VMS Software

VSI OpenVMS

VSI DECnet-Plus for OpenVMS DECdts Programming

Operating System and Version: VSI OpenVMS IA-64 Version 8.4-1H1 or higher
VSI OpenVMS Alpha Version 8.4-2L1 or higher

VMS Software, Inc. (VSI)
Boston, Massachusetts, USA

VSI DECnet-Plus for OpenVMS DECdts Programming

VMS Software

Legal Notice

Table of Contents

Preface

The VSI Digital Distributed Time Service (DECdts) is a networkwide service that runs on OpenVMS and Digital UNIX operating systems. DECdts enables systems to synchronize their clocks with all the other system clocks in the network.

This manual provides reference information about the DECdts application programming interface (API) routines you can use to obtain, convert, and calculate time values. The software interface between DECdts and external time-provider programs is also described.

1. About VSI

VMS Software, Inc. (VSI) is an independent software company licensed by Hewlett Packard Enterprise to develop and support the OpenVMS operating system.

2. Intended Audience

This manual is intended for two audiences: programmers who use the DECdts API to obtain timestamps or convert time values, and programmers who write the software interface between time-provider (TP) hardware and DECdts. Both audiences should have a sound understanding of DECdts concepts and the programming interface before using the software.

3. Document Structure

This manual is organized into four chapters. The first chapter introduces DECdts API. The second chapter describes each DECdts portable routine. The third chapter contains a C programming example that shows a practical application

of the DECdts API programming routines, and the fourth chapter describes the DECdts time-provider interface (TPI) for DECdts software on OpenVMS and Digital UNIX operating systems.

4. Related Documents

For a list of additional documents that are available in support of this version of the operating system, refer to the *VSI DECnet-Plus for OpenVMS Introduction and User's Guide* or the *DECnet-Plus for VSI UNIX Introduction and User's Guide*.

5. OpenVMS Documentation

The full VSI OpenVMS documentation set can be found on the VMS Software Documentation webpage at https://docs.vmssoftware.com.

6. VSI Encourages Your Comments

You may send comments or suggestions regarding this manual or any VSI document by sending electronic mail to the following Internet address: <docinfo@vmssoftware.com>. Users who have VSI OpenVMS support contracts through VSI can contact <support@vmssoftware.com> for help with this product.

7. Conventions

The following conventions may be used in this manual:

Convention	Meaning
`monospace type`	Indicates a literal example of system output or user input. In text, indicates command names, keywords, node names, file names, directories, utilities, and tools. On a DECnet-Plus for OpenVMS or UNIX system, enter the word or phrase in the exact case shown. You can abbreviate command keywords to the smallest number of characters that OpenVMS, UNIX, NCL, DECdns, DECdts, and the other utilities accept, usually three characters.
italic	Indicates a variable.
bold	Indicates a new term or important text.
Return	Indicates that you press the Return key.
Crtl/*x*	Indicates that you press the Control key while you press the key noted by *x*.
[]	In command format descriptions, indicates optional elements. You can enter as many as you want.
{ }	In command format descriptions, indicates you must enter at least one listed element.

Chapter 1. Introduction to the DECdts API

This chapter describes the Digital Distributed Time Service (DECdts) programming routines. You can use these routines to obtain timestamps that are based on Coordinated Universal Time (UTC). You can also use the DECdts routines to translate among different timestamp formats and perform calculations on timestamps. Applications can use the timestamps that DECdts supplies to determine event sequencing, duration, and scheduling. Applications can call the DECdts routines from DECdts server or clerk systems.

The Digital Distributed Time Service routines are written in the C programming language. Be familiar with the basic DECdts concepts before you attempt to use the applications programming interface (API).

The DECdts API routines offer the following basic functions:

- Retrieve timestamp information
- Convert between binary timestamps that use different time structures
- Convert between binary timestamps and ASCII representations
- Convert between UTC time and local time
- Convert the binary time values in the OpenVMS (Smithsonian-based) format to or from UTC-based binary timestamps (OpenVMS systems only)
- Manipulate binary timestamps
- Compare two binary time values
- Calculate binary time values
- Obtain time zone information

The following sections describe DECdts time representations, DECdts time structures, API header files, and API routines.

1.1. DECdts Time Representation

UTC is the international time standard that has largely replaced Greenwich Mean Time (GMT). The standard is administered by the International Time Bureau (BIH) and is widely used. DECdts uses opaque binary timestamps that represent UTC for all of its internal processes. You cannot read or disassemble a DECdts binary timestamp; the DECdts API allows applications to convert or manipulate timestamps, but they cannot be displayed. DECdts also translates the binary timestamps into ASCII text strings, which can be displayed.

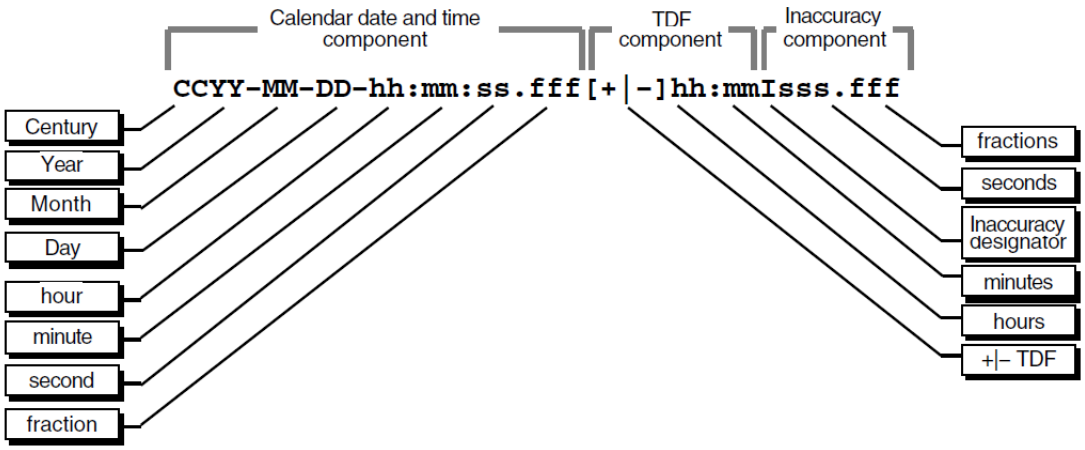
1.1.1. Absolute Time Representation

An **absolute time** is a point on a time scale. For DECdts, absolute times reference the UTC time scale; absolute time measurements are derived from system clocks or external time-providers. When DECdts reads a system clock time, it records the time in an opaque binary timestamp that also includes the inaccuracy and other information. When you display an absolute time, DECdts converts the time to ASCII text, as shown in the following display:

```
1996-11-21-13:30:25.785-04:00I000.082
```

DECdts displays all times in a format that complies with the International Standards Organization (ISO) 8601 (1988) standard. Note that the inaccuracy portion of the time is not defined in the ISO standard (times that do not include an inaccuracy are accepted). Figure 1–1 explains the ISO format that generated the previous display.

Figure 1.1. Time Display Format



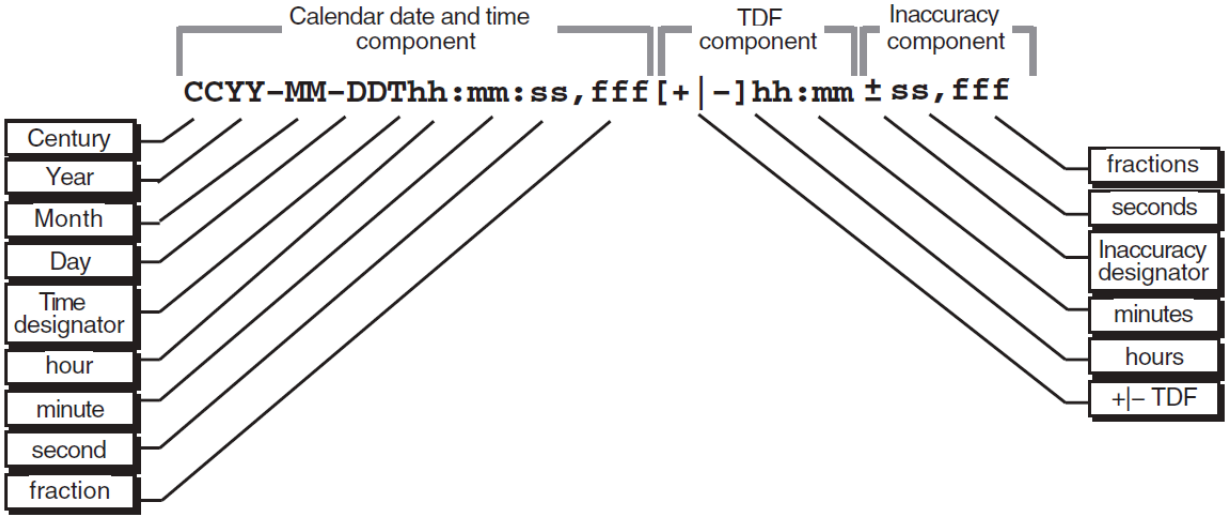
In Figure 1.1, the relative time preceded by the plus (+) or minus (-) character indicates the hours and minutes that the calendar date and time are offset from UTC. The presence of this **time differential factor** (TDF) in the string also indicates that the calendar date and time are the local time of the system, not UTC. Local time is UTC minus the TDF. The Inaccuracy designator I indicates the beginning of the inaccuracy component associated with the time.

Although DECdts displays all times in the previous format, variations in the ISO format shown in Figure 1–2 are also accepted as input for the ASCII conversion routines.

In Figure 1.2, the Time designator T separates the calendar date from the time, a comma separates seconds from fractional seconds, and the plus or minus character indicates the beginning of the inaccuracy component.

The following examples show some valid time formats.

Figure 1.2. Time Display Format Variants



The following represents July 4, 1776 17:01 GMT and an infinite inaccuracy (default).

```
1776-7-4-17:01:00
```

The following represents a local time of 12:01 (17:01 GMT) on July 4, 1776 with a TDF of -5 hours and an inaccuracy of 100 seconds.

```
1776-7-4-12:01:00-05:00I100
```

Both of the following represent 12:00 GMT in the current day, month, and year with an infinite inaccuracy.

```
12:00 and T12
```

The following represents July 14, 1792 00:00 GMT with an infinite inaccuracy.

```
1792-7-14
```

1.1.2. Relative Time Representation

A **relative time** is a discrete time interval that is usually added to or subtracted from another time. A TDF associated with an absolute time is one example of a relative time. A relative time is normally used as input for commands or system routines.

The following example shows a relative time of 21 days, 8 hours, and 30 minutes, 25 seconds with an inaccuracy of 0.300 second.

```
21-08:30:25.000I00.300
```

The following example shows a negative relative time of 20.2 seconds with an infinite inaccuracy (default).

```
-20.2
```

The following example shows a relative time of 10 minutes, 15.1 seconds with an inaccuracy of 4 seconds.

```
10:15.1I4
```

Figure 1.3. Relative Time Syntax

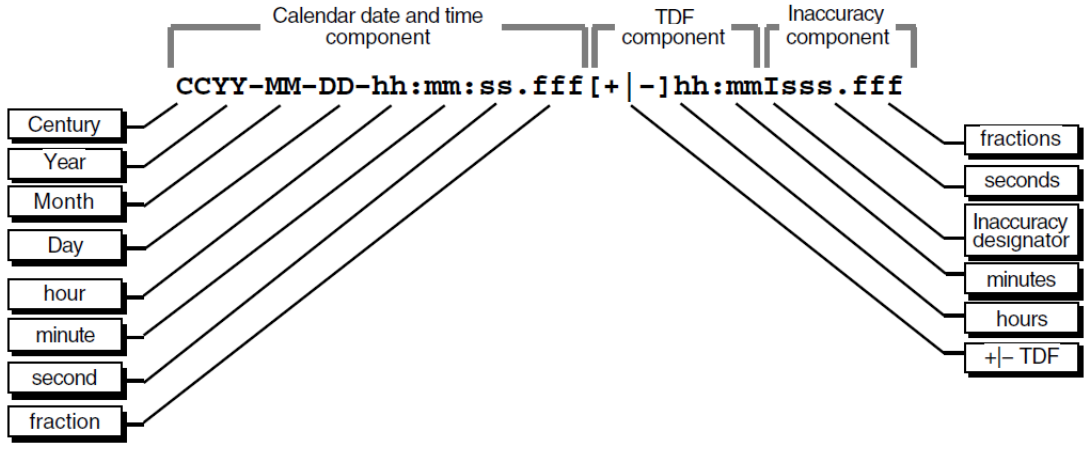


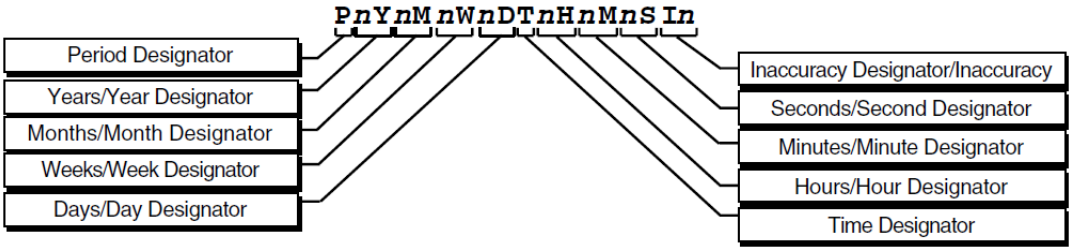
Figure 1.3 shows the full syntax for a relative time.

Notice that a relative time does not use the calendar date fields, because these fields concern absolute time. A positive relative time is unsigned; a negative relative time is preceded by a minus ( ) sign. A relative time is often subtracted from or added to another relative or absolute time. The relative times that DECdts uses internally are opaque binary timestamps. The DECdts API offers several routines that can be used to calculate new times using relative binary timestamps.

Representing Periods of Time

A given duration of a period of time can be represented by a data element of variable length that uses the syntax shown in Figure 1.4.

Figure 1.4. Time Period Syntax



The data element contains the following parts:

- The designator P precedes the part that includes the calendar components, including the following:
  - The number of years followed by the designator `Y`
  - The number of months followed by the designator `M`
  - The number of weeks followed by the designator `W`
  - The number of days followed by the designator `D`
- The designator T precedes the part that includes the time components, including the following:
  - The number of hours followed by the designator `H`
  - The number of minutes followed by the designator `M`
  - The number of seconds followed by the designator `S`
- The designator I precedes the number of seconds of inaccuracy.

The following example represents a period of 1 year, 6 months, 15 days, 11 hours, 30 minutes, and 30 seconds and an infinite inaccuracy.

```
P1Y6M15DT11H30M30S
```

The following example represents a period of 3 weeks and an inaccuracy of 4 seconds.

```
P3WI4
```

1.2. Time Structures

DECdts can convert between several types of binary time structures that are based on different base dates and time unit measurements. DECdts uses UTC- based time structures and can convert other types

of time structures to its own presentation of UTC-based time. The DECdts API routines are used to perform these conversions for applications on your system.

Table 1.1 lists the absolute time structures that the DECdts API uses to modify binary times for applications.

Table 1.1. Absolute Time Structures

Structure	Time Units	Base Date	Approximate Range
`utc`	100-nanosecond	15 October 1582	A.D. 1 to A.D. 30,000
`tm`	second	1 January 1900	A.D. 1 to A.D. 30,000
`timespec`	nanosecond	1 January 1970	A.D. 1970 to A.D. 2106

Table 1.2 lists the relative time structures that the DECdts API uses to modify binary times for applications.

Table 1.2. Relative Time Structures

Structure	Time Units	Approximate Range
`utc`	100-nanosecond	± 30,000 years
`tm`	second	± 30,000 years
`reltimespec`	nanosecond	± 68 years

The remainder of this section explains the DECdts time structures in detail.

1.2.1. The utc Structure

Coordinated Universal Time (UTC) is useful for measuring time across local time zones and for avoiding the seasonal changes (summer time or daylight saving time) that can affect the local time. DECdts uses 128-bit binary numbers to represent time values internally; throughout this manual, these binary numbers representing time values are referred to as **binary timestamps**. The DECdts utc structure determines the ordering of the bits in a binary timestamp; all binary timestamps that are based on the utc structure contain the following information:

- The count of 100-nanosecond units since 00:00:00.00, 15 October 1582 (the date of the Gregorian reform to the Christian calendar)
- The count of 100-nanosecond units of inaccuracy applied to the above
- The time differential factor (TDF), expressed as the signed quantity
- The timestamp version number

The binary timestamps that are derived from the DECdts utc structure have an opaque format. This format is a cryptic character sequence that DECdts uses and stores internally. The opaque binary timestamp is designed for use in programs, protocols, and databases.

Note

Applications use the opaque binary timestamps when storing time values or when passing them to DECdts.

The API provides the necessary routines for converting between opaque binary timestamps and character strings that can be displayed and read by users.

1.2.2. The tm Structure

The `tm` structure is based on the time in years, months, days, hours, minutes, and seconds since 00:00:00 GMT (Greenwich Mean Time), 1 January 1900. The `tm` structure is defined in the `<time.h>` header file.

The `tm` structure declaration follows:

```
struct tm {
               int tm_sec;     /* Seconds (0 - 59)                 */
               int tm_min;     /* Minutes (0 - 59)                 */
               int tm_hour;    /* Hours (0 - 23)                   */
               int tm_mday;    /* Day of Month (1 - 31)            */
               int tm_mon;     /* Month of Year (0 - 11)           */
               int tm_year;    /* Year - 1900                      */
               int tm_wday;    /* Day of Week (Sunday = 0)         */
               int tm_yday;    /* Day of Year (0 - 364)            */
               int tm_isdst;   /* Nonzero if Daylight Savings Time */
                               /* is in effect                     */
          };
```

Not all of the `tm` structure fields are used for each routine that converts between `tm` structures and `utc` structures. See the parameter descriptions that accompany the routines in Chapter 2 for additional information about which fields are used for specific routines.

1.2.3. The timespec Structure

The `timespec` structure is normally used in combination with or in place of the `tm` structure to provide finer resolution for binary times. The `timespec` structure is similar to the `tm` structure, but the `timespec` structure specifies the number of seconds and nanoseconds since the base time of 00:00:00 GMT, 1 January 1970. You can find the structure in the `<utc.h>` header file.

The `timespec` structure declaration follows:

```
struct timespec {
                 unsigned long tv_sec; /* Seconds since 00:00:00 GMT,  */
                                       /* 1 January 1970               */
                 long tv_nsec;         /* Additional nanoseconds since */
                                       /* tv_sec                       */
                }         timespec_t;
```

1.2.4. The reltimespec Structure

The `reltimespec` structure represents relative time. This structure is similar to the `timespec` structure, except that the first field is *signed* in the `reltimespec` structure. (The field is *unsigned* in the `timespec` structure.) You can find the `reltimespec` structure in the `<utc.h>` header file.

The `reltimespec` structure declaration follows:

```
struct reltimespec {
                     long tv_sec;   /* Seconds of relative time    */
                     long tv_nsec;  /* Additional nanoseconds of   */
                                    /* relative time               */
```

```
} reltimespec_t;
```

1.2.5. The OpenVMS Time Structure (OpenVMS only)

The OpenVMS time structure is based on Smithsonian time, which has a base date of November 17, 1858. The binary OpenVMS structure is a signed, 64-bit integer that has a positive value for absolute times. You can use the DECdts API to translate an OpenVMS structure representing an absolute time to or from the DECdts UTC-based binary timestamp.

1.3. DECdts API Header Files

The `<time.h>` and `<utc.h>` header files contain the data structures, type definitions, and define statements that are referenced by the DECdts API routines. The `<time.h>` header file is present on all OpenVMS systems and is a standard UNIX file as well. The `<utc.h>` header file includes `<time.h>` and contains the `timespec`, `reltimespec`, and `utc` structures.

On OpenVMS systems, the header files are located in the sys$library directory; on UNIX systems, these header files are located in `/usr/include`.

1.4. Linking Programs with the DECdts API

The DECdts API is implemented by a shared image (OpenVMS) or object library (UNIX); to use the API with your program, you must link the program with the shared image or object library. The procedure for linking a program differs according to the operating system running on a given node. The following sections describe how to link programs with the DECdts API on DECnet-Plus for OpenVMS and DECnet-Plus for UNIX systems.

1.4.1. Linking Programs on DECnet-Plus for OpenVMS systems (OpenVMS only)

On DECnet-Plus for OpenVMS systems, the DECdts API is implemented by the shared image sys $library:dtss$shr.exe. The following example shows how to link a program with the DECdts shared image:

```
$ cc myprogram.c/output=myprogram.obj
$ link myprogram.obj, sys$input:/options Return
sys$library:dtss$shr.exe/share Ctrl-z
$
```

1.4.2. Linking Programs on DECnet-Plus for UNIX systems (UNIX only)

On DECnet-Plus for UNIX systems, the DECdts API is implemented by the object library `usr/lib/libutc.a`. The following example shows how to link a program with the DECdts object library:

```
# cc myprogram.c -lutc -o myprogram
#
```

Chapter 2. DECdts Portable Applications Programming Interface

The Digital Distributed Time Service programming routines can obtain timestamps that are based on Coordinated Universal Time (UTC), translate between different timestamp formats, and perform calculations on timestamps. Applications can call the DECdts routines from DECdts server or clerk systems and use the timestamps that DECdts supplies to determine event sequencing, duration, and scheduling.

The DECdts routines can perform the following basic functions:

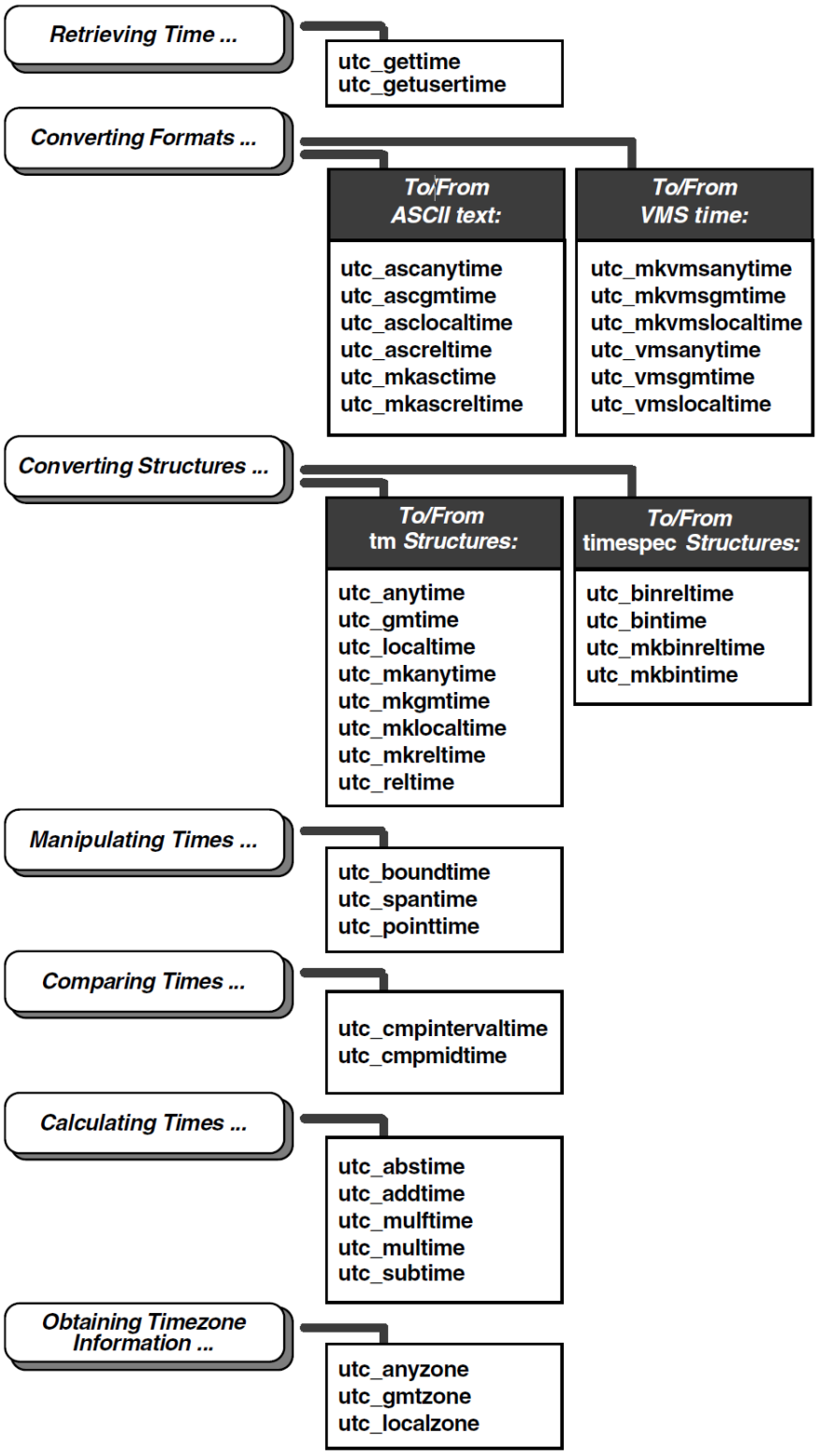
- Retrieve the current (UTC-based) time from DECdts.
- Convert binary timestamps expressed in the `utc` time structure to or from `tm` structure components.
- Convert the binary timestamps expressed in the `utc` time structure to or from `timespec` structure components.
- Convert the binary timestamps expressed in the utc time structure to or from ASCII strings.
- Convert the binary time values in the 64-bit OpenVMS (Smithsonian-based) format to or from UTC-based binary timestamps (OpenVMS systems only).
- Compare two binary time values.
- Calculate binary time values.
- Obtain time zone information.

DECdts can convert between several types of binary time structures that are based on different calendars and time unit measurements. DECdts uses UTC-based time structures and can convert other types of time structures to its own presentation of UTC-based time.

2.1. DECdts API Routine Functions

Figure 2.1 categorizes the DECdts portable interface routines by function.

Figure 2.1. DTS Portable Interface Categories



An alphabetical listing of the DECdts portable interface routines and a brief description of each one follows:

`utc_abstime`	Computes the absolute value of a binary relative time.

`utc_addtime`	Computes the sum of two binary timestamps; the timestamps can be two relative times or a relative time and an absolute time.
`utc_anytime`	Converts a binary timestamp into a `tm` structure, using the TDF information contained in the timestamp to determine the TDF returned with the `tm` structure.
`utc_anyzone`	Gets the time zone label and offset from GMT, using the TDF contained in the input `utc`.
`utc_ascanytime`	Converts a binary timestamp into an ASCII string that represents an arbitrary time zone.
`utc_ascgmtime`	Converts a binary timestamp into an ASCII string that expresses a GMT time.
`utc_asclocaltime`	Converts a binary timestamp to an ASCII string that represents a local time.
`utc_ascreltime`	Converts a binary timestamp that expresses a relative time to its ASCII representation.
`utc_binreltime`	Converts a relative binary timestamp into `timespec` structures that express relative time and inaccuracy.
`utc_bintime`	Converts a binary timestamp into a `timespec` structure.
`utc_boundtime`	Given two UTC times, one before and one after an event, returns a single UTC time whose inaccuracy includes the event.
`utc_cmpintervaltime`	Compares two binary timestamps or two relative binary timestamps.
`utc_cmpmidtime`	Compares two binary timestamps or two relative binary timestamps, ignoring inaccuracies.
`utc_gettime`	Returns the current system time and inaccuracy as an opaque binary timestamp.
`utc_getusertime`	Returns the time and process-specific TDF, rather than the system-specific TDF.
`utc_gmtime`	Converts a binary timestamp into a `tm` structure that expresses GMT or the equivalent UTC.
`utc_gmtzone`	Gets the time zone label and zero offset from GMT, given `utc`.
`utc_localtime`	Converts a binary timestamp into a `tm` structure that expresses local time.
`utc_localzone`	Gets the time zone label and offset from GMT, given `utc`.
`utc_mkanytime`	Converts a `tm` structure and TDF (expressing the time in an arbitrary time zone) into a binary timestamp.

`utc_mkascreltime`	Converts a null-terminated character string, which represents a relative timestamp to a binary timestamp.
`utc_mkasctime`	Converts a null-terminated character string, which represents an absolute timestamp, to a binary timestamp.
`utc_mkbinreltime`	Converts a `timespec` structure expressing a relative time to a binary timestamp.
`utc_mkbintime`	Converts a `timespec` structure into a binary timestamp.
`utc_mkgmtime`	Converts a `tm` structure that expresses GMT or UTC to a binary timestamp.
`utc_mklocaltime`	Converts a `tm` structure that expresses local time to a binary timestamp.
`utc_mkreltime`	Converts a `tm` structure that expresses relative time to a binary timestamp.
`utc_mkvmsanytime`	Converts a binary OpenVMS format time and TDF (expressing the time in an arbitrary time zone) to a binary timestamp.
`utc_mkvmsgmtime`	Converts a binary OpenVMS format time expressing GMT (or the equivalent UTC) into a binary timestamp.
`utc_mkvmslocaltime`	Converts a local binary OpenVMS format time to a binary timestamp, using the host system's TDF.
`utc_mulftime`	Multiplies a relative binary timestamp by a floating-point value.
`utc_multime`	Multiplies a relative binary timestamp by an integer factor.
`utc_pointtime`	Converts a binary timestamp to three binary timestamps that represent the earliest, most likely, and latest time.
`utc_reltime`	Converts a binary timestamp that expresses a relative time into a `tm` structure.
`utc_spantime`	Given two (possibly unordered) UTC timestamps, returns a single UTC time interval whose inaccuracy spans the two input timestamps.
`utc_subtime`	Computes the difference between two binary timestamps that express two relative times (an absolute time and a relative time, two relative times, or two absolute times).
`utc_vmsanytime`	Converts a binary timestamp to a binary OpenVMS-format time, using the TDF contained in the binary timestamp.
`utc_vmsgmtime`	Converts a binary timestamp to a binary OpenVMS-format time expressing GMT or the equivalent UTC.

`utc_vmslocaltime`	Converts a binary timestamp to a local binary OpenVMS format time, using the host system's time differential factor.

Absolute time is a point on a time scale; absolute time measurements are derived from system clocks or external time-providers. For DECdts, absolute times reference the UTC standard and include the inaccuracy and other information. When you display an absolute time, DECdts converts the time to ASCII text, as shown in the following display:

```
1996-11-21-13:30:25.785-04:00I000.082
```

Relative time is a discrete time interval that is usually added to or subtracted from an absolute time. A time differential factor (TDF) associated with an absolute time is one example of a relative time. Note that a relative time does not use the calendar date fields, because these fields concern absolute time.

Coordinated Universal Time (UTC) is the international time standard that DECdts uses. The zero hour of UTC is based on the zero hour of Greenwich Mean Time (GMT). The documentation consistently refers to the time zone of the Greenwich Meridian as GMT. However, this time zone is also sometimes referred to as UTC.

The **time differential factor** (TDF) is the difference between UTC and the time in a particular time zone.

The user's environment determines the time zone rule (details are system dependent):

- **OpenVMS:** The user selects a time zone by defining `sys$timezone_rule` during the `sys$manager:net$configure.com` procedure, or by explicitly defining `sys$timezone_rule`. See the DECdts section of the configuration guide for information on how to construct a time zone rule.
- **UNIX:** The user selects a time zone by specifying the time zone environment variable. (The reference page for the `utc_localtime( )` system call provides additional information.)

If the user's environment does not specify a time zone rule, the system's rule is used (details of the rule are system dependent):

- **OpenVMS:** OpenVMS systems do not have a default time zone rule. You must run the `sys$manager:net$configure` procedure to specify a time zone.
- **UNIX:** The rule in `/etc/zoneinfo/localtime` applies.

Chapter 1 provides additional information about UTC and GMT, TDFs and time zones, and relative and absolute times.

Unless otherwise specified, the default input and output parameters are as follows:

- If `utc` is not specified as an input parameter, the current time is used.
- If `inacc` is not specified as an input parameter, infinity is used.
- If no output parameter is specified, no result (or an error) is returned.

The following section is a command reference, which includes all DECdts API routines.

utc_abstime

utc_abstime — Computes the absolute value of a relative binary timestamp.

Syntax

```
#include <utc.h>

int utc_abstime(result, *utc1)

  utc_t result;
  const utc_t *utc1;
```

Parameters

Input

utc1

Relative binary timestamp.

Output

result

Absolute value of the input relative binary timestamp.

Description

The **Absolute Time** routine computes the absolute value of a relative binary timestamp. The input timestamp represents a relative (delta) time.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time parameter or invalid results.

Examples

The following example scales a relative time, computes its absolute value, and prints the result.

```
utc_t       relutc, scaledutc;
char        timstr[UTC_MAX_STR_LEN];

/*
 *   Make sure relative timestamp represents a positive interval...
 */

utc_abstime(&relutc,            /* Out: Abs-value of rel time  */
            &relutc);           /* In:  Relative time to scale */

/*
 *   Scale it by a factor of 17...
 */

utc_multime(&scaledutc,         /* Out: Scaled relative time   */
            &relutc,            /* In:  Relative time to scale */
            17L);               /* In:  Scale factor           */
```

```
utc_ascreltime(timstr,          /* Out: ASCII relative time    */
               UTC_MAX_STR_LEN, /* In:  Length of input string */
               &scaledutc);     /* In:  Relative time to       */
                                /*      convert                */

printf("%s\n",timstr);

/*
 *   Scale it by a factor of 17.65...
 */

utc_mulftime(&scaledutc,        /* Out: Scaled relative time   */
             &relutc,           /* In:  Relative time to scale */
             17.65);            /* In:  Scale factor           */

utc_ascreltime(timstr,          /* Out: ASCII relative time    */
               UTC_MAX_STR_LEN, /* In:  Length of input string */
               &scaledutc);     /* In:  Relative time to       */
                                /*      convert                */

printf("%s\n",timstr);
```

utc_addtime

utc_addtime — Computes the sum of two binary timestamps; the timestamps can be two relative times or a relative time and an absolute time.

Syntax

```
#include <utc.h>

int utc_addtime(result, *utc1, *utc2)

  utc_t result;
  const utc_t *utc1;
  const utc_t *utc2;
```

Parameters

Input

utc1

Binary timestamp or relative binary timestamp.

utc2

Binary timestamp or relative binary timestamp.

Output

result

Resulting binary timestamp or relative binary timestamp, depending on the operation performed:

- *relative time* + *relative time* = **relative time**

- *absolute time* + *relative time* = **absolute time**
- *relative time* + *absolute time* = **absolute time**
- *absolute time* + *absolute time* is undefined. See **NOTES**.

Description

The **Add Time** routine adds two binary timestamps, producing a third binary timestamp whose inaccuracy is the sum of the two input inaccuracies. One or both of the input timestamps typically represent a relative (delta) time. The TDF in the first input timestamp is copied to the output.

Notes

Although no error is returned, do **not** use the combination *absolute time* + *absolute time*

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time parameter or invalid results.

Example

The following example shows how to compute a timestamp that represents a time at least 5 seconds in the future.

```
utc_t               now, future, fivesec;
reltimespec_t       tfivesec;
timespec_t          tzero;

/*
 *   Construct a timestamp that represents 5 seconds...
 */
tfivesec.tv_sec = 5;
tfivesec.tv_nsec = 0;
tzero.tv_sec = 0;
tzero.tv_nsec = 0;
utc_mkbinreltime(&fivesec, /* Out: 5 secs in binary timestamp */
                 &tfivesec, /* In:  5 secs in timespec         */
                 &tzero);   /* In:  0 secs inaccuracy in timespec */

/*
 *  Get the maximum possible current time...
 *  (NULL input parameter is used to specify the current time.)
 */
utc_pointtime((utc_t *)0, /* Out: Earliest possible current time */
              (utc_t *)0, /* Out: Midpoint of current time       */
              &now,       /* Out: Latest possible current time   */
              (utc_t *)0);/* In:  Use current time               */

/*
 *   Add 5 seconds to get future timestamp...
 */
utc_addtime(&future,     /* Out: Future binary timestamp   */
```

```
            &now,         /* In:  Latest possible time now  */
            &fivesec);    /* In:  5 secs                    */
```

Related Information

Functions: `utc_subtime`

utc_anytime

utc_anytime — Converts a binary timestamp to a `tm` structure, using the time differential factor (TDF) information contained in the timestamp to determine the TDF returned with the `tm` structure.

Syntax

```
#include <utc.h>

int utc_anytime(timetm, *tns, *inacctm, *ins, *tdf, *utc)

   struct tm timetm;
   long *tns;
   struct tm *inacctm;
   long *ins;
   long *tdf;
   const utc_t *utc;
```

Parameters

Input

utc

Binary timestamp.

Output

timetm

Time component of the binary timestamp expressed in the timestamp's local time.

tns

Nanoseconds since time component of the binary timestamp.

inacctm

Seconds of inaccuracy component of the binary timestamp. If the inaccuracy is finite, then `tm_mday` returns a value of –1 and `tm_mon` and `tm_year` return values of 0. The field `tm_yday` contains the inaccuracy in days. If the inaccuracy is infinite, all `tm` structure fields return values of –1.

ins

Nanoseconds of inaccuracy component of the binary timestamp.

tdf

TDF component of the binary timestamp in units of seconds east or west of GMT.

Description

The **Any Time** routine converts a binary timestamp to a `tm` structure. The TDF information contained in the timestamp is returned with the time and inaccuracy components; the TDF component determines the offset from GMT and the local time value of the `tm` structure. Additional returns include nanoseconds since Time and nanoseconds of inaccuracy.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument or invalid results.

Example

The following example converts a timestamp, using the TDF information in the timestamp, then prints the result.

```
utc_t               evnt;
struct tm           tmevnt;
timespec_t          tevnt, ievnt;
char                tznam[80];

/*
 *   Assume evnt contains the timestamp to convert...
 *
 *   Get time as a tm structure, using the time zone information in
 *   the timestamp...
 */
utc_anytime(&tmevnt,         /* Out: tm struct of time of evnt  */
            (long *)0,       /* Out: nanosec of time of evnt    */
            (struct tm *)0,  /* Out: tm struct of inacc of evnt */
            (long *)0,       /* Out: nanosec of inacc of evnt   */
            (int *)0,        /* Out: tdf of evnt                */
            &evnt);          /* In:  binary timestamp of evnt   */

/*
 *   Get the time and inaccuracy as timespec structures...
 */
utc_bintime(&tevnt,          /* Out: timespec of time of evnt   */
            &ievnt,          /* Out: timespec of inacc of evnt  */
            (int *)0,        /* Out: tdf of evnt                */
            &evnt);          /* In:  Binary timestamp of evnt   */

/*
 *   Construct the time zone name from time zone information in the
 *   timestamp...
 */
utc_anyzone(tznam,           /* Out: Time zone name             */
            80,              /* In:  Size of time zone name     */
            (long *)0,       /* Out: tdf of event               */
            (long *)0,       /* Out: Daylight saving flag       */
            &evnt);          /* In:  Binary timestamp of evnt   */

/*
 *   Print timestamp in the format:
 *
```

```
 *           1991-03-05-21:27:50.023I0.140 (GMT-5:00)
 *           1992-04-02-12:37:24.003Iinf (GMT+7:00)
 *
 */

printf("%d-%02d-%02d-%02d:%02d:%02d.%03d",
        tmevnt.tm_year+1900, tmevnt.tm_mon+1, tmevnt.tm_mday,
        tmevnt.tm_hour, tmevnt.tm_min, tmevnt.tm_sec,
        (tevnt.tv_nsec/1000000));

if ((long)ievnt.tv_sec == -1)
    printf("Iinf");
else
    printf("I%d.%03d", ievnt.tv_sec, (ievnt.tv_nsec/1000000));

printf(" (%s)\n", tznam);
```

Related Information

Functions: `utc_mkanytime`, `utc_anyzone`, `utc_gettime`, `utc_getusertime`, `utc_gmtime`, `utc_localtime`

utc_anyzone

utc_anyzone — Gets the time zone label and offset from GMT, using the TDF contained in the input utc.

Syntax

```
#include <utc.h>

int utc_anyzone(tzname, tzlen, *tdf, isdst, *utc)

   char tzname;
   size_t tzlen;
   long *tdf;
   int *isdst;
   const utc_t *utc;
```

Parameters

Input

tzlen

Length of the *tzname* buffer.

utc

Binary time.

Output

tzname

Character string that is long enough to hold the time zone label.

tdf

Longword with differential in seconds east or west of GMT.

isdst

Integer with a value of –1, indicating that no information is supplied as to whether it is standard time or daylight saving time. A value of –1 is always returned.

Description

The **Any Zone** routine gets the time zone label and offset from GMT, using the TDF contained in the input `utc`. The label returned is always of the form GMT+ *n* or GMT - *n*, where *n* is the TDF expressed in hours:minutes. (The label associated with an arbitrary time zone is not known; only the offset is known.)

Notes

All of the output parameters are optional. No value is returned and no error occurs if the pointer is null.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument or an insufficient buffer.

See the sample program in the Examples section of the `utc_anytime` routine.

Related Information

Functions: `utc_anytime`, `utc_gmtzone`, `utc_localzone`

utc_ascanytime

utc_ascanytime — Converts a binary timestamp to an ASCII string that represents an arbitrary time zone.

Syntax

```
#include <utc.h>

int utc_ascanytime(*cp, stringlen, *utc)

   char *cp;
   size_t stringlen;
   const utc_t *utc;
```

Parameters

Input

stringlen

The length of the *cp* buffer.

utc

Binary timestamp.

Output

cp

ASCII string that represents the time.

Description

The **ASCII Any Time** routine converts a binary timestamp to an ASCII string that expresses a time. The TDF component in the timestamp determines the local time used in the conversion.

Return Values

0	Indicates that the routine executed successfully.
–1	Indicates an invalid time parameter or invalid results.

Example

The following example converts a time to an ASCII string that expresses the time in the time zone where the timestamp was generated.

```
utc_t      evnt;
char       localTime[UTC_MAX_STR_LEN];

/*
 *  Assuming that evnt contains the timestamp to convert, convert
 *  the time to ASCII in the following format:
 *
 *          1991-04-01-12:27:38.37-8:00I2.00
 */

utc_ascanytime(localtime,        /* Out: Converted time    */
               UTC_MAX_STR_LEN,  /* In:  Length of string  */
               &evnt);           /* In:  Time to convert   */
```

Related Information

Functions: `utc_ascgmtime`, `utc_asclocaltime`

utc_ascgmtime

utc_ascgmtime — Converts a binary timestamp to an ASCII string that expresses a GMT time.

Syntax

```
#include <utc.h>

int utc_ascgmtime(*cp, stringlen, *utc)

   char *cp;
```

```
size_t stringlen;
const utc_t *utc;
```

Parameters

Input

stringlen

Length of the *cp* buffer.

utc

Binary timestamp.

Output

cp

ASCII string that represents the time.

Description

The **ASCII GMT Time** routine converts a binary timestamp to an ASCII string that expresses a time in GMT.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time parameter or invalid results.

Example

The following example converts the current time to GMT format.

```
char   gmTime[UTC_MAX_STR_LEN];

/*
 *   Convert the current time to ASCII in the following format:
 *
 *          1991-04-01-12:27:38.37I2.00
 */

utc_ascgmtime(gmTime,           /* Out: Converted time     */
              UTC_MAX_STR_LEN,  /* In:  Length of string   */
              (utc_t*) NULL);   /* In:  Time to convert    */
                                /* Default is current time */
```

Related Information

Functions: `utc_ascanytime`, `utc_asclocaltime`

utc_asclocaltime

utc_asclocaltime — Converts a binary timestamp to an ASCII string that represents a local time.

Syntax

```
#include <utc.h>

int utc_asclocaltime(*cp, stringlen, *utc)

   char *cp;
   size_t stringlen;
   const utc_t *utc;
```

Parameters

Input

stringlen

Length of the *cp* buffer.

utc

Binary timestamp.

Output

cp

ASCII string that represents the time.

Description

The **ASCII Local Time** routine converts a binary timestamp to an ASCII string that expresses local time.

The user's environment determines the time zone rule (details are system dependent).

OpenVMS only: The user selects a time zone by defining `sys$timezone_rule`, which is created when the `sys$manager:net$configure.com` is run.

UNIX only: The user selects a time zone by specifying the time zone environment variable. (The reference page for the `localtime( )` system call provides additional information.)

If the user's environment does not specify a time zone rule, the system's rule is used (details of the rule are system dependent).

OpenVMS only: OpenVMS systems do not have a default time zone rule. You must run the `sys $manager:net$configure` procedure to specify a time zone.

UNIX only: The rule in `/etc/zoneinfo/localtime` applies.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time parameter or invalid results.

Example

The following example converts the current time to local time.

```
char   localTime[UTC_MAX_STR_LEN];

/*
 *   Convert the current time...
 */

utc_asclocaltime(localTime,      /* Out: Converted time          */
                 UTC_MAX_STR_LEN, /* In:  Length of string       */
                 (utc_t*) NULL);  /* In:  Time to convert        */
                                  /*      Default is current time */
```

Related Information

Functions: `utc_ascanytime`, `utc_ascgmtime`

utc_ascreltime

utc_ascreltime — Converts a relative binary timestamp to an ASCII string that represents the time.

Syntax

```
#include <utc.h>

int utc_ascreltime(*cp, stringlen, *utc)

   char *cp;
   const size_t stringlen;
   const utc_t *utc;
```

Parameters

Input

utc

Relative binary timestamp.

stringlen

Length of the *cp* buffer.

Output

cp

ASCII string that represents the time.

Description

The **ASCII Relative Time** routine converts a relative binary timestamp to an ASCII string that represents the time.

Return Values

0	Indicates that the routine executed successfully.

-1	Indicates an invalid time parameter or invalid results.

Example

See the sample program in the **Examples** section of the `utc_abstime` routine.

Related Information

Functions: `utc_mkascreltime`

utc_binreltime

utc_binreltime — Converts a relative binary timestamp to two `timespec` structures that express relative time and inaccuracy.

Syntax

```
#include <utc.h>

int utc_binreltime(*timesp, *inaccsp, *utc)

reltimespec_t *timesp;
timespec_t *inaccsp;
const utc_t *utc;
```

Parameters

Input

utc

Relative binary timestamp.

Output

timesp

Time component of the relative binary timestamp, in the form of seconds and nanoseconds since the base time (1970-01-01:00:00:00.0 + 00:00I0).

inaccsp

Inaccuracy component of the relative binary timestamp, in the form of seconds and nanoseconds.

Description

The **Binary Relative Time** routine converts a relative binary timestamp to two `timespec` structures that express relative time and inaccuracy. These `timespec` structures describe a time interval.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument or invalid results.

Example

The following example measures the duration of a process, then prints the resulting relative time and inaccuracy.

```
utc_t               before, duration;
reltimespec_t       tduration;
timespec_t          iduration;

/*
 *   Get the time before the start of the operation...
 */

utc_gettime(&before);      /* Out: Before binary timestamp    */

/*
 * ...Later...
 *
 * Subtract, getting the duration as a relative time.
 *
 *    NOTE: The NULL argument is used to obtain the current time.
 */

utc_subtime(&duration,     /* Out: Duration rel bin timestamp  */
            (utc_t *)0,    /* In:  After binary timestamp      */
            &before);      /* In:  Before binary timestamp     */

/*
 *    Convert the relative times to timespec structures...
 */

utc_binreltime(&tduration, /* Out: Duration time timespec      */
               &iduration, /* Out: Duration inacc timespec     */
               &duration); /* In:  Duration rel bin timestamp  */

/*
 *    Print the duration...
 */

printf("%d.%04d", tduration.tv_sec, (tduration.tv_nsec/10000));

if ((long)iduration.tv_sec == -1)
    printf("Iinf\n");
else
printf("I%d.%04d\n", iduration.tv_sec, (iduration.tv_nsec/100000));
```

Related Information

Functions: `utc_mkbinreltime`

utc_bintime

utc_bintime — Converts a binary timestamp to a `timespec` structure.

Syntax

```
#include <utc.h>
```

```
int utc_bintime(*timesp, *inaccsp, *tdf, *utc)

   timespec_t *timesp;
   timespec_t *inaccsp;
   long *tdf;
   const utc_t *utc;
```

Parameters

Input

utc

Binary timestamp.

Output

timesp

Time component of the binary timestamp, in the form of seconds and nanoseconds since the base time.

inaccsp

Inaccuracy component of the binary timestamp, in the form of seconds and nanoseconds.

tdf

TDF component of the binary timestamp in the form of signed number of seconds east or west of GMT.

Decsription

The **Binary Time** routine converts a binary timestamp to a `timespec` structure. The TDF information contained in the timestamp is returned.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument or invalid results.

Example

See the sample program in the **Examples** section of the `utc_anytime` routine.

Related Information

Functions: `utc_binreltime`, `utc_mkbintime`

utc_boundtime

utc_boundtime — Given two UTC times, one before and one after an event, returns a single UTC time whose inaccuracy includes the event.

Syntax

```
#include <utc.h>
```

```
int utc_boundtime(*result, *utc1, *utc2)

   utc_t *result;
   const utc_t *utc1;
   const utc_t *utc2;
```

Parameters

Input

utc1

Before binary timestamp or relative binary timestamp.

utc2

After binary timestamp or relative binary timestamp.

Output

result

Spanning timestamp.

Description

Given two UTC times, the **Bound Time** routine returns a single UTC time whose inaccuracy bounds the two input times. This is useful for timestamping events; the routine gets the `utc` values before and after the event, then calls `utc_boundtime` to build a timestamp that includes the event.

Notes

The TDF in the output UTC value is copied from the *utc2* input. If one or both input values have infinite inaccuracies, the returned time value also has an infinite inaccuracy and is the average of the two input values.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time parameter or invalid parameter order.

Example

The following example records the time of an event and constructs a single timestamp, which includes the time of the event. Note that the `utc_getusertime` routine is called so the time zone information that is included in the timestamp references the user's environment rather than the system's default time zone.

The user's environment determines the time zone rule (details are system dependent).

OpenVMS: The user selects a time zone by defining `sys$timezone_rule`, which is created when the `sys$manager:net$configure.com` is run.

UNIX: The user selects a time zone by specifying the time zone environment variable. (The reference page for the `localtime( )` system call provides additional information.)

If the user's environment does not specify a time zone rule, the system's rule is used (details of the rule are system dependent).

OpenVMS: OpenVMS systems do not have a default time zone rule. You must run the `sys $manager:net$configure` procedure to specify a time zone.

UNIX: The rule in `/etc/zoneinfo/localtime` applies.

```
utc_t               before, after, evnt;

/*
 *   Get the time before the event...
 */

utc_getusertime(&before);  /* Out: Before binary timestamp  */

/*
 *   Get the time after the event...
 */

utc_getusertime(&after);   /* Out: After binary timestamp  */

/*
 *   Construct a single timestamp that describes the time of the
 *   event...
 */

utc_boundtime(&evnt,   /* Out: Timestamp that bounds event */
              &before, /* In:  Before binary timestamp     */
              &after); /* In:  After binary timestamp      */
```

Related Information

Functions: `utc_gettime`, `utc_pointtime`, `utc_spantime`

utc_cmpintervaltime

utc_cmpintervaltime — Compares two binary timestamps or two relative binary timestamps.

Syntax

```
#include <utc.h>

int utc_cmpintervaltime(*relation, *utc1, *utc2)

   enum utc_cmptype *relation;
   const utc_t *utc1;
   const utc_t *utc2;
```

Parameters

Input

utc1

Binary timestamp or relative binary timestamp.

utc2

Binary timestamp or relative binary timestamp.

Output

relation

Receives the result of the comparison of *utc1*:*utc2*, where the result is an enumerated type with one of the following values:

- `utc_equalTo`
- `utc_lessThan`
- `utc_greaterThan`
- `utc_indeterminate`

Description

The **Compare Interval Time** routine compares two binary timestamps and returns a flag indicating that the first time is greater than, less than, equal to, or overlapping with the second time. Two times overlap if the intervals (time inaccuracy, time + inaccuracy) of the two times intersect.

The input binary timestamps express two absolute or two relative times. Do not compare relative binary timestamps and binary timestamps. If you do, no meaningful results and no errors are returned.

This routine does a temporal ordering of the time intervals.

```
utc1 is utc_lessThan utc2 iff
        utc1.time + utc1.inacc < utc2.time - utc2.inacc

utc1 is utc_greaterThan utc2 iff
        utc1.time - utc1.inacc > utc2.time + utc2.inacc

utc1 utc_equalTo utc2 iff
        utc1.time == utc2.time and
        utc1.inacc == 0 and
        utc2.inacc == 0
```

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument.

Example

The following example checks to see if the current time is definitely after 1:00 P.M. today GMT.

```
struct tm           tmtime, tmzero;
enum utc_cmptype    relation;
utc_t               testtime;

/*
 *   Zero the tm structure for inaccuracy...
 */
```

```
memset(&tmzero, 0, sizeof(tmzero));

/*
 *   Get the current time, mapped to a tm structure...
 *
 *        NOTE: The NULL argument is used to get the current time.
 */

utc_gmtime(&tmtime,      /* Out: Current GMT time in tm struct   */
         (long *)0,      /* Out: Nanoseconds of time             */
         (struct tm *)0, /* Out: Current inaccuracy in tm struct */
         (long *)0,      /* Out: Nanoseconds of inaccuracy       */
         (utc_t *)0);    /* In:  Current timestamp               */

/*
 *   Construct a tm structure that corresponds to 1:00 PM...
 */

tmtime.tm_hour = 13;
tmtime.tm_min = 0;
tmtime.tm_sec = 0;

/*
 *   Convert to a binary timestamp...
 */

utc_mkgmtime(&testtime,  /* Out: Binary timestamp of 1:00 PM  */
             &tmtime,    /* In:  1:00 PM in tm struct         */
             0,          /* In:  Nanoseconds of time          */
             &tmzero,    /* In:  Zero inaccuracy in tm struct */
             0);         /* In:  Nanoseconds of inaccuracy    */

/*
 *   Compare to the current time, noting the use of the
 *   NULL argument...
 */

utc_cmpintervaltime(&relation,    /* Out: Comparison relation */
                    (utc_t *)0,   /* In:  Current timestamp   */
                    &testtime);   /* In:  1:00 PM timestamp   */

/*
 *   If it is not later - wait, print a message, etc.
 */

if (relation != utc_greaterThan) {

/*
 *     Note: It could be earlier than 1:00 PM or it could be
 *           indeterminate. If indeterminate, for some applications
 *           it might be worth waiting.
 */
}
```

Related Information

Functions: `utc_cmpmidtime`

utc_cmpmidtime

utc_cmpmidtime — Compares two binary timestamps or two relative binary timestamps, ignoring inaccuracies.

Syntax

```
#include <utc.h>

int utc_cmpmidtime(*relation, *utc1, *utc2)

   enum utc_cmptype *relation;
   const utc_t *utc1;
   const utc_t *utc2;
```

Parameters

Input

utc1

Binary timestamp or relative binary timestamp.

utc2

Binary timestamp or relative binary timestamp.

Output

relation

Result of the comparison of *utc1*:*utc2*, where the result is an enumerated type with one of the following values:

- `utc_equalTo`
- `utc_lessThan`
- `utc_greaterThan`

Description

The **Compare Midpoint Times** routine compares two binary timestamps and returns a flag indicating that the first timestamp is greater than, less than, or equal to the second timestamp. Inaccuracy information is ignored for this comparison; the input values are, therefore, equivalent to the midpoints of the time intervals described by the input binary timestamps.

The input binary timestamps express two absolute or two relative times. Do not compare relative binary timestamps and binary timestamps. If you do, no meaningful results and no errors are returned.

The following routine does a lexical ordering on the time interval midpoints.

```
utc1 is utc_lessThan utc2 iff
        utc1.time < utc2.time

utc1 is utc_greaterThan utc2 iff
```

```
        utc1.time > utc2.time

utc1 is utc_equalTo utc2 iff
        utc1.time == utc2.time
```

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument.

Example

The following example checks if the current time (ignoring inaccuracies) is after 1:00 P.M. today local time.

```
struct tm           tmtime, tmzero;
enum utc_cmptype    relation;
utc_t               testtime;

/*
 *   Zero the tm structure for inaccuracy...
 */

memset(&tmzero, 0, sizeof(tmzero));

/*
 *   Get the current time, mapped to a tm structure...
 *
 *   NOTE:  The NULL argument is used to get the current time.
 */

utc_localtime(&tmtime,   /* Out: Current local time in tm struct */
         (long *)0,      /* Out: Nanoseconds of time             */
         (struct tm *)0, /* Out: Current inacc in tm struct      */
         (long *)0,      /* Out: Nanoseconds of inaccuracy       */
         (utc_t *)0);    /* In:  Current timestamp               */

/*
 *   Construct a tm structure that corresponds to 1:00 P.M....
 */

tmtime.tm_hour = 13;
tmtime.tm_min = 0;
tmtime.tm_sec = 0;

/*
 *   Convert to a binary timestamp...
 */

utc_mklocaltime(&testtime, /* Out: Binary timestamp of 1:00 P.M. */
                &tmtime,   /* In:  1:00 P.M. in tm struct        */
                0,         /* In:  Nanoseconds of time           */
                &tmzero,   /* In:  Zero inaccuracy in tm struct  */
                0);        /* In:  Nanoseconds of inaccuracy     */

/*
 *   Compare to the current time, noting the use of the
```

```
 *   NULL argument...
 */

utc_cmpmidtime(&relation,    /* Out: Comparison relation         */
               (utc_t *)0,   /* In:  Current timestamp           */
               &testtime);   /* In:  1:00 P.M. timestamp         */

/*
 *   If the time is not later - wait, print a message, etc.
 */

if (relation != utc_greaterThan) {

/*          It is not later then 1:00 P.M. local time. Note that
 *          this depends on the setting of the user's environment.
 */
}
```

Related Information

Functions: `utc_cmpintervaltime`

utc_gettime

utc_gettime — Returns the current system time and inaccuracy as a binary timestamp.

Syntax

```
#include <utc.h>

int utc_gettime(*utc)

   utc_t *utc;
```

Parameters

Input

None.

Output

utc

System time as a binary timestamp.

Description

The **Get Time** routine returns the current system time and inaccuracy in a binary timestamp. The routine takes the TDF from the operating system's kernel; the TDF is specified in a system-dependent manner.

Return Values

0	Indicates that the routine executed successfully.

-1	Generic error that indicates the time service cannot be accessed.

Example

See the sample program in the **Examples** section of the `utc_binreltime` routine.

utc_getusertime

utc_getusertime — Returns the time and process-specific TDF, rather than the system-specific TDF.

Syntax

```
#include <utc.h>

int utc_getusertime(*utc)

   utc_t *utc;
```

Parameters

Input

None.

Output

utc

System time as a binary timestamp.

Description

The **Get User Time** routine returns the system time and inaccuracy in a binary timestamp. The routine takes the TDF from the user's environment, which determines the time zone rule (details are system dependent).

OpenVMS: The user selects a time zone by defining `sys$timezone_rule`, which is created when the `sys$manager:net$configure.com` is run.

UNIX: The user selects a time zone by specifying the time zone environment variable. (The reference page for the `localtime( )` system call provides additional information.)

If the user's environment does not specify a TDF, the system's TDF is used. The system's time zone rule is applied (details of the rule are system dependent).

OpenVMS: OpenVMS systems do not have a default time zone rule. You must run the `sys $manager:net$configure` procedure to specify a time zone.

UNIX: The rule in `/etc/zoneinfo/localtime` applies.

Return Values

0	Indicates that the routine executed successfully.

-1	Generic error that indicates the time service cannot be accessed.

Example

See the sample program in the **Examples** section of the `utc_boundtime` routine.

Related Information

Functions: `utc_gettime`

utc_gmtime

utc_gmtime — Converts a binary timestamp to a tm structure that expresses GMT or the equivalent UTC.

Syntax

```
#include <utc.h>

int utc_gmtime(*timetm, *tns, *inacctm, *ins, *utc)

   struct tm *timetm;
   long *tns;
   struct tm *inacctm;
   long *ins;
   const utc_t *utc;
```

Parameters

Input

utc

Binary timestamp to be converted to `tm` structure components.

Output

timetm

Time component of the binary timestamp.

tns

Nanoseconds since time component of the binary timestamp.

inacctm

Seconds of inaccuracy component of the binary timestamp. If the inaccuracy is finite, then `tm_mday` returns a value of –1 and `tm_mon` and `tm_year` return values of zero. The field `tm_yday` contains the inaccuracy in days. If the inaccuracy is infinite, all `tm` structure fields return values of –1.

ins

Nanoseconds of inaccuracy component of the binary timestamp. If the inaccuracy is infinite, *ins* returns a value of –1.

Description

The **Greenwich Mean Time** (GMT) routine converts a binary timestamp to a `tm` structure that expresses GMT (or the equivalent UTC). Additional returns include nanoseconds since time and nanoseconds of inaccuracy.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument or invalid results.

Example

See the sample program in the **Examples** section of the `utc_cmpintervaltime` routine.

Related Information

Functions: `utc_anytime`, `utc_gmtzone`, `utc_localtime`, `utc_mkgmtime`

utc_gmtzone

utc_gmtzone — Gets the time zone label for GMT.

Syntax

```
#include <utc.h>

int utc_gmtzone(*tzname, tzlen, *tdf, *isdst, *utc)

   char *tzname;
   size_t tzlen;
   long *tdf;
   int *isdst;
   const utc_t *utc;
```

Parameters

Input

tzlen

Length of buffer *tzname*.

utc

Binary timestamp. This parameter is ignored.

Output

tzname

Character string long enough to hold the time zone label.

tdf

Longword with differential in seconds east or west of GMT. A value of zero is always returned.

isdst

Integer with a value of zero, indicating that daylight saving time is not in effect. A value of zero is always returned.

Description

The **Greenwich Mean Time Zone** routine gets the time zone label and zero offset from GMT. Outputs are always *tdf* = 0 and *tzname* = GMT. This routine exists for symmetry with the **Any Zone** (`utc_anyzone`) and the **Local Zone** (`utc_localzone`) routines.

Notes

All of the output parameters are optional. No value is returned and no error occurs if the *tzname* pointer is NULL.

Return Values

0	Indicates that the routine executed successfully (always returned).

Example

The following example prints out the current time in both local time and GMT time.

```
utc_t       now;
struct tm   tmlocal, tmgmt;
long        tzoffset;
int         tzdaylight;
char        tzlocal[80], tzgmt[80];

/*
 *   Get the current time once, so both conversions use the same
 *   time...
 */

utc_gettime(&now);

/*
 *   Convert to local time, using the process TZ environment
 *   variable...
 */

utc_localtime(&tmlocal,       /* Out: Local time tm structure  */
              (long *)0,      /* Out: Nanosec of time          */
              (struct tm *)0, /* Out: Inaccuracy tm structure  */
              (long *)0,      /* Out: Nanosec of inaccuracy    */
              &now);          /* In:  Current binary timestamp */

/*
 *   Get the local time zone name, offset from GMT, and current
 *   daylight savings flag...
 */

utc_localzone(tzlocal,     /* Out: Local time zone name         */
```

```
              80,          /* In:  Length of loc time zone name  */
              &tzoffset,   /* Out: Loc time zone offset in secs  */
              &tzdaylight, /* Out: Local time zone daylight flag */
              &now);       /* In:  Current binary timestamp      */

/*
 *   Convert to GMT...
 */

utc_gmtime(&tmgmt,         /* Out: GMT tm structure          */
           (long *)0,      /* Out: Nanoseconds of time       */
           (struct tm *)0, /* Out: Inaccuracy tm structure   */
           (long *)0,      /* Out: Nanoseconds of inaccuracy */
           &now);          /* In:  Current binary timestamp  */

/*
 *   Get the GMT time zone name...
 */

utc_gmtzone(tzgmt,      /* Out: GMT time zone name           */
            80,         /* In:  Size of GMT time zone name   */
            (long *)0,  /* Out: GMT time zone offset in secs */
            (int *)0,   /* Out: GMT time zone daylight flag  */
            &now);      /* In:  Current binary timestamp     */

/*
 *   Print out times and time zone information in the following
 *   format:
 *
 *          12:00:37 (EDT) = 16:00:37 (GMT)
 *          EDT is -240 minutes ahead of Greenwich Mean Time.
 *          Daylight savings time is in effect.
 */

printf("%d:%02d:%02d (%s) = %d:%02d:%02d (%s)\n",
       tmlocal.tm_hour, tmlocal.tm_min, tmlocal.tm_sec, tzlocal,
       tmgmt.tm_hour, tmgmt.tm_min, tmgmt.tm_sec, tzgmt);
printf("%s is %d minutes ahead of Greenwich Mean Time\n",
       tzlocal, tzoffset/60);
if (tzdaylight != 0)
    printf("Daylight savings time is in effect\n");
```

Related Information

Functions: `utc_anyzone`, `utc_gmtime`, `utc_localzone`

utc_localtime

utc_localtime — Converts a binary timestamp to a tm structure that expresses local time.

Syntax

```
#include <utc.h>

int utc_localtime(*timetm, *tns, *inacctm, *ins, *utc)
```

```
struct tm *timetm;
long *tns;
struct tm *inacctm;
long *ins;
const utc_t *utc;
```

Parameters

Input

utc

Binary timestamp.

Output

timetm

Time component of the binary timestamp, expressing local time.

tns

Nanoseconds since time component of the binary timestamp.

inacctm

Seconds of inaccuracy component of the binary timestamp. If the inaccuracy is finite, then `tm_mday` returns a value of –1 and `tm_mon` and `tm_year` return values of zero. The field `tm_yday` contains the inaccuracy in days. If the inaccuracy is infinite, all `tm` structure fields return values of –1.

ins

Nanoseconds of inaccuracy component of the binary timestamp. If the inaccuracy is infinite, *ins* returns a value of –1.

Description

The **Local Time** routine converts a binary timestamp to a `tm` structure that expresses local time.

The user's environment determines the time zone rule (details are system dependent).

OpenVMS: The user selects a time zone by defining `sys$timezone_rule`, which is created when the `sys$manager:net$configure.com` is run.

UNIX: The user selects a time zone by specifying the time zone environment variable. (The reference page for the `localtime()` system call provides additional information.)

If the user's environment does not specify a time zone rule, the system's rule is used (details of the rule are system dependent).

OpenVMS: OpenVMS systems do not have a default time zone rule. You must run the `sys $manager:net$configure` procedure to specify a time zone.

UNIX: The rule in `/etc/zoneinfo/localtime` applies.

Additional returns include nanoseconds since time and nanoseconds of inaccuracy.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument or invalid results.

Example

See the sample program in the **Examples** section of the `utc_gmtzone` routine.

Related Information

Functions: `utc_anytime`, `utc_gmtime`, `utc_localzone`, `utc_mklocaltime`

utc_localzone

utc_localzone — Gets the local time zone label and offset from GMT, given `utc`.

Syntax

```
#include <utc.h>

int utc_localzone(*tzname, tzlen, *tdf, *isdst, *utc)

   char *tzname;
   size_t tzlen;
   long *tdf;
   int *isdst;
   const utc_t *utc;

#include <utc.h>

int utc_localzone(*tzname, tzlen, *tdf, *isdst, *utc)
```

Parameters

Input

tzlen

Length of the *tzname* buffer.

utc

Binary timestamp.

Output

tzname

Character string long enough to hold the time zone label.

tdf

Longword with differential in seconds east or west of GMT.

isdst

Integer with a value of zero if standard time is in effect or a value of 1 if daylight savings time is in effect.

Description

The **Local Zone** routine gets the local time zone label and offset from GMT, given `utc`.

The user's environment determines the time zone rule (details are system dependent).

OpenVMS: The user selects a time zone by defining `sys$timezone_rule`, which is created when the `sys$manager:net$configure.com` is run.

UNIX: The user selects a time zone by specifying the time zone environment variable. (The reference page for the `localtime( )` system call provides additional information.)

If the user's environment does not specify a time zone rule, the system's rule is used (details of the rule are system dependent).

OpenVMS: OpenVMS systems do not have a default time zone rule. You must run the `sys $manager:net$configure` procedure to specify a time zone.

UNIX: The rule in `/etc/zoneinfo/localtime` applies.

Notes

All of the output parameters are optional. No value is returned and no error occurs if the pointer is null.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument or an insufficient buffer.

Example

See the sample program in the **Examples** section of the `utc_gmtzone` routine.

Related Information

Functions: `utc_anyzone`, `utc_gmtzone`, `utc_localtime`

utc_mkanytime

utc_mkanytime — Converts a `tm` structure and TDF (expressing the time in an arbitrary time zone) to a binary timestamp.

Syntax

```
#include <utc.h>

int utc_mkanytime(*utc, *timetm, tns, *inacctm, ins, tdf)

   utc_t *utc;
   const struct tm *timetm;
```

```
long tns;
const struct tm *inacctm;
long ins;
long tdf;
```

Parameters

Input

timetm

A tm structure that expresses the local time; `tm_wday` and `tm_yday` are ignored on input.

tns

Nanoseconds since time component.

inacctm

A `tm` structure that expresses days, hours, minutes, and seconds of inaccuracy. If `tm_yday` is negative, the inaccuracy is considered to be infinite; `tm_mday`, `tm_mon`, `tm_wday`, `tm_isdst`, `tm_gmtoff`, and `tm_zone` are ignored on input.

ins

Nanoseconds of inaccuracy component.

tdf

Time differential factor to use in conversion.

Output

utc

Resulting binary timestamp.

Description

The **Make Any Time** routine converts a tm structure and TDF (expressing the time in an arbitrary time zone) to a binary timestamp. Required inputs include nanoseconds since time and nanoseconds of inaccuracy.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument or invalid results.

Example

The following example converts a string ISO format time in an arbitrary time zone to a binary timestamp. This may be part of an input timestamp routine, although a real implementation will include range checking.

```
utc_t       utc;
struct tm   tmtime, tminacc;
```

```
float       tsec, isec;
double      tmp;
long        tnsec, insec;
int         i, offset, tzhour, tzmin, year, mon;
char        *string;

/*  Try to convert the string...           */

if(sscanf(string, "%d-%d-%d-%d:%d:%e+%d:%dI%e",
          &year, &mon, &tmtime.tm_mday, &tmtime.tm_hour,
          &tmtime.tm_min, &tsec, &tzhour, &tzmin, &isec) != 9) {

/*  Try again with a negative TDF...                          */

if (sscanf(string, "%d-%d-%d-%d:%d:%e-%d:%dI%e",
           &year, &mon, &tmtime.tm_mday, &tmtime.tm_hour,
           &tmtime.tm_min, &tsec, &tzhour, &tzmin, &isec) != 9) {

/*  ERROR                                                     */

        exit(1);
    }

/*  TDF is negative                                           */

    tzhour = -tzhour;
    tzmin = -tzmin;

}

/*  Fill in the fields...                                     */

tmtime.tm_year = year - 1900;
tmtime.tm_mon = --mon;
tmtime.tm_sec = tsec;
tnsec = (modf(tsec, &tmp)*1.0E9);
offset = tzhour*3600 + tzmin*60;
tminacc.tm_sec = isec;
insec = (modf(isec, &tmp)*1.0E9);

/* Convert to a binary timestamp...                           */

utc_mkanytime(&utc,     /* Out: Resultant binary timestamp      */
              &tmtime,  /* In:  tm struct that represents input */
              tnsec,    /* In:  Nanoseconds from input          */
              &tminacc, /* In:  tm struct that represents inacc */
              insec,    /* In:  Nanoseconds from input          */
              offset);  /* In:  TDF from input                  */
```

Related Information

Functions: `utc_anytime`, `utc_anyzone`

utc_mkascreltime

utc_mkascreltime — Converts a null-terminated character string that represents a relative timestamp to a binary timestamp.

Syntax

```
#include <utc.h>

int utc_mkascreltime(*utc, *string)

   utc_t *utc;
   char *string;
```

Parameters

Input

string

A null-terminated string that expresses a relative timestamp in its ISO format.

Output

utc

Resulting binary timestamp.

Description

The **Make ASCII Relative Time** routine converts a null-terminated string, which represents a relative timestamp, to a binary timestamp.

Notes

The ASCII string must be null-terminated.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time parameter or invalid results.

Example

```
utc_t      utc;
char       str[UTC_MAX_STR_LEN];

/*
 * Relative time of 333 days, 12 hours, 1 minute, 37.223 seconds
 * Inaccuracy of 50.22 sec. in the format: -333-12:01:37.223I50.22
 */

(void)strcpy((void *)str,
             "-333-12:01:37.223I50.22");

utc_mkascreltime(&utc,  /* Out: Binary utc  */
                 str);  /* In:  String      */
```

Related Information

Functions: `utc_ascreltime`

utc_mkasctime

utc_mkasctime — Converts a null-terminated character string that represents an absolute time to a binary timestamp.

Syntax

```
#include <utc.h>

int utc_mkasctime(*utc, *string)

   utc_t *utc;
   char *string;
```

Parameters

Input

string

A null-terminated string that expresses an absolute time.

Output

utc

Resulting binary timestamp.

Description

The **Make ASCII Time** routine converts a null-terminated string that represents an absolute time to a binary timestamp.

Notes

The ASCII string must be null-terminated.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time parameter or invalid results.

Example

The following example converts an ASCII time string to its binary equivalent.

```
utc_t     utc;
char      str[UTC_MAX_STR_LEN];

/*
 *   July 4, 1776, 12:01:37.223 local time
 *   TDF of -5:00 hours
 *   Inaccuracy of 3600.32 seconds
 */
```

```
(void)strcpy((void *)str,
             "1776-07-04-12:01:37.223-5:00 I 3600.32");

utc_mkasctime(&utc,     /* Out: Binary utc  */
              str);     /* In:  String      */
```

Related Information

Functions: `utc_ascanytime`, `utc_ascgmtime`, `utc_asclocaltime`

utc_mkbinreltime

utc_mkbinreltime — Converts a `timespec` structure expressing a relative time to a binary timestamp.

Syntax

```
#include <utc.h>

int utc_mkbinreltime(*utc, *timesp, *inaccsp)

   utc_t *utc;
   const reltimespec_t *timesp;
   const timespec_t *inaccsp;
```

Parameters

Input

timesp

A `reltimespec` structure that expresses a relative time.

inaccsp

A `timespec` structure that expresses inaccuracy. If `tv_sec` is set to a value of –1, the inaccuracy is considered to be infinite.

Output

utc

Resulting relative binary timestamp.

Description

The **Make Binary Relative Time** routine converts a `timespec` structure that expresses relative time to a binary timestamp.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument or invalid results.

Example

See the sample program in the **Examples** section of the `utc_addtime` routine.

Related Information

Functions: `utc_binreltime`, `utc_mkbintime`

utc_mkbintime

utc_mkbintime — Converts a `timespec` structure to a binary timestamp.

Syntax

```
#include <utc.h>

int utc_mkbintime(*utc, *timesp, *inaccsp)

   utc_t *utc;
   const timespec_t *timesp;
   const timespec_t *inaccsp;
   long tdf;
```

Parameters

Input

timesp

A `timespec` structure that expresses time since 1970-01-01:00:00:00.0+0:00I0.

inaccsp

A `timespec` structure that expresses inaccuracy. If `tv_sec` is set to a value of –1, the inaccuracy is considered to be infinite.

tdf

TDF component of the binary timestamp.

Output

utc

Resulting binary timestamp.

Description

The **Make Binary Time** routine converts a `timespec` structure time to a binary timestamp. The TDF input is used as the TDF of the binary timestamp.

Return Values

0	Indicates that the routine executed successfully.

-1	Indicates an invalid time argument or invalid results.

Example

The following example obtains the current time from `time( )`, converts it to a binary timestamp with an inaccuracy of 5.2 seconds, and specifies GMT.

```
timespec_t   ttime, tinacc;
utc_t        utc;

/*
 *   Obtain the current time (without the inaccuracy)...
 */

ttime.tv_sec = time((time_t *)0);
ttime.tv_nsec = 0;

/*
 *   Specify the inaccuracy...
 */

tinacc.tv_sec = 5;
tinacc.tv_nsec = 200000000;

/*
 *   Convert to a binary timestamp...
 */

utc_mkbintime(&utc,     /* Out: Binary timestamp          */
              &ttime,   /* In:  Current time in timespec  */
              &tinacc,  /* In:  5.2 seconds in timespec   */
              0);       /* In:  TDF of GMT                */
```

Related Information

Functions: `utc_bintime`, `utc_mkbinreltime`

utc_mkgmtime

utc_mkgmtime — Converts a `tm` structure that expresses GMT or UTC to a binary timestamp.

Syntax

```
#include <utc.h>

int utc_mkgmtime(*utc, *timetm, tns, *inacctm, ins)

   utc_t *utc;
   const struct tm *timetm;
   long tns;
   const struct tm *inacctm;
   long ins;
```

Parameters

Input

timetm

A `tm` structure that expresses GMT. On input, `tm_wday` and `tm_yday` are ignored.

tns

Nanoseconds since time component.

inacctm

A `tm` structure that expresses days, hours, minutes, and seconds of inaccuracy. If `tm_yday` is negative, the inaccuracy is considered to be infinite. On input, `tm_mday`, `tm_mon`, `tm_wday`, `tm_isdst`, `tm_gmtoff`, and `tm_zone` are ignored.

ins

Nanoseconds of inaccuracy component.

Output

utc

Resulting binary timestamp.

Description

The **Make Greenwich Mean Time** routine converts a `tm` structure that expresses GMT or UTC to a binary timestamp. Additional inputs include nanoseconds since the last second of time and nanoseconds of inaccuracy.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument or invalid results.

Example

See the sample program in the **Examples** section of the `utc_cmpintervaltime` routine.

Related Information

Functions: `utc_gmtime`

utc_mklocaltime

utc_mklocaltime — Converts a `tm` structure that expresses local time to a binary timestamp.

Syntax

```
#include <utc.h>

int utc_mklocaltime(*utc, *timetm, tns, *inacctm, ins)
```

```
utc_t *utc;
const struct tm *timetm;
long tns;
const struct tm *inacctm;
long ins;
```

Parameters

Input

timetm

A `tm` structure that expresses the local time. On input, `tm_wday` and `tm_yday` are ignored.

tns

Nanoseconds since time component.

inacctm

A `tm` structure that expresses days, hours, minutes, and seconds of inaccuracy. If `tm_yday` is negative, the inaccuracy is considered to be infinite. On input, `tm_mday`, `tm_mon`, `tm_wday`, `tm_isdst`, `tm_gmtoff`, and `tm_zone` are ignored.

ins

Nanoseconds of inaccuracy component.

Output

utc

Resulting binary timestamp.

Description

The **Make Local Time** routine converts a tm structure that expresses local time to a binary timestamp.

The user's environment determines the time zone rule (details are system dependent).

OpenVMS: The user selects a time zone by defining `sys$timezone_rule`, which is created when the `sys$manager:net$configure.com` is run.

UNIX: The user selects a time zone by specifying the time zone environment variable. (The reference page for the `localtime( )` system call provides additional information.)

If the user's environment does not specify a time zone rule, the system's rule is used (details of the rule are system dependent).

OpenVMS: OpenVMS systems do not have a default time zone rule. You must run the `sys $manager:net$configure` procedure to specify a time zone.

UNIX: The rule in `/etc/zoneinfo/localtime` applies.

Additional inputs include nanoseconds since the last second of time and nanoseconds of inaccuracy.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument or invalid results.

Example

See the sample program in the **Examples** section of the `utc_cmpmidtime` routine.

Related Information

Functions: `utc_localtime`

utc_mkreltime

utc_mkreltime — Converts a `tm` structure that expresses relative time to a relative binary timestamp.

Syntax

```
#include <utc.h>

int utc_mkreltime(*utc, *timetm, tns, *inacctm, ins)

   utc_t *utc;
   const struct tm *timetm;
   long tns;
   const struct tm *inacctm;
   long ins;
```

Parameters

Input

timetm

A `tm` structure that expresses a relative time. On input, `tm_wday` and `tm_yday` are ignored.

tns

Nanoseconds since time component.

inacctm

A `tm` structure that expresses seconds of inaccuracy. If `tm_yday` is negative, the inaccuracy is considered to be infinite. On input, `tm_mday`, `tm_mon`, `tm_year`, `tm_wday`, `tm_isdst`, and `tm_zone` are ignored.

ins

Nanoseconds of inaccuracy component.

Output

utc

Resulting relative binary timestamp.

Description

The **Make Relative Time** routine converts a `tm` structure that expresses relative time to a relative binary timestamp. Additional inputs include nanoseconds since the last second of time and nanoseconds of inaccuracy.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument or invalid results.

Example

The following example converts a string relative time in the format (1991-04-01- 12:12:12.12I12.12) to a binary timestamp. This may be part of an input relative timestamp routine, though a real implementation will include range checking.

```
utc_t       utc;
struct tm   tmtime, tminacc;
float       tsec, isec;
double      tmp;
long        tnsec, insec;
int         i, tzhour, tzmin, year, mon;
char        *string;

/*
 *   Try to convert the string...
 */

if(sscanf(string, "%d-%d-%d-%d:%d:%eI%e",
          &year, &mon, &tmtime.tm_mday, &tmtime.tm_hour,
          &tmtime.tm_min, &tsec, &isec) != 7) {

/*
 *   ERROR...
 */
    exit(1);

}

/*
 *   Fill in the fields...
 */

tmtime.tm_year = year - 1900;
tmtime.tm_mon = --mon;
tmtime.tm_sec = tsec;
tnsec = (modf(tsec, &tmp)*1.0E9);
tminacc.tm_sec = isec;
insec = (modf(isec, &tmp)*1.0E9);

/*
 * Convert to a binary timestamp...
 */
```

```
utc_mkreltime(&utc,     /* Out: Resultant binary timestamp      */
              &tmtime,  /* In:  tm struct that represents input */
              tnsec,    /* In:  Nanoseconds from input          */
              &tminacc, /* In:  tm struct that represents inacc */
              insec);   /* In:  Nanoseconds from input          */
```

Related Information

Functions: `utc_reltime`

utc_mkvmsanytime

utc_mkvmsanytime — Converts a binary OpenVMS format time and TDF (expressing the time in an arbitrary time zone) to a binary timestamp.

Syntax

```
#include <utc.h>

int utc_mkvmsanytime(*utc, *timadr, tdf)

   utc_t *utc;
   const long *timadr;
   const long tdf;
```

Parameters

Input

**timadr*

Binary OpenVMS format time.

tdf

Time differential factor to use in conversion.

Output

**utc*

Binary timestamp.

Description

The **Make VMS Any Time** routine converts a binary time in the OpenVMS (Smithsonian) format and an arbitrary TDF to a UTC-based binary timestamp. Because the input and output values are based on different time standards, any input representing a value after A.D. 30,000 returns an error.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument or invalid results.

Example

The following example shows how to convert between OpenVMS format binary timestamps and UTC binary timestamps, while specifying the TDF for each. The TDF value determines the offset from GMT and the local time.

```
/*****
 start example mkvmsanytime,vmsanytime
 *****/
#include <utc.h>

main()
{
struct utc utcTime;
int vmsTime[2];

SYS$GETTIM(vmsTime);    /* read the current time */

/*
 * convert the VMS local time to a UTC, applying a TDF of
 * -300 minutes (the timezone is -5 hours from GMT)
 */
if (utc_mkvmsanytime(&utcTime,vmsTime,-300))
    exit(1);

/*
 * convert UTC back to VMS local time.  A TDF of -300 is applied
 * to the UTC, since utcTime was constructed with that same value.
 * This effectively gives us the same VMS time value we started
 * with.
 */
if (utc_vmsanytime(vmsTime,&utcTime))
    exit(2);
}
/****
  end example
 ****/
```

Related Information

Function: `utc_vmsanytime`

utc_mkvmsgmtime (OpenVMS only)

utc_mkvmsgmtime (OpenVMS only) — Converts a binary OpenVMS format time expressing GMT (or the equivalent UTC) into a binary timestamp.

Parameters

Input

**timadr*

Binary OpenVMS format time representing GMT or the UTC equivalent.

Output

**utc*

Binary timestamp.

Description

The **Make VMS Greenwich Mean Time** routine converts an OpenVMS format binary time representing GMT to a binary timestamp with the equivalent UTC value. Since the input and output values are based on different time standards, any input representing a value after A.D. 30,000 returns an error.

Example

See the sample program in the **Examples** section of the `vmsgmtime` routine.

Related Information

Function: `utc_vmsgmtime`

utc_mkvmslocaltime

utc_mkvmslocaltime — Converts a local binary OpenVMS format time to a binary timestamp, using the host system's time differential factor.

Syntax

```
#include <utc.h>

int utc_mkvmslocaltime(*utc, *timadr)

   const long *timadr;
   utc_t *utc;
```

Parameters

Input

**timadr*

Binary OpenVMS format time expressing local time.

Output

**utc*

Binary timestamp expressing the system's local time.

Description

The **Make VMS Local Time** routine converts a binary OpenVMS format time, representing the local time of the host system, to a binary timestamp. The system's local time value is defined by the time zone rule in `sys$timezone_rule`, which is created by the system configuration process `sys$manager:net$configure.com`.

Notes

If the routine call is made during a seasonal time zone change when the local time is indeterminate, an error is returned. For example, if the time zone change occurs at the current local time of 2:00 A.M. to a new local time of 1:00 A.M., and the routine is called between 1:00 A.M. and 2:00 A.M., it cannot be determined which TDF applies.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument, invalid results, or invalid routine call during a time zone change.

Example

The following example shows how to retrieve the current local time of the system in the binary OpenVMS format, convert the OpenVMS format time to a UTC-based binary timestamp (using the system's TDF), and print an ASCII representation of the binary timestamp.

```
/*********
  start example mkvmslocaltime
 *********/
#include <utc.h>

main()
{
char outstring[UTC_MAX_STR_LEN];
struct utc utcTime;
int vmsTime[2];

SYS$GETTIM(vmsTime);                        /* read current time      */

if (utc_mkvmslocaltime(&utcTime,vmsTime)) /* convert the local time */
    exit(1);                                /* vmsTime to UTC using   */
                                            /* the system tdf.        */
```

Related Information

Function: `utc_vmslocaltime`

utc_mulftime

utc_mulftime — Multiplies a relative binary timestamp by a floating-point value.

Syntax

```
#include <utc.h>

int utc_mulftime(*result, *utc1, factor)

   utc_t *result;
   const utc_t *utc1;
   const double factor;
```

Parameters

Input

utc1

Relative binary timestamp.

factor

Real scale factor (double-precision floating-point) (G format floating-point on VAX systems).

Output

result

Resulting relative binary timestamp.

Description

The **Multiply a Relative Time by a Real Factor** routine multiplies a relative binary timestamp by a floating-point value. Either or both may be negative; the resulting relative binary timestamp has the appropriate sign. The unsigned inaccuracy in the relative binary timestamp is also multiplied by the absolute value of the floating-point value.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument or invalid results. The following example scales and prints a relative time.

Example

The following example scales and prints a relative time.

```
utc_t       relutc, scaledutc;
struct tm   sacledreltm;
char        timstr[UTC_MAX_STR_LEN];

/*
 * Assume relutc contains the time to scale.
 * Scale it by a factor of 17...
 */

utc_multime(&scaledutc,          /* Out: Scaled rel time     */
            &relutc,             /* In:  Rel time to scale   */
            17L);                /* In:  Scale factor        */

utc_ascreltime(timstr,           /* Out: ASCII rel time      */
               UTC_MAX_STR_LEN,  /* In:  Length of input str */
               &scaledutc);      /* In:  Rel time to convert */

printf("%s\n",timstr);

/*
 *   Scale it by a factor of 17.65...
```

```
 */

utc_mulftime(&scaledutc,          /* Out: Scaled rel time     */
             &relutc,             /* In:  Rel time to scale   */
             17.65);              /* In:  Scale factor        */

utc_ascreltime(timstr,            /* Out: ASCII rel time      */
               UTC_MAX_STR_LEN,   /* In:  Input str length    */
               &scaledutc);       /* In:  Rel time to convert */

printf("%s\n",timstr);

/*
 *    Convert it to a tm structure and print it.
 */

utc_reltime(&scaledreltm,         /* Out: Scaled rel tm       */
            (long *)0,            /* Out: Scaled rel nano-sec */
            (struct tm *)0,       /* Out: Scaled rel inacc tm */
            (long *)0,            /* Out: Scd rel inacc nanos */
            &scaledutc);          /* In:  Rel time to convert */

printf("Approximately %d days, %d hours and %d minutes\n",
 scaledreltm.tm_yday, scaledreltm.tm_hour, scaledreltm.tm_min);
```

Related Information

Functions: `utc_multime`

utc_multime

utc_multime — Multiplies a relative binary timestamp by an integer factor.

Syntax

```
#include <utc.h>

int utc_multime(*result, *utc1, factor)

   utc_t *result;
   const utc_t *utc1;
   long factor;
```

Parameters

Input

utc1

Relative binary timestamp.

factor

Integer scale factor.

Output

result

Resulting relative binary timestamp.

Description

The **Multiply Relative Time by an Integer Factor** routine multiplies a relative binary timestamp by an integer. Either or both may be negative; the resulting binary timestamp has the appropriate sign. The unsigned inaccuracy in the binary timestamp is also multiplied by the absolute value of the integer.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument or invalid results.

Example

See the sample program in the **Examples** section of the `utc_mulftime` routine.

Related Information

Functions: `utc_mulftime`

utc_pointtime

utc_pointtime — Converts a binary timestamp to three binary timestamps that represent the earliest, most likely, and latest time.

Syntax

```
#include <utc.h>

int utc_pointtime(*utclp, *utcmp, *utchp, *utc)

   utc_t *utclp;
   utc_t *utcmp;
   utc_t *utchp;
   const utc_t *utc;
```

Parameters

Input

utc

Binary timestamp or relative binary timestamp.

Output

utclp

Lowest (earliest) possible time that the input binary timestamp or shortest possible relative time that the relative binary timestamp can represent.

utcmp

Midpoint of the input binary timestamp or the midpoint of the input relative binary timestamp.

utchp

Highest (latest) possible time that the input binary timestamp or the longest possible relative time that the relative binary timestamp can represent.

Description

The **Point Time** routine converts a binary timestamp to three binary timestamps that represent the earliest, latest, and most likely (midpoint) times. If the input is a relative binary time, the outputs represent relative binary times.

Notes

All outputs have zero inaccuracy. An error is returned if the input binary timestamp has an infinite inaccuracy.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument.

Example

See the sample program in the **Examples** section of the `utc_addtime` routine.

Related Information

Functions: `utc_boundtime`, `utc_spantime`

utc_reltime

utc_reltime — Converts a relative binary timestamp to a `tm` structure.

Syntax

```
#include <utc.h>

int utc_reltime(*timetm, *tns, *inacctm, *ins, *utc)

   struct tm *timetm;
   long *tns;
   struct tm *inacctm;
   long *ins;
   const utc_t *utc;
```

Parameters

Input

utc

Relative binary timestamp.

Output

timetm

Relative time component of the relative binary timestamp. The field `tm_mday` returns a value of –1 and the fields `tm_year` and `tm_mon` return values of zero. The field `tm_yday` contains the number of days of relative time.

tns

Nanoseconds since time component of the relative binary timestamp.

inacctm

Seconds of inaccuracy component of the relative binary timestamp. If the inaccuracy is finite, then `tm_mday` returns a value of –1 and `tm_mon` and `tm_year` return values of zero. The field `tm_yday` contains the inaccuracy in days. If the inaccuracy is infinite, all `tm` structure fields return values of –1.

ins

Nanoseconds of inaccuracy component of the relative binary timestamp.

Description

The **Relative Time** routine converts a relative binary timestamp to a `tm` structure. Additional returns include nanoseconds since time and nanoseconds of inaccuracy.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument or invalid results.

Example

See the sample program in the **Examples** section of the `utc_mulftime` routine.

Related Information

Functions: `utc_mkreltime`

utc_spantime

utc_spantime — Given two (possibly unordered) binary timestamps, returns a single UTC time interval whose inaccuracy spans the two input binary timestamps.

Syntax

```
#include <utc.h>

int utc_spantime(*result, *utc1, *utc2)

   utc_t *result;
   const utc_t *utc1;
```

```
const utc_t *utc2;
```

Parameters

Input

utc1

Binary timestamp.

utc2

Binary timestamp.

Output

result

Spanning timestamp.

Description

Given two binary timestamps, the **Span Time** routine returns a single UTC time interval whose inaccuracy spans the two input timestamps (that is, the interval resulting from the earliest possible time of either timestamp to the latest possible time of either timestamp).

Notes

The *tdf* in the output UTC value is copied from the *utc2* input. If either input binary timestamp has an infinite inaccuracy, an error is returned.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument.

Example

The following example computes the earliest and latest times for an array of 10 timestamps.

```
utc_t               time_array[10], testtime, earliest, latest;
int                 i;

/*
 *   Set the running timestamp to the first entry...
 */

testtime = time_array[0];

for (i=1; i<10; i++) {

   /*
    *   Compute the minimum and the maximum against the next
    *   element...
    */
```

```
utc_spantime(&testtime,       /* Out: Resultant interval        */
             &testtime,       /* In:  Largest previous interval */
             &time_array[i]); /* In:  Element under test        */
}

/*
 *   Compute the earliest possible time...
 */

utc_pointtime(&earliest,    /* Out: Earliest poss time in array */
              (utc_t *)0,   /* Out: Midpoint                    */
              &latest,      /* Out: Latest poss time in array   */
              &testtime);   /* In:  Spanning interval           */
```

Related Information

Functions: `utc_boundtime`, `utc_gettime`, `utc_pointtime`

utc_subtime

utc_subtime — Computes the difference between two binary timestamps that express either an absolute time and a relative time, two relative times, or two absolute times.

Syntax

```
#include <utc.h>

int utc_subtime(*result, *utc1, *utc2)

   utc_t *result;
   const utc_t *utc1;
   const utc_t *utc2;
```

Parameters

Input

utc1

Binary timestamp or relative binary timestamp.

utc2

Binary timestamp or relative binary timestamp.

Output

result

Resulting binary timestamp or relative binary timestamp, depending on the operation performed:

- *absolute time absolute time* = **relative time**
- *relative time relative time* = **relative time**
- *absolute time relative time* = **absolute time**

- *relative time absolute time* is undefined. See **No**.

Description

The **Subtract Time** routine subtracts one binary timestamp from another.

The resulting timestamp is *utc1* minus *utc2*. The inaccuracies of the two input timestamps are combined and included in the output timestamp. The TDF in the first timestamp is copied to the output.

Notes

Although no error is returned, do **not** use the combination *relative time*

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument or invalid results.

Example

See the sample program in the **Examples** section of the `utc_binreltime` routine.

Related Information

Functions: `utc_addtime`

utc_vmsanytime (OpenVMS only)

utc_vmsanytime (OpenVMS only) — Converts a binary timestamp to a binary OpenVMS format time. The TDF encoded in the input timestamp determines the TDF of the output.

Syntax

```
#include <utc.h>

int utc_vmsanytime(*timadr, *utc)

   const utc_t *utc;
   long *timadr;
```

Parameters

Input

**utc*

Binary timestamp.

Output

**timadr*

Binary OpenVMS format time.

Description

The **VMS Any Time** routine converts a UTC-based binary timestamp to a 64-bit binary time in the OpenVMS (Smithsonian) format. Because the input and output values are based on different time standards, any input representing a value before the Smithsonian base time of November 17, 1858 returns an error.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument or invalid results.

Example

See the sample program in the **Examples** section of the `mkvmsanytime` routine.

Related Information

Function: `utc_mkvmsanytime`

utc_vmsgmtime (OpenVMS only)

utc_vmsgmtime (OpenVMS only) — Converts a binary timestamp to a binary OpenVMS format time expressing GMT or the equivalent UTC.

Syntax

```
#include <utc.h>

int utc_vmsgmtime(*timadr, *utc)

   const utc_t *utc;
   long *timadr;
```

Parameters

Input

**utc*

Binary timestamp to be converted.

Output

**timadr*

Binary OpenVMS format time representing GMT or the UTC equivalent.

Description

The **OpenVMS Greenwich Mean Time** routine converts a UTC-based binary timestamp to a 64-bit binary time in the OpenVMS (Smithsonian) format. The OpenVMS format time represents Greenwich Mean Time or the equivalent UTC. Because the input and output values are based on different time

standards, any input representing a value before the Smithsonian base time of November 17, 1858 returns an error.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument or invalid results.

Example

The following example shows the following time zone and time format conversions:

1. Retrieve a binary timestamp representing UTC with the sys$getutc system service.
2. Convert the binary timestamp to a OpenVMS format binary time representing GMT
3. Convert the OpenVMS format binary time representing GMT back to a UTC-based binary timestamp with a TDF of 0 (zero)
4. Convert the UTC-based binary time to a binary OpenVMS format time representing the local time; use the TDF from the system

```
/*****
  start example vmsgmtime, mkvmsgmtime, vmslocaltime
 *****/
#include <utc.h>

main()
{
int status;
struct utc utcTime;
int vmsTime[2];

 if (!((status=SYS$GETUTC(&utcTime))&1))
     exit(status);              /* read curr time as a utc */

/*
 * convert the utcvalue into a vms time, with a timezone of 0
 * (GMT).  Printing the resultant vmstime yields the time at
 * the prime meridian in Greenwich, not (necessarily) the
 * local time.
 */
 if (utc_vmsgmtime(vmsTime,&utcTime))
    exit(1);

/*
 * Convert the vmstime (which is in GMT) to a utc
 */
if (utc_mkvmsgmtime(&utcTime, vmsTime))
    exit(2);

/*
 *  convert the UTC to local 64-bit time.  Note that this is the
 *  value we would have read if we had issued a 'SYS$GETTIM' in
 *  the initial statement.
 */
```

```
if (utc_vmslocaltime(vmsTime, &utcTime))
    exit(3);
}
/*****
 end example
     *****/
```

Related Information

Function: `utc_mkvmsgmtime`

utc_vmslocaltime (OpenVMS only)

utc_vmslocaltime (OpenVMS only) — Converts a binary timestamp to a local binary OpenVMS format time, using the host system's time differential factor.

Syntax

```
#include <utc.h>

int utc_vmslocaltime(*timadr, *utc)

   const utc_t *utc;
   long *timadr;
```

Parameters

Input

**utc*

Binary timestamp.

Output

**timadr*

Binary OpenVMS format time expressing local time.

Description

The **VMS Local Time** routine converts a binary timestamp to a binary OpenVMS format time; the output value represents the local time of the host system. The system's offset from UTC and the local time value are defined by the time zone rule in `sys$timezone_rule`, which is created by the system configuration process `sys$manager:net$configure.com`.

Return Values

0	Indicates that the routine executed successfully.
-1	Indicates an invalid time argument or invalid results.

Example

See the sample program in the **Examples** section of the `vmsgmtime` routine.

Related Information

Function: `utc_vmsmklocaltime`

Chapter 3. Using the DECdts API Routines

This chapter contains a C programming example showing a practical application of the DECdts API programming routines. The program performs the following actions:

- Prompts the user to enter time coordinates.
- Stores those coordinates in a `tm` structure.
- Converts the `tm` structure to a `utc` structure.
- Determines which event occurred first.
- Determines if Event 1 may have caused Event 2 by comparing the intervals.
- Prints out the `utc` structure in ISO text format.

```
#include <time.h>   /* time data structures                    */
#include <utc.h>    /* utc structure definitions               */

void ReadTime();
void PrintTime();

/*
 * This program requests user input about events, then prints out
 * information about those events.
 */

main()
{
    struct utc event1,event2;
    enum utc_cmptype relation;

    /*
     * Read in the two events.
     */

    ReadTime(&event1);
    ReadTime(&event2);

    /*
     * Print out the two events.
     */

    printf("The first event is : ");
    PrintTime(&event1);
    printf("\nThe second event is : ");
    PrintTime(&event2);
    printf("\n");

    /*
     * Determine which event occurred first.
     */
    if (utc_cmpmidtime(&relation,&event1,&event2))
```

```
        exit(1);

    switch( relation )
    {
        case utc_lessThan:
        printf("comparing midpoints: Event1 < Event2\n");
        break;
        case utc_greaterThan:
        printf("comparing midpoints: Event1 > Event2\n");
        break;
        case utc_equalTo:
        printf("comparing midpoints: Event1 == Event2\n");
        break;
        default:
        exit(1);
        break;
    }

    /*
     * Could Event 1 have caused Event 2?  Compare the intervals.
     */

    if (utc_cmpintervaltime(&relation,&event1,&event2))
        exit(1);

    switch( relation )
    {
        case utc_lessThan:
        printf("comparing intervals: Event1 < Event2\n");
        break;
        case utc_greaterThan:
        printf("comparing intervals: Event1 > Event2\n");
        break;
        case utc_equalTo:
        printf("comparing intervals: Event1 == Event2\n");
        break;
        case utc_indeterminate:
        printf("comparing intervals: Event1 ? Event2\n");
        default:
        exit(1);
        break;
    }

}

/*
 * Print out a utc structure in ISO text format.
 */

void PrintTime(utcTime)
struct utc *utcTime;
{

    char   string[50];

    /*
     * Break up the time string.
     */
```

```
    if (utc_ascgmtime(string,      /* Out: Converted time    */
                      50,          /* In:  String length     */
                      utcTime))    /* In:  Time to convert   */
        exit(1);
    printf("%s\n",string);
}

/*
 * Prompt the user to enter time coordinates.  Store the
 * coordinates in a tm structure and then convert the
 * tm structure to a utc structure.
 */

void ReadTime(utcTime)
struct utc *utcTime;
{
struct tm tmTime,tmInacc;

    (void)memset((void *)&tmTime,  0,sizeof(tmTime));
    (void)memset((void *)&tmInacc, 0,sizeof(tmInacc));
    (void)printf("Year? ");
    (void)scanf("%d",&tmTime.tm_year);
    tmTime.tm_year -= 1900;
    (void)printf("Month? ");
    (void)scanf("%d",&tmTime.tm_mon);
    tmTime.tm_mon -= 1;
    (void)printf("Day? ");
    (void)scanf("%d",&tmTime.tm_mday);
    (void)printf("Hour? ");
    (void)scanf("%d",&tmTime.tm_hour);
    (void)printf("Minute? ");
    (void)scanf("%d",&tmTime.tm_min);
    (void)printf("Inacc Secs? ");
    (void)scanf("%d",&tmInacc.tm_sec);

    if (utc_mkanytime(utcTime,
                      &tmTime,
                      (long)0,
                      &tmInacc,
                      (long)0,
                      (long)0))
        exit(1);

}
```

OpenVMS: Assume the preceding program is named `compare_events.c`. To compile and link the program on a DECnet-Plus for OpenVMS system, enter the following command:

```
$ cc compare_events.c/output=compare_events.obj
$ link compare_events.obj, sys$input:/options Return
sys$library:dtss$shr.exe/share Ctrl-z
$
```

UNIX: To compile and link the program on a DECnet-Plus for UNIX system, enter the following command:

```
# cc compare_events.c -lutc -o compare_events
```

#

Chapter 4. Time-Provider Interface

This chapter describes the Distributed Time Service (DECdts) time-provider interface (TPI) for DECdts software on systems running the DECnet-Plus for UNIX and DECnet-Plus for OpenVMS operating systems. The chapter begins with a brief overview of the TPI and explains how to use external time-providers with DECdts; the rest of the chapter describes the data structures and message protocols that make up the TPI.

Coordinated Universal Time (UTC) is disseminated throughout the world by various standards organizations. Several manufacturers supply devices that can acquire UTC time values via radio, satellite, or telephone; these devices can then provide standardized time values to computer systems. Typically, one of these devices is connected to a computer system; a process runs on the system and interacts with the device to interpret signals and translate them to time values, which can either be displayed or be provided to a server process running on a connected system.

To synchronize its system clock with UTC using an external time-provider device, a DECdts server needs a software interface to the device to periodically obtain UTC. This interface is the intermediary between the DECdts server and external time-provider processes. The server requires the interface to obtain UTC time values and to determine the associated inaccuracy of each value. The interface between the DECdts server and the time-provider process is called the Time-Provider Interface.

The remainder of this chapter describes the TPI and its attendant processes in detail. The following section describes the control flow between the DECdts server process, the TPI, and the time-provider process.

4.1. General TPI Control Flow

When you use a time-provider with a system running DECdts, an external time- provider is implemented as an independent process that exchanges messages with DECdts (mailbox messages with OpenVMS, socket messages with UNIX). The DECdts server and the time-provider process (TP process) must both be running on the same system. The DECdts server initiates communication with the TP process by sending a connection request to the TP process.

OpenVMS: At each system synchronization, a DECdts server contacts the TP process by issuing a connect request to a well known OpenVMS mailbox, which is identified by the system logical name `DTSS$_TSTP_MBX` (also referred to as the **request mailbox**).

UNIX: The connect request is issued to a well known UNIX domain socket, which is identified by the name `/usr/var/tmp/dssTSTP` (also referred to as the **request socket**).

If the TP process is active, it immediately acknowledges the connect request and writes the initial control response message to one of the following:

OpenVMS: A second well known OpenVMS mailbox, which is identified by the system logical name `DTSS$_TPTS_MBX` (also referred to as the **response mailbox**).

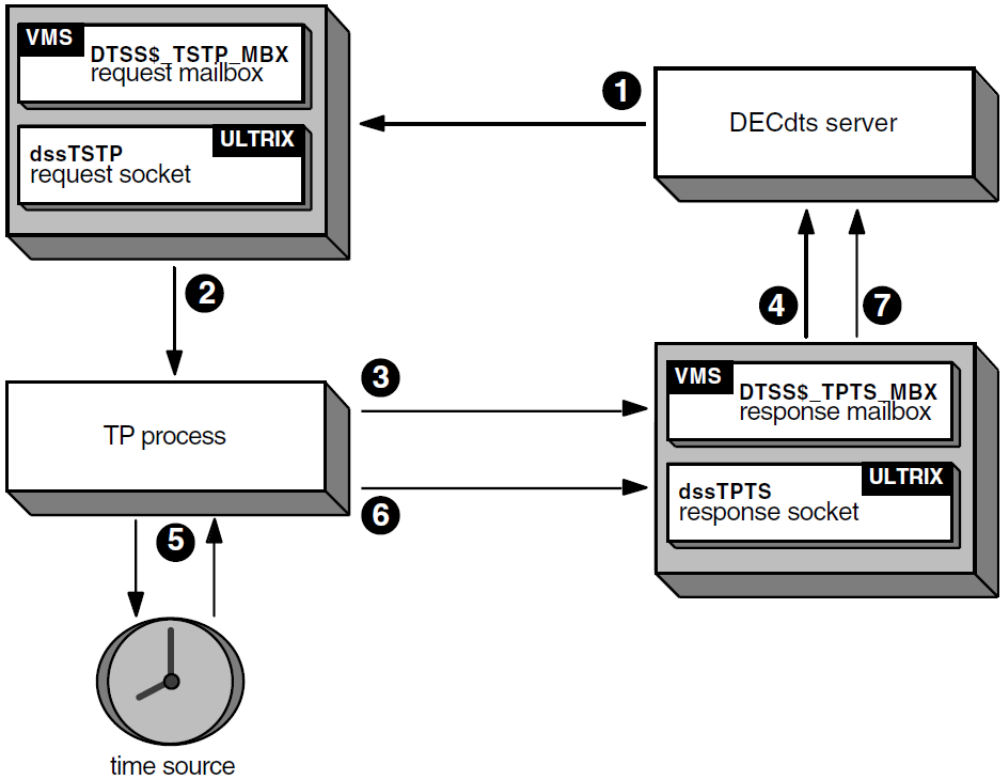
UNIX: A second UNIX domain socket, which is identified by the name `/usr/var/tmp/dssTPTS` (also referred to as the **response socket**).

When the DECdts server is enabled on the system, it creates this mailbox/socket. If the DECdts server cannot write its request message to the request mailbox /socket (because the TP process is not available) or does not immediately receive a control message from the TP process, the DECdts server synchronizes with other servers instead of with the external time-provider.

If the initial message exchange is successful, the DECdts server waits for a second response message (data message) that contains the timestamp values read from the external time source. The length of time the server process waits for the data message is specified by the TP process in the initial control message. When the TP process writes a data message to the response mailbox/socket, the DECdts server uses the timestamp in the data message to complete its synchronization.

Figure 4.1 shows the message exchange between the DECdts server and the TP process.

Figure 4.1. DECdts Server/TP Process Message Exchange



The following steps describe the process illustrated in Figure 4.1:

1. The DECdts server sends a request message to the request mailbox/socket (DTSS$_TSTP_MBX or dssTSTP).
2. The TP process receives the message from the request mailbox/socket.
3. The TP process sends the initial response message (control message) to the response mailbox/socket (DTSS$_TPTS_MBX or dssTPTS).
4. The DECdts server receives the control message, extracts three data fields (next poll time, TP response timeout, noclockset) and waits for the arrival of the data message.
5. The TP process polls its external time source (the time-provider hardware).
6. The TP process stores the UTC time values it obtains from the external time source in a data message and then sends the message to the response mailbox/socket.
7. The DECdts server reads the data message from the response mailbox/socket and extracts the timestamps to complete a synchronization.

Section 4.2 describes the message types that are exchanged by the DECdts server and the TP process during the previous sequence.

4.2. Message Types

The DECdts TPI uses request and response messages/sockets to exchange information between the DECdts server and the TP process. The following sections describe the message functions, the functions of the settable fields in each message, and the range of settings for each field. The definitions for the TP process message types can be found in `dtssprovider.h`. See Section 4.7 for additional information about these definitions.

4.2.1. The Time Request Message

Time request messages are issued by the DECdts server to initiate a synchronization with the TP process. Each message contains the current synchronization serial number and a TPI version number field.

The TPI version number field defines the major and minor version numbers of the TPI:

- The TPI major version subfield must be set to K_TPI_MAJOR_VERS.
- The minor version subfield must be set to K_TPI_MINOR_VERS.

The TP process ignores any message with a version number field that does not contain the correct TPI version number.

4.2.2. Time Response Messages

The TPI uses two types of time response messages: control messages and data messages. The TP process sends both types of messages to the response mailbox/socket when replying to a request message from the DECdts server. The following data fields are common to both time response messages:

- **Synchronization ID**

  Contains the current DECdts synchronization serial number. The current synchronization ID is obtained from the *synchID* field of the request message from the DECdts server.

- **Time-Provider Status**

  Contains either the value `K_TPI_SUCCESS(1)` or `K_TPI_FAILURE(0)`. If the TP process attempts to terminate a synchronization, it writes a response message to the response mailbox/socket with a status of `K_TPI_FAILURE`; otherwise, the status field contains `K_TPI_SUCCESS`.

- **Message Type**

  Distinguishes the two types of response messages. The message type field contains one of two values; `K_TPI_TIME_MESSAGE` specifies a data message, and `K_TPI_CTL_MESSAGE` specifies a control message.

- **TPI Versions**

  Contains the major and minor version numbers of the TPI. The major version subfield must be set to K_TPI_MAJOR_VERS; the minor version subfield must be set to K_TPI_MINOR_VERS. Any message received by the DECdts server is ignored.

The control and data response messages also have unique fields. Section 4.2.2.1 describes control messages; Section 4.2.2.2 describes data messages.

4.2.2.1. The Control Message

The TP process initially writes a control message to the response mailbox/socket in reply to a request message from the DECdts server. Control messages contain the following fields:

- **Next Poll Value**

  Contains an integer in the range `K_MIN_NEXTPOLL` to `K_MAX_NEXTPOLL`. If the current synchronization is successful, DECdts issues the next request message in *nextPoll* seconds.

- **Timeout Value**

  Contains an integer in the range `K_MIN_TIMEOUT` to `K_MAX_TIMEOUT`. DECdts waits a maximum of *timeout* seconds for the arrival of a data message before asserting that the TP process is no longer available.

- **No Set Value**

  Specifies whether or not the service is allowed to alter the system clock. If *noSet* is set to the value 0x01 (true), the DECdts server does not adjust or set the clock during the current synchronization. DECdts does, however, assert the inaccuracy returned in the data message.

4.2.2.2. The Data Message

The TP process writes a data message to the response mailbox/socket within *timeout* seconds after it writes a control message. The data message contains two fields:

- The timestamp array
- The timestamp count

The timestamp array contains one or more timestamps. Each timestamp consists of three *utc* time values:

- The system clock time immediately before the TP process polls the external time source. (The TP process normally obtains the time from the `utc_gettime` DECdts API routine.)
- The time value returned to the TP process by the external time source.
- The system clock time immediately after the external time source was read. (The TP process again obtains the time from the `utc_gettime` DECdts API routine.)

The other unique data message field contains the timestamp count. The timestamp count is an integer in the range `K_MIN_TIMESTAMPS` to `K_MAX_TIMESTAMPS`. The integer equals the number of timestamps contained in the data message.

4.3. Interprocess Communication

Interprocess communication between the DECdts server and the TP process is accomplished by using two OpenVMS mailboxes or two UNIX domain sockets.

4.3.1. Interprocess Communications on OpenVMS Systems

The TP process creates the request mailbox (`DTSS$_TSTP_MBX`) and the DECdts server creates the response mailbox (`DTSS$_TPTS_MBX`). The time-provider uses the `SYS$CREMBX` system service to create its mailbox. The arguments to the `SYS$CREMBX` system service follow:

```
SYS$CREMBX(
    prmflg = 1,                          /* permanent mail box   */
    maxmsg = sizeof( TPreqMsg ),         /* size of each message */
    bufquo = 2 * sizeof( TPreqMsg ),     /* allow a maximum of 2 */
                                         /* messages at any time */
    promsk = 0xFF00, /* no access to world */
    /* w:xxx=111=0xF */
    /* g:xxx=111=0xF */
    /* o:lprw=0000=0x0 */
    /* s:lprw=0000=0x0 */
    acmode = PSL$C_USER /* nonprivileged access mode */
    lognam = "DTSS$_TSTP_MBX" /* well-known logical name */
    )
```

The DECdts server attempts to assign a channel to this mailbox by using the mailbox's well-known logical name. The TP process only reads and never writes to this mailbox. The DECdts server only writes to this mailbox. The TP process uses the `sys$assign` system service command to attach to the mailbox created by the DECdts server.

The arguments to the `sys$assign` service follow:

```
SYS$ASSIGN(
    devnam = "DTSS$_TPTS_MBX",
    channel = (specified by user),
    acmode = PSL$C_USER,
    mbxnam = 0,
           )
```

The TP process writes data to the response mailbox; it must never attempt to read data from response mailbox.

4.3.2. Interprocess Communications on UNIX Systems

A **communication domain** is identified by a manifest constant defined in the file `<sys/socket.h>`. UNIX domain sockets (`AF_UNIX`) are used for communication within the system. The TP process creates the request socket (`dssTSTP`); the DECdts server creates the response socket (`dssTPTS`). Both sockets are of the socket type `SOCK_STREAM` (stream sockets), which are full-duplex, reliable byte streams that have no record boundaries. Stream sockets are available if your system includes TCP/IP.

You can create UNIX sockets with the socket call. This call yields an unconnected socket descriptor, which must be made ready to accept connections by binding it to a name within the communications domain. The bind call accomplishes this process. Once the socket is bound to a name in the domain, the socket must listen for connections through the listen call. When a connection is requested from the DECdts server, the TP process must be ready to accept the connection. The arguments to these calls follow:

```
socket_id = socket (AF_UNIX,      /* UNIX domain path names */
                    SOCK_STREAM,  /* socket type            */
                    0             /* protocol - set to zero */
                    );

bind (socket_id,     /* Descriptor that refers to the created socket */
      sock_name,     /* Name that is assigned to the created socket; */
                     /* in the case of the TP: /usr/var/tmp/dssTSTP  */
      sizeof(sock_name)
      );
```

```
listen (socket_id,   /* Descriptor that refers to the created socket */
        back_log     /* Maximum number of pending connections in     */
        );           /* the queue */

accept (socket_id,   /* Descriptor that refers to the created socket */
        address,     /* Address of the connecting entity             */
        address_len  /* Address length                               */
        );
```

The DECdts server makes a connection to the request socket (socket created by the TP process) by issuing a connect call. The DECdts server only writes to this socket; the TP process should only read (never write to) the socket. Conversely, the TP process communicates with the DECdts server by connecting to and then sending messages to the response socket. The TP process only writes to (never reads from) this socket. The arguments to the connect call follow:

```
connect (socket_id,  /* Descriptor that refers to the created socket */
         sock_name,  /* Name of the socket to establish connection;  */
                     /* in the case of the TP: /usr/var/tmp/dssTPTS  */
         sizeof(sock_name)
         );
```

4.4. Time-Provider Algorithm

The algorithm to create a generic time-provider follows:

1. Create the request mailbox/socket (`DTSS$_TSTP_MBX` or `dssTSTP`).

2. Perform the step that corresponds to your operating system:

   OpenVMS: Post a synchronous read to the request mailbox. The TP process remains in LEF state until the DECdts server writes a request to the mailbox.

   UNIX: Issue a connect system call to connect to the request socket. If the connection is unsuccessful, then exit the program with an error.

3. Perform the step that corresponds to your operating system:

   OpenVMS: The DECdts server writes a request message to the request mailbox. The outstanding synchronous read completes. If the TPI version number is correct, accept the message; otherwise return to step 2, ignoring the received message.

   UNIX: Issue a select system call to the TSTP socket. When the selection is completed, issue an accept system call to respond to the connection request from the DECdts server.

4. Perform the step that corresponds to your operating system:

   OpenVMS: Assign a channel to the response mailbox using its well known logical name DTSS $_TPTS_MBX.

   UNIX: Post a (synchronous) blocking read to the TSTP socket and wait for the request message from the DECdts server.

5. Initialize a control message by setting:

   - The TPI version number field to the appropriate value (K_TPI_MAJOR_VERS, K_TPI_MINOR_VERS).

- The time-provider status to K_TPI_SUCCESS.
- The synchronization ID equal to *synchId* (from the request message).
- The variables *nextPoll*, *timeout*, and *noSet* to valid integer values.

6. Perform the step that corresponds to your operating system:

   OpenVMS: Write the control message to the response mailbox using an asynchronous write.

   UNIX: Write the TP process control message to the response socket.

7. Read the system time using the `utc_gettime` DECdts API routine.
8. Poll the external time source, reading a UTC value. Convert the time value to a binary timestamp using the API.
9. Read the system time using the `utc_gettime` DECdts API routine.
10. Repeat steps 7, 8, and 9 between `K_MIN_TIMESTAMPS` times to `K_MAX_TIMESTAMPS` times.
11. Initialize a data message using the timestamps and the correct TPI version numbers.
12. If steps 7, 8, or 9 return erroneous data, initialize the TP status field (*TPstatus*) of the data message to `K_TPI_FAILURE`; otherwise, initialize the data message timestamps.
13. Perform the step that corresponds to your operating system:

    OpenVMS: Write the data message to the DTSS$_TPTS_MBX mailbox.

    UNIX: Write the data message to the response socket and issue a close system call to close all interprocess communication connections to the TSTP and TPTS sockets. Do not delete the TSTP socket.

14. Go to step 2 (loop forever).

4.5. Time Server (DECdts Server Process) Algorithm

The time server algorithm follows:

1. At startup time, create the response mailbox/socket.
2. At synchronization time, attempt to connect to the response mailbox/socket, assumed to have been created by the TP process. If the connection attempt fails, synchronize with peer servers. Otherwise continue.
3. Initialize a request message with the current synchronization serial ID and correct time-provider interface (TPI) version number, then send the message to the request mailbox/socket.
4. Wait for a control message response from the TP process. If no message arrives within the elapsed time specified by the `LAN_QUERY_TIMEOUT` DECdts management parameter, synchronize with peer servers and ignore any subsequent TP process messages. Otherwise, go to step 5.
5. Read the arriving control message and verify the following:

- The message type (it should not be a data message).
- The state of the TP process is `K_TPI_SUCCESS`.
- The current synchronization ID matches *synchID*.
- The TPI version numbers are correct.

If any values are incorrect, ignore this message and go to step 4.

6. Wait for a data message response from the TP process. If no message arrives within the elapsed time specified by the control message (*timeout*), then synchronize with peer servers. Schedule the next synchronization based on the applicable DECdts management parameters, ignoring *nextPoll*.
7. When the next message arrives, read the message type to verify that it is a data message. Also verify that the state of the TP process is K_TPI_SUCCESS and that the TPI version numbers are correct; otherwise, synchronize with peer servers and schedule the next synchronization as in step 6.
8. Extract the timestamps from the data message and synchronize using the timestamps.
9. Close all interprocess communication (IPC) connections with the DECdts server. Do not delete the DTSS$_TPTS_MBX mailbox or the TPTS socket.
10. Schedule the next synchronization time by adding the value of *nextPoll* seconds to the current time. At the next synchronization, go to step 2.

4.6. Running the Time-Provider Process

OpenVMS: The TP process and the DECdts server must both be in the same UIC group. Only processes in the DECdts server's process group can write to the response mailbox.

UNIX: Both the DECdts server and the TP process must run on the same node and have root privileges. The response and request sockets are created such that only root can write to them.

Restricting writes prevents unauthorized users from supplying incorrect times to the DECdts server process and from sending requests to the time-provider. The TP process can always exit without affecting the DECdts server. The DECdts server dynamically reestablishes communications with the TP process.

4.7. Time-Provider Interface, User-Accessible Definitions

The following constant definitions written in the ANSI C programming language define the time ranges (in seconds) for time-provider (TP) control parameters.

OpenVMS: The constant definitions are in the following file:

```
sys$common:[syshlp.examples.dtss]dtss$provider.h
```

UNIX: The constant definitions are in the file `/usr/include /dtssprovider.h`.

```
/*
 *   Valid range for NextPoll. If a time-provider exists, it must be
 *   polled within a 31-day interval.
 */
```

```
#define K_MAX_TP_POLL (31*24*60*60)   /* Maximum 31 days to the next */
                                      /* next time-provider poll.    */

#define K_MIN_TP_POLL (1)             /* Minimum 1 second between    */
                                      /* time-provider polls.        */

/*
 * Valid range for TimeOut...
 * The DECdts server process waits a maximum of 5 minutes for a data
 * message from the TP process to arrive.
 */

#define K_MAX_TP_TMO (5*60)          /* Maximum 5 minutes to wait for */
                                     /* the TP process to respond.    */

#define K_MIN_TP_TMO (1)             /* Minimum 1 second to wait for  */
                                     /* the TP process to respond.    */
```

The following constant definition limits the number of timestamp triplets the TP process can transmit:

```
/*
 * Maximum number of time stamp triplets returned by the
 * TP process...
 */

#define K_MIN_TIMESTAMPS     1
#define K_MAX_TIMESTAMPS     6

/*
 * TPI version numbers...
 */

#define K_TPI_MAJOR_VERS     1
#define K_TPI_MINOR_VERS     0
```

The time-provider process message types are defined by the following definitions written in the ANSI C language.

```
/*
 *   The status of the TP process is either K_TPI_SUCCESS or
 *   K_TPI_FAILURE...
 */

#define   K_TPI_FAILURE   0
#define   K_TPI_SUCCESS   1

/*
 *   Two types of messages...
 *   - Control messages (TPctlMessage)
 *   - Time or Data messages (TPtimeMessage)
 */

#define   K_TPI_TIME_MESSAGE   0
#define   K_TPI_CTL_MESSAGE    1

/*
 *   DECdts version identifier...
 */
```

```
typedef struct VersionType
{
    unsigned short dtss_major;   /* major version */
    unsigned short dtss_minor;   /* minor version */
} VersionType;

/*
 *   A single time stamp...
 *   Contains a reading of the local clock just before the external
 *   time source is queried, the UTC value returned by the external
 *   time source, and a reading of the local clock just after the
 *   external time source is queried.
 */

typedef struct TimeResponseType
{
   struct utc beforeTime;  /* local clk just before getting UTC */
   utc TPtime;             /* external source UTC               */
   utc afterTime;          /* local clk just after getting UTC  */
} TimeResponseType;

/*
 *  TP process control message type...
 *  The initial message returned by the TP process in response to
 *  a time service request.
 */

typedef struct TPctlMsg
{
   unsigned long nextPoll;
   unsigned long timeout;
   unsigned long noSet;
} TPctlMsg;

/*
 *   TP process data message type...
 *   The time stamp values returned by the TP process after it sends
 *   its initial response.
 */

typedef struct TPtimeMsg
{
   unsigned long timeStampCount;
   TimeResponseType timeStampList[K_MAX_TIMESTAMPS];
} TPtimeMsg;

/*
 *   TP process response message...
 *   Contains either a control message or a data message.  Issued by
 *   the TP process, directing the DECdts server to transmit data or
 *   control information.
 *
 *   TPI Control Message (304 bytes) :
 *
 * 31 0
 * +--------------------+--------------------+
 * | TPI Minor Version  | TPI Major Version  |
```

```
 * +----------------------------------------+
 * |    Message Type    |Time-Provider Status|
 * +----------------------------------------+
 * |           Synchronization ID            |   TPI Control Message
 * +----------------------------------------+
 * |              Next Poll Delta            |
 * +----------------------------------------+
 * |             Message Time Out            |
 * +----------------------------------------+
 * |                    NoSet                |
 * +----------------------------------------+
 * |                                         |
 * v         Not Used                        v
 *
 * TPI Data Message (304 bytes) :
 * 31                                        0
 * +--------------------+-------------------+
 * |     TPI Minor Vers | TPI Major Vers    |
 * +----------------------------------------+
 * |       Message Type |Time-Provider Status|
 * +----------------------------------------+
 * |             Synchronization ID          |   TPI Time Message
 * +----------------------------------------+
 * |              Time Stamp Count           |
 * +----------------------------------------+
 * |                                         |
 * |                                         |   Time Stamp One,
 * |                                         |   48 bytes
 * |                                         |     .
 * +-----------------------------------------+     .
 * |                                         |     .
 * v                                         v     .
 *
 * |                                         |   Time Stamp Six,
 * +-----------------------------------------+   48 bytes
 *
 *                                           Total = 48 * 6 = 288 bytes
 *                                           in timestamp/data portion
 *
 * A single Time Stamp (48 bytes):
 *
 * 128                                       0
 * +----------------------------------------+
 * |               Before Time               |
 * +----------------------------------------+
 * |               TP time                   |
 * +----------------------------------------+
 * |               After Time                |
 * +----------------------------------------+
 */

typedef struct TPrspMsg
{
    VersionType TPIversion;    /* Time-provider major & minor versions */
    unsigned short status;     /* Status of the TP process             */
    unsigned short TPmsgType;  /* Message Type: control or data        */
    unsigned long TPsyncID;    /* Synchronization Serial Number        */
    union
```

```
    {
        TPctlMsg TPctlMsg;     /* Control message data  */
        TPtimeMsg TimeMsg;     /* Data message data     */
    } TPdata;
} TPrspMsg;

/*
 * Request message sent from the DECdts server process to the
 * TP process.
 */

typedef struct TPreqMsg
{
    VersionType TPIversion;  /* TPI major, minor version number       */
    unsigned long TPsyncID;  /* service synchronization Serial Number */
} TPreqMsg;

/*
 * TPI Request Message : 8 bytes.
 *
 * 31 0
 * +--------------------+--------------------+
 * |     TPI Minor Vers | TPI Major Vers     |
 * +----------------------------------------+
 * |           Synchronization ID            |   TPI Time Message
 * +----------------------------------------+
 */
```

4.8. Sample Time-Provider Programs and External Time-Provider Sources

OpenVMS: See `sys$common:[syshlp.examples.dtss]` for examples of time-provider programs you can use with various types of external time-provider devices.

UNIX: See `/usr/examples/dtss` for examples of time-provider programs you can use with various types of external time-provider devices.

The *DECdts Management* manual provides additional information about commercial sources of external time-provider devices.

Table 4.1 and Table 4.2 list the time-provider programs and related time-provider hardware/software suppliers that are currently available for DECnet-Plus for OpenVMS and for DECnet-Plus for UNIX systems.

Table 4.1. Time-Provider Programs and Related Time-Provider Suppliers (DECnet-Plus for OpenVMS Systems)

File Name	Related Supplier	Time-Provider Type
`dtss$provider_acts.c`	U.S. NIST (North America)	Time-provider program for data communications
`dtss$provider_acts.com`	U.S. NIST (North America)	Command procedure for the ACTS time-provider program

File Name	Related Supplier	Time-Provider Type
dtss$provider.c	Traconex, Spectracom, Heath, and Hopf (North America and Europe)	Time-provider program for RF receivers

Table 4.2. Time-Provider Programs and Related Time-Provider Suppliers (DECnet-Plus for UNIX Systems)

File Name	Related Supplier	Time-Provider Type
dtss_acts_provider.c	U.S.NIST (North America)	Time-provider program for data communications
dtss_spectracom_provider.c	Spectracom (North America and Europe)	Time-provider program for RF receiver
dtss_traconex_provider.c	Traconex (North America)	Time-provider program for RF receiver
dtss_hopf_provider.c	Hopf (Europe)	Time-provider program for RF receiver
dtss_ntp_provider.c	Various	Time-provider program for Internet Network Time Protocol
dtss_null_provider.c	Digital	Local server clock