*' (parenthesized time zone to time zone)
   '[s]'  ---> '*s*' (parenthesized time system to time system)
   'ie',  ---> 'Ye'  (integer-era  to Year-era)
```

3. Eras, weekdays, AM/PM, time zones, time systems are noted and removed from the token list.

4. The string is examined for redundant commas, dashes, slashes periods, etc. If any are found the string is regarded as erroneous.

## Built in Representations

Having processed the token list to this point, we check to see if what remains is one of those in a large set of immediately recognized token lists. The complete list is shown below. As in the case of ISO formats, the left item is the token list, the right item is the transformation after removing delimiters. Note that the letter `d' stands for a day-of-year delimiter ( `//' or `::' ).

| | | | |
|---|---|---|---|
| Y-i-it | YmD | i/i/ii:i:n | mDYHMS |
| Y-i-iti | YmDH | i/i/ii:n | mDYHM |
| Y-i-iti:i | YmDHM | i/i/ii:n | mDYHM |
| Y-i-iti:i:i | YmDHMS | i:i:ii-i-Y | HMSmDY |
| Y-i-iti:i:n | YmDHMS | i:i:ii/i/Y | HMSmDY |
| Y-i-iti:n | YmDHM | i:i:ii/i/i | HMSmDY |
| Y-i-itn | YmDH | i:i:iimY | HMSDmY |
| Y-i/ | Yy | i:i:imiY | HMSmDY |
| Y-i/i:i | YyHM | i:i:ni-i-Y | HMSmDY |
| Y-i/i:i:i | YyHMS | i:i:ni/i/Y | HMSmDY |
| Y-i/i:i:n | YyHMS | i:i:ni/i/i | HMSmDY |
| Y-i/i:n | YyHM | i:i:nimY | HMSDmY |
| Y-id | Yy | i:i:nmiY | HMSmDY |
| Y-idi:i | YyHM | i:ii-i-Y | HMmDY |
| Y-idi:i:i | YyHMS | i:ii/i/Y | HMmDY |
| Y-idi:i:n | YyHMS | i:ii/i/i | HMmDY |
| Y-idi:n | YyHM | i:iimY | HMDmY |
| Y-it | Yy | i:imiY | HMmDY |
| Y-iti | YyH | i:ni-i-Y | HMmDY |
| Y-iti:i | YyHM | i:ni/i/Y | HMmDY |
| Y-iti:i:i | YyHMS | i:ni/i/i | HMmDY |
| Y-iti:i:n | YyHMS | i:nimY | HMDmY |
| Y-iti:n | YyHM | i:nmiY | HMmDY |
| Y-itn | YyH | iYd | yY |
| Yid | Yy | iYdi:i | yYHM |
| Yidi:i | YyHM | iYdi:i:i | yYHMS |
| Yidi:i:i | YyHMS | iYdi:i:n | yYHMS |
| Yidi:i:n | YyHMS | iYdi:n | yYHM |
| Yidi:n | YyHM | iiY | mDY |
| Yii | YmD | iiYi | mDYH |
| Yiii | YmDH | iiYi:i | mDYHM |
| Yiii:i | YmDHM | iiYi:i:i | mDYHMS |
| Yiii:i:i | YmDHMS | iiYi:i:n | mDYHMS |
| Yiii:i:n | YmDHMS | iiYi:n | mDYHM |
| Yiii:n | YmDHM | iiYn | mDYH |
| Yiiii | YmDHM | iid | Yy |
| Yiiiii | YmDHMS | iidi:i | YyHM |
| Yiiiin | YmDHMS | iidi:i:i | YyHMS |
| Yiiin | YmDHM | iidi:i:n | YyHMS |
| Yiin | YmDH | iidi:n | YyHM |
| Yim | YDm | iim | YDm |
| Yimi | YDmH | iimi | YDmH |
| Yimi:i | YDmHM | iimi:i | YDmHM |
| Yimi:i:i | YDmHMS | iimi:i:i | YDmHMS |
| Yimi:i:n | YDmHMS | iimi:i:n | YDmHMS |
| Yimi:n | YDmHM | iimi:n | YDmHM |
| Yimn | YDmH | iimii | YDmHM |

| | |
|---|---|
| Yin............ | YmD |
| Ymi............ | YmD |
| Ymii........... | YmDH |
| Ymii:i......... | YmDHM |
| Ymii:i:i....... | YmDHMS |
| Ymii:i:n....... | YmDHMS |
| Ymii:n......... | YmDHM |
| Ymin........... | YmDH |
| Ymn............ | YmD |
| Ynm............ | YDm |
| i-Y/........... | yY |
| i-Y/i:i........ | yYHM |
| i-Y/i:i:i...... | yYHMS |
| i-Y/i:i:n...... | yYHMS |
| i-Y/i:n........ | yYHM |
| i-Yd........... | yY |
| i-Ydi:i........ | yYHM |
| i-Ydi:i:i...... | yYHMS |
| i-Ydi:i:n...... | yYHMS |
| i-Ydi:n........ | yYHM |
| i-i-Y.......... | mDY |
| i-i-Yi:i....... | mDYHM |
| i-i-Yi:i:i..... | mDYHMS |
| i-i-Yi:i:n..... | mDYHMS |
| i-i-Yi:n....... | mDYHM |
| i-i-it......... | YmD |
| i-i-iti........ | YmDH |
| i-i-iti:i...... | YmDHM |
| i-i-iti:i:i.... | YmDHMS |
| i-i-iti:i:n.... | YmDHMS |
| i-i-iti:n...... | YmDHM |
| i-i-itn........ | YmDH |
| i-i/i:i........ | YyHM |
| i-i/i:i:i...... | YyHMS |
| i-i/i:i:n...... | YyHMS |
| i-i/i:n........ | YyHM |
| i-idi:i........ | YyHM |
| i-idi:i:i...... | YyHMS |
| i-idi:i:n...... | YyHMS |
| i-idi:n........ | YyHM |
| i-it........... | Yy |
| i-iti.......... | YyH |
| i-iti:i........ | YyHM |
| i-iti:i:i...... | YyHMS |
| i-iti:i:n...... | YyHMS |
| i-iti:n........ | YyHM |
| i-itn.......... | YyH |
| i/i/Y.......... | mDY |
| i/i/Y/i:n...... | mDYHM |
| i/i/Yi:i....... | mDYHM |
| i/i/Yi:i:i..... | mDYHMS |
| i/i/Yi:i:n..... | mDYHMS |
| i/i/i.......... | mDY |
| i/i/ii:i....... | mDYHM |
| i/i/ii:i:i..... | mDYHMS |
| iimiii......... | YDmHMS |
| iimiin......... | YDmHMS |
| iimin.......... | YDmHM |
| iimn........... | YDmH |
| imY............ | DmY |
| imYi........... | DmYH |
| imYi:i......... | DmYHM |
| imYi:i:i....... | DmYHMS |
| imYi:i:n....... | DmYHMS |
| imYi:n......... | DmYHM |
| imYn........... | DmYH |
| imi............ | YmD |
| imi:i:iY....... | DmHMSY |
| imi:i:nY....... | DmHMSY |
| imi:iY......... | DmHMY |
| imi:nY......... | DmHMY |
| imii........... | YmDH |
| imii:i......... | YmDHM |
| imii:i:i....... | YmDHMS |
| imii:i:n....... | YmDHMS |
| imii:n......... | YmDHM |
| imiii.......... | YmDHM |
| imiiii......... | YmDHMS |
| imiiin......... | YmDHMS |
| imiin.......... | YmDHM |
| imin........... | YmDH |
| imn............ | YmD |
| inY............ | mDY |
| inm............ | YDm |
| miY............ | mDY |
| miYi........... | mDYH |
| miYi:i......... | mDYHM |
| miYi:i:i....... | mDYHMS |
| miYi:i:n....... | mDYHMS |
| miYi:n......... | mDYHM |
| miYn........... | mDYH |
| mii............ | mDY |
| mii:i:iY....... | mDHMSY |
| mii:i:nY....... | mDHMSY |
| mii:iY......... | mDHMY |
| mii:nY......... | mDHMY |
| miii........... | mDYH |
| miii:i......... | mDYHM |
| miii:i:i....... | mDYHMS |
| miii:i:n....... | mDYHMS |
| miii:n......... | mDYHM |
| miiii.......... | mDYHM |
| miiiii......... | mDYHMS |
| miiiin......... | mDYHMS |
| miiin.......... | mDYHM |
| miin........... | mDYH |
| mnY............ | mDY |
| mni............ | mDY |
| nmY............ | DmY |

If the token list agrees with one of the items in the above list, the double precision value corresponding to each token is computed and the parsing process halts with success.

## Last Resort Production Rules

If the token list did not match one of the built-in patterns above, several checks are performed to see if there is redundant information in the token list (duplicate time systems, eras, etc.) If any such duplicate items are located, the input string is diagnosed as erroneous.

Assuming that the error checks just discussed do not produce an error diagnosis, the string is processed according to the following rules:

1. Commas, dashes, and slashes are removed from the token list. The resulting token list is then compared once more against the list of token patterns above. If there is a successful match, the parsing process halts with success.

2. The following list of transformations are attempted in the order indicated.

```
'i:i:i:n'  ---> 'D*H*M*S' (days, hours, minutes, seconds)
'i:i:i:i'  ---> 'D*H*M*S' (days, hours, minutes, seconds)
'i:i:n'    ---> 'H*M*S'   (hours, minutes, seconds)
'i:i:i'    ---> 'H*M*S'   (hours, minutes, seconds)
'i:n'      ---> 'H*M'     (hours, minutes)
'i:i'      ---> 'H*M'     (hours, minutes)
```

3. All colons are removed from the token list.

4. The following list of transformations are attempted in the order indicated.

```
'<miiH' ---> 'mDY'  (month, day, year)
'<mi'   ---> 'mD'   (month, day)
'Siim>' ---> 'SYDm' (seconds, year, day, month)
'im>'   ---> 'Dm'   (day, month)
'miY>'  ---> 'mDY'  (month, day, year)
'Ymi'   ---> 'YmD'  (year, month, day)
'Smi'   ---> 'SmD'  (seconds, month, day)
'Mmi'   ---> 'MmD'  (minutes, month, day)
'imY'   ---> 'DmY'  (day, month, year)
'imH'   ---> 'DmH'  (day, month, hour)
'Yid'   ---> 'Yy*'  (year, day-of-year)
'iYd'   ---> 'yY*'  (day-of-year, year)
'Ydi'   ---> 'Y*y'  (year, day-of-year)
```

```
The characters '<' and '>' mean that the transformation is
performed only if the token list occurs at the beginning or
end respectively of the the token list.
```

5. The token list is now examined to determine whether any unresolved numeric tokens remain. If unresolved numeric tokens are present, the input string is diagnosed as erroneous. If no unresolved components remain, the token list is checked for consistency. For example there can be only one of each type of token, and there must be a sufficient number of tokens present to unambiguously determine the epoch.

## Conclusion

---

As can be surmised from the preceding discussion, it is very difficult to give a complete list of all token patterns that might yield a parsed time string. Nevertheless, we feel that the approach taken and the transformations applied will yield correct and consistent interpretations of the many ways people choose to represent time.