Common Calendar Enhanced Timestamp

Common Calendar Timestamp System

Brooks Harris Version 1 2025-02-02

Table of Contents

Notation
"YMDhms" is shorthand for year-month-day hour:minute:second representation.
ISO 8601 representation is supplemented with suffixes (UTC) and (TAI), for example
1970-01-01 00:00:10 (TAI) = 1970-01-01T00:00:00 (UTC).
"UTC1970" is shorthand for 1970-01-01 00:00:10 (TAI) = 1970-01-01T00:00:00 (UTC).

1 Introduction

The Common Calendar Enhanced timestamp includes extended metadata to represent all aspects of TzDb data which are typically not included in timestamps used in common practice. This includes UTC-offset transitions (STDOFF), DST transitions (dstoff), POSIX abbreviated name transitions and leap-second transitions.

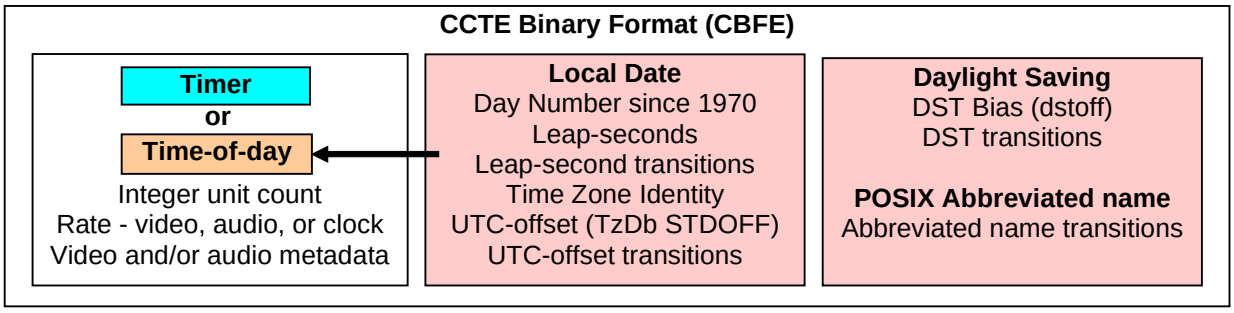
CCTE is designed to facilitate calculation of time marks within the current day without need for access to external metadata for systems that may require inexpensive timestamp receivers. It is also helpful in development because it reflects all the metadata thus revealing any inaccuracies in algorithms and other formats. It may form the basis of a "forensic tracing" application, useful for "legal time" traceability. Like CCTM it can support video and audio media labeling.

Common Calendar Enhanced Binary Format (CBFE) provides a compact binary data format to facilitate fast transfer, efficient interchange, and economical storage. The CBFE design is intended for binary systems, protocols, and languages, such as embedded systems, time dissemination, and c/c++ , while offering compatible expression on other platforms and formats such as Java and XML.

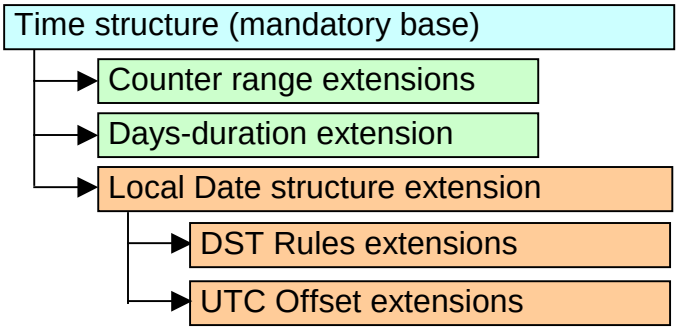
CBFE acts in concert with Common Calendar Enhanced Character Format (CCFE) to provide comprehensive description of local date and time with symmetrical conversion between the binary and character based YMDhms representations.

Common Calendar Enhanced Binary Format (CBFE)

The CBFE binary timestamp is a variable length compound counter with associated metadata. It is made up of the mandatory Time structure base structure that provides the anchor for any configuration of CBFE, and optional extensions, including the important Local Date structure.



The variable size design provides a single syntax to construct binary timestamps to accommodate use as a simple 24 hour timer (using only the mandatory Time structure), or a full local date and time-of-day timestamp (with the Local Date extension). This flexibility minimizes the total size of the timestamp for each purpose.



The mandatory Time structure contains a 35-bit unsigned counter and sign bit (signed 36-bit). The range may be increased to 48, 64, or 80 bits using the counter range extensions. It can represent resolutions from seconds to picoseconds depending on the rate indicator. Used by itself it is a simple 24-hour timer.

The Local Date structure extension is added to provide full local date and time-of-day representation. Its calendar date uses the TAI-UTC API date data structure which includes the day-number-since-UTC1970 and TAI-UTC (leap-seconds). Its time zone metadata uses elements derived from the Common Calendar Time Zone API.

Additional leap-second metadata provides sufficient information for interpretation by a simple receiving application that has no leap-second or local time information available, and facilitates conversion to the CCFE YMDhms representation.

When Local Date is used the values of the base Time structure are interpreted as time-of-day. Together, the compound counter, made up of the time-of-day from Time, and the day-number and leap-seconds from Local Date, represent the local date and time as an absolute seconds-since-UTC1970 value. For example, at a seconds resolution (rate), the formula is:

```
seconds-since-UTC1970=(DayNumber*86400)+(leap-seconds)+(seconds-since-
midnight)
```

The design, with its compound data elements, lends advantage to reading and interpreting a Common Calendar timestamp. The emitting application has the responsibility to accurately populate the timestamp parameters while it has access to leap-second and time zone information. Important aspects of the emitter's processing are transferred to the timestamp data and metadata and this simplifies the receiver's role. The receiver is relieved from performing some computationally intensive processing and there is no need for access to leap-second and local time information.

Common Calendar specifies a policy of recording the local date and time as known to the emitting system in all cases. This brings clarity and specificity to the meaning of the data and consistency and simplicity to generating and interpreting Common Calendar timestamp values.

An outline of CBF features:

- **Time structure** - mandatory fixed size 64-bit structure containing:

- clock rate enumeration (supports seconds, milliseconds, nanoseconds, etc)
- 36-bit counter extensible to 48, 64, and 80 bits
- flag to indicate the presence of Days-duration extension
- flag to indicate the presence of Local Date structure extension

- **Local Date structure** – optional fixed size 15 bytes, 120-bit, local date structure containing:
  - The CCT origin is UTC1970 (1970-01-01 00:00:10 (TAI) = 1970-01-01T00:00:00 (UTC))
  - CCT always records local time as known to the emitting application.
  - Ordinal day counter with leap-seconds for approximately -22967 to 22967 year range
  - UTC-Offset of local time
  - IANA Time Zone Database time zone
  - DST bias, change day, change time-of-day
  - UTC-Offset shifts and shift time-of-day
  - Leap-second Time-of-day Count Mode (TOD_LEAPSECOND) defines the linkage between the Time and Local Date:
    - TOD_LEAPSECOND_UTC_UTC - Leap-seconds are introduced simultaneous with UTC on local timescales labeled as xx:59:60 as per UTC specification
    - TOD_LEAPSECOND_UTC_NTP - Leap-seconds are introduced simultaneous with UTC on local timescales labeled as xx:59:59 (“freeze”) as per NTP specification
    - TOD_LEAPSECOND_UTC_POSIX - Leap-seconds are introduced simultaneous with UTC on local timescales labeled as xx:00:00 ("roll over and reset") as per POSIX specification
    - TOD_LEAPSECOND_MIDNIGHT - Leap-seconds are introduced immediately before the following date on each Local Timescale labeled as 23:59:60 as per UTC specification. This is sometimes called “rolling leap-seconds”.
    - TOD_86400S_DAY_DATE – Date and time-of-day data are treated as 86400-second-days (Leap-seconds are unavailable or unknown).
    - TOD_LEAPSECOND_NONE - time not related to date

Common Calendar Enhanced Character Format (CCFE)

People generally expect calendar date and time-of-day to be shown in some form of YMDhms, not some form of machine efficient integer count. A seconds-since-UTC1970 value of 78796801s is unintelligible to a human, while a YMDhms encoding of this number as 1972-07-01 00:00:00 (UTC) imparts immediate familiar meaning.

Machine readable character-based interchange formats using a YMDhms form are common. They provide a machine-readable format that is also meaningful to humans. However none are capable of representing unambiguous UTC accurate local time because they lack sufficient metadata to fully describe the meaning of the YMDhms values.

Common Calendar Character Format (CCFE) specifies a character based machine-readable interchange format in YMDhms form to impart familiar meaning to human users with the necessary and sufficient metadata to fully describe local date and time for interoperable machine interchange. It also supports representation of time points and intervals unrelated to calendar date.

CCFE is designed to reflect equivalent data and metadata contained in Common Calendar Enhanced Binary Format (CBFE) to provide symmetrical and complete conversion between the two. They are designed as tightly coupled pair, and this specification references the CBF specification and the CCT reference implementations of CCFE and CBFE to make clear the connections between the CBFE data and metadata. The corresponding CCFE character representations, and the details of conversion algorithms required.

The CCFE format can be used independently of CBFE if sufficient information is available. It may be most convenient to implement CCFE in combination with CBFE.

CCTE Character Format (CCFE)
`D2024-03-10T03:00:00U-06Zamerica/denverAmdtV2024aL27S01t01a02cMuX`

The CCT reference implementation implements CCFE and CBFE in the CCTELib library. CBFE is implemented as class CCbfE and the CBFE data types are defined in CBFE.h header. CCFE is implemented as class CCcfE, and delimiter characters are explicitly defined in the CCFE.h header. See CCTELib/CCFE.h, CCcfE.h, and CCcfE.cpp. See CCTELib/CBFE.h, CCbfE.h, and CCctE.cpp.

Geostamp

The Common Calendar Timestamp (CCT) specification has been extended to include geographic coordinates to create a Geostamp. The Geostamp specification was developed in collaboration with Son Voba of Sync-n-Scale to support "tractability provenance". A Geostamp consists of geographic coordinates and a CCT timestamp.

For an overview of Common Calendar in general please see
Common Calendar Introduction and Scope

2 Scope

This interoperability standard specifies an extended metadata timestamp type with a variable length binary data format and a corresponding character format with mandatory and optional components and metadata to fully represent deterministic time points and time intervals including UTC accurate local date and time.

3 Normative References

Common Calendar Date and Time Terms and Definitions

Common Calendar TAI-UTC API

Common Calendar YMDhms API

Common Calendar Local Timescales

Common Calendar Time Zone API

ISO 8601 2004-12-01, Data elements and interchange formats — Information interchange — Representation of dates and times
http://www.iso.org/iso/iso8601

Common Calendar Geostamp

National Marine Electronics Association
NMEA 0183 Interface Standard
GGA Global Positioning System Fix Data. Time,
Position and fix related data for a GPS receiver $GPGGA
https://www.nmea.org/nmea-0183.html

4 Common Calendar Enhanced Binary Format (CBFE)

Common Calendar Enhanced Binary Format (CBFE) specifies a compact variable length binary data structure containing time, date (if applicable), and metadata. It includes sufficient information to construct a corresponding Common Calendar Enhanced Character Format (CCFE).

A CBFE and its corresponding CCFE format can have one of six meanings:

- time point with UTC accurate local date and time-of-day
- time point with UTC accurate local date with time portion having no relation to the date
- time point less than 24 hours with no relation to date (< 86400 seconds)
- time point 24 hours or greater with no relation to date (>= 86400 seconds)
- time interval less than 24 hours (< 86400 seconds)
- time interval 24 hours or greater (>= 86400 seconds)

In the case of representation of a time point of UTC accurate local date and time-of-day a CBFE and its corresponding CCFE shall contain the local date, time-of-day, and local metadata as known to the emitting system when generated in accordance with the rules and guidelines set out in Common Calendar Local Timescales.
A CBFE shall not represent an incomplete, ambiguous, or non-deterministic time-point or interval. Any incomplete data or metadata for the intended purpose shall be regarded as an error and the CBFE as a

whole shall be regarded as malformed. Applications should take appropriate action to protect the system and users from incomplete or erroneous data.

4.1 CBFE Components

A CBFE is constructed from the mandatory Time data structure CBFETime_st with optional extensions. The components are assembled to support the required functionality. Each component is described in detail with sections below. The order of construction of optional components is shown in sections CBFE Construction and Assembly.

4.1.1 Time

The CBFETime_st structure is mandatory and provides the fixed-size (64-bit) anchor of the variable length format. It contains a 35-bit unsigned counter and a sign bit (a 36-bit signed value) together with rate enumeration, counter size indicator, and extension flags.

The range of the CBFETime_st 35-bit counter may be extended to 48, 64, or 80 bits by addition of the optional CBFECounterHigh16_st and/or CBFECounterHigh32_st extensions to accommodate 24 hours at rates higher than milliseconds.

The presence of one or both of the range extensions is indicated by CBFETime_st::m_eCounterSize holding a CBFECounterSize_et enumeration. See CBFECounterSize_et. The extensions are treated as the high-words and the CBFETime_st::m_ulCounterLow32 counter as the low-word.

CCT supports several rates, or resolutions. These are indicated by enumerations. See CBFERate_et. Each requires a counter of sufficient size to accommodate 24 hours (86400 seconds) plus one leap-second when applicable (86401 seconds).

```
typedef struct CBFETime_st           // 8 bytes, 64-bit
{
unsigned char m_eRateEnumeration:8; // Enumerated clock rates
                                    // See CRateTable.h CBFRate_et
unsigned char m_bLocalDateExt:1;    // CBFLocalDate_st extension
                                    // is present
                                    // See CBFLocalDate_st
unsigned char m_b24HourPeriodExt:1; // CBF24HourPeriod_st extension
                                    // is present
unsigned char m_eCounterSize:2;     // Enumerated 35/48/64-bit size
                                    // See CBFCounterSize_et
unsigned char m_bCounterSign:1;     // 1 = counter value is negative
unsigned char m_iCounterHigh3:3;   // MSBs of 35-bit counter.
                                   // Unused when  (set to zero) when
                                   // 16-bit or 32-bit counter high extensions
                                   // are active for 48-bit or 64-bit counters
unsigned long m_ulCounterLow32;    // 32 bit unsigned counter
                                   // treated as low word if counter
                                   // size extension present
unsigned char m_bIsInterval:1;     // CBF is an interval
unsigned char m_bSpeedExt:1;       // Speed extension is present
unsigned char m_eCountMode:4;      // Enumerated clock or media Count Mode
                                   // See COUNTMODE_et
unsigned char m_eBaseCountModulus:2; // Enumerated Base Count
                                     // Modulus (ST12 heritage)
                                     // See ST12COUNT_et
unsigned char m_eVideoRateOfAudio:6; // Specifies video rate
                                     // associated with audio
                                     // Also used to specify TCF
                                     // hh:mm:ss:ff/aaaa
                                     // audio encoding
                                     // If m_eRateEnumeration ==
                                     // any AUDIO rate (enum
                                     // 84-128), one of the
                                     // standard video rates
                                     // See CRateTable.h CBFRate_et
```

```
unsigned char m_Reserved:2;
} CBFETime_st;
```

4.1.1.1 Rate enumeration

CBFE supports several rates, or resolutions. These are indicated by enumerations. Each requires a counter of sufficient size to accommodate 25 hours (86400 seconds) plus one leap-second when applicable (86401 seconds). CCT “standard rates” are enumerated in CCTRateLib, CRateTable.h. See Annex A - CCT Standard Rates

4.1.1.2 Counter Size enumeration

The range of the CBFETime_st 36-bit counter may be extended to 48, 64, or 80 bits by addition of the optional CBFECounterHigh16_st and/or CBFECounterHigh32_st extensions to accommodate 24 hours at rates higher than milliseconds.

The presence of one or both of the range extensions is indicated by CBFETime_st::m_eCounterSize holding a CBFECounterSize_et enumeration. See CBFECounterSize_et. The extensions are treated as the high-words and the CBFETime_st 32-bit counter as the low-word.

```
typedef enum CBFECounterSize_et     // m_eCounterSize:2
{
COUNTERSIZE_32 = 0, // 32-bit counter contained in base object
COUNTERSIZE_48,     // 48-bit counter
                    // 16-bit counter extension present,
                    // CBFETime_st::m_lCounterLow32,
COUNTERSIZE_64,     // 64-bit counter
                    // 32-bit counter extension present,
                    // TBOCounter32_st::m_ulCounterHigh32 and
                    // CBFETime_st::m_lCounterLow32,
COUNTERSIZE_80      // 80-bit counter
                    // 16-bit counter extension present,
                    // 32-bit counter extension present,
                    // TBOCounter16_st::m_unCounterHigh16 and
                    // TBOCounter32_st::m_ulCounterHigh32 and
                    // CBFETime_st::m_lCounterLow32,
} CBFECounterSize_et;
```

The counter size must accommodate 24 hours (86400 seconds) plus one leap-second if applicable (86401 seconds) of the rate, or resolution. The required sizes have been calculated, for examples, as:

Microseconds requires 37 bits -
86401 * 1000000 = 86401000000, 2^37 = 137438953472 MAX, unused 51037953472

Nanoseconds requires 47 bits -
86401 * 1000000000 = 86401000000000, 2^47 = 140737488355328 MAX, unused 54336488355328

Picoseconds requires 57-bits -
86401 * 1000000000000 = 86401000000000000, 2^57 = 144115188075856000 MAX, unused 57714188075855900

The rates supported, and the required counter sizes for each are:

```
enum       rate             resolution        counter size
CBFERate_et                                   CBFECounterSize_et
CLOCK_0    1/1 second                          COUNTERSIZE_32
CLOCK_1    1/10 tenths                         COUNTERSIZE_32
CLOCK_2    1/100 hundredths                    COUNTERSIZE_32
CLOCK_3    1/1000 Millisecond                  COUNTERSIZE_32
CLOCK_6    1/1000000 Microsecond               COUNTERSIZE_48
CLOCK_9    1/1000000000 Nanosecond             COUNTERSIZE_48
CLOCK_12   1/1000000000000 Picosecond          COUNTERSIZE_64
CLOCK_15   1/1000000000000000 Femtosecond      COUNTERSIZE_80
```

```
CLOCK_18   1/1000000000000000000 Attosecond   COUNTERSIZE_80
```

(current code implementation supports up to picoseconds due to platform 64-bit limit)
(Zeptosecond would require 87 bits and Yoctosecond 97 bits, not currently supported)

If rate is <= CLOCK_3 (milliseconds) the CBFETime_st::m_ulCounterLow32 forms the entire counter and no counter size extension is required..

If rate is CLOCK_6 (microseconds) or CLOCK_9 (nanoseconds) CBFECounter16_st shall be added and CBFECounter16_st::m_CounterHigh16 is treated as high word and CBFETime_st::m_ulCounterLow32 as the low word of the compound 48-bit counter.

If rate is CLOCK_12 (picoseconds) CBFECounter16_st and CBFECounter32_st shall be added. The CBFECounter32_st::m_CounterHigh32 is treated as high word, CBFECounter16_st::m_CounterHigh16 as a "mid word", and CBFETime_st::m_ulCounterLow32 as the low word of the compound 80-bit counter.

See CCbf::SetTimeCounter(..) for bit-shift examples from each CBFE counter size to uint64_t (unsigned 64-bit integer type).

See CCbf::GetTimeSeconds(..) for bit-shift examples to set CBFE compound counter values from seconds and fractions of seconds.

4.1.2 Counter Size Extensions

The CBFETime_st::m_ulCounterLow32 counter can be extended to 48-bits by adding CBFECounter16_st::m_CounterHigh16, or 64-bits by adding CBFECounter32_st::m_CounterHigh32, or 80-bits by adding both. The presence of these extensions are indicated by CBFETime_st::m_eCounterSize enumeration. See CBFECounterSize_et.

If rate is <= CLOCK_3 (milliseconds) the CBFETime_st::m_ulCounterLow32 forms the entire counter.

If rate is CLOCK_6 (microseconds) or CLOCK_9 (nanoseconds) CBFECounter16_st is added and CBFECounter16_st::m_CounterHigh16 is treated as high word and CBFETime_st::m_ulCounterLow32 as the low word of the compound 48-bit counter.

If rate is CLOCK_12 (picoseconds) CBFECounter16_st and CBFECounter32_st are added and CBFECounter32_st::m_CounterHigh32 is treated as high word, CBFECounter16_st::m_CounterHigh16 as a "mid word", and CBFETime_st::m_ulCounterLow32 as the low word of the compound 80-bit counter.

See CCbf::SetTimeCounter(..) for bit-shift examples from each CBFE counter size to uint64_t (unsigned 64-bit integer type).

See CCbf::GetTimeSeconds(..) for bit-shift examples to set CBFE compound counter values from seconds and fractions of seconds.

The corresponding CCF encoding of rate is contained in the CCF Time Element. See CCF.h, Time Element.

```
typedef struct CBFECounterHigh16_st // 16 bit counter extension
{
  unsigned short m_unCounterHigh16;
} CBFECounterHigh16_st;

typedef struct CBFECounterHigh32_st  // 32 bit counter extension
{
  unsigned long m_ulCounterHigh32;
} CBFECounterHigh32_st;
```

4.1.3 24 Hour Period

CBFE24HourPeriod_st extension is used to increase the range of the CBFETime_st counter equal to or greater than 24 hours (>= 86400 seconds).

The name "24 hour period" is used to avoid the word "day" in this context because only UTC, with its occasional 86401 second leap-second days, represents accurate calendar dates. The 24-hour Period Extension indicates a count of fixed-length 86400 second periods, not UTC days.

If the CBFE24HourPeriod_st extension is present the CBFETime_st counter value shall be interpreted as a zero-base 86400 second period within the CBFE24HourPeriod_st count.

The CBFE24HourPeriod_st::m_ul24HourPeriods member is 21 bits thus matching the range of the 21-bit DateTaiUtc_st::m_ui1970DayNumber member of DateTaiUtc_st used by TAI-UTC API. See TaiUtcApi.h - DateTaiUtc_st.

The corresponding CCF element is 24-hour Period Element. See CCF.h, 24-hour Period Element.

CBFE24HourPeriod_st can be used to extend the range of a time-point or interval.

If CBFETime_st::m_bIsInterval == false the CBFE represents a time-point and the CBFE24HourPeriod_st::m_ul24HourPeriods value holds a zero-based count of 24 hour periods.

The corresponding CCF 24-hour Period Element encoding used as a time-point is E delimiter - (E)vent time-point >= 24 hours

If CBFETime_st::m_bIsInterval == true the CBFE represents an interval and the CBFE24HourPeriod_st::m_ul24HourPeriods value holds a zero-based count of 24 hour periods.

The corresponding CCF 24-hour Period Element encoding used as an interval is P delimiter - (P)eriod interval >= 24 hours

Use of the CBFELocalDate_st is prohibited if CBFETime_st::m_bIsInterval == true because it is impossible to represent an accurate duration from a specific date and time without the full values and local time parameters to represent both the initial and final time points of that interval. For that case consider using two full CBFE date-time local representations and calculate the duration between those two time points. A CBFE can be used to represent that difference.

```
                                    // 32 bit 24 hour (86400
typedef struct CBFE24HourPeriod_st   // second) period extension
{
unsigned long m_ul24HourPeriods:21; // count of 86400-second-
                                    // periods from arbitrary
                                    // zero origin
                                    // 2^21 = 2097152 MAX
unsigned long m_ulReserved:11;
} CBFE24HourPeriod_st;
```

4.1.4 Local Date

The optional CBFELocalDate_st structure represents the local calendar date together with sufficient metadata to fully describe local date and time-of-day in accordance with the rules and guidelines set out in Common Calendar Local Timescales. The CBFEDstBias_st extension adds Daylight Saving bias if applicable. The CBFEDstTransDay_st signals a DST change during the day if applicable. The CBFEUtcShift_st extension adds support of UTC-offset shifts if applicable. The presence of CBFELocalDate_st is indicated by CBFETime_st::m_bLocalDateExt.

If the CBFELocalDate_st extension is present the CBFETime_st counter value represents time-of-day (24 hours plus one leap-second if applicable) on the calendar date indicated by the values of CBFELocalDate_st. DST information shall be provided by the CBFEDstBias_st and CBFEDstTransDay_st extensions if applicable.

The values and metadata of CBFETime_st, CBFELocalDate_st and CBFEDstBias_st (if present) shall *always* represent the emitting system's local time. There is no need for a choice between "local time" and "GMT time", because all CBFE and CCF timestamps represent local time. One of those local timescales is "Etc/UTC" which can be used if needed.

The local time information is obtained from the IANI Time Zone Database (Tz Database, TzDb) through the Time Zone API, including the time zone name and UTC offset, and DST rules if applicable. The version of the Tz Database source files is recorded to make accurate forensic analysis possible in cases where the Tz Database data may have changed since a timestamp was written. See TzDatabaseApi.h

The calendar date value itself, CBFELocalDate_st::m_DateTaiUtc_st, includes a zero-based count of days-since-UTC1970 together with the positive and negative leap-second values. This is the same DateTaiUtc_st data type used by TaiUtcApi, see TaiUtcApi.h.

Excerpt from TaiUtcApi.h

```
typedef struct DateTaiUtc_st // 7 bytes, 52-bit
```

```
{
signed long m_l1970DayNumber:24;  // signed zero-based 86400-second days
                                  // since 1970-01-01T00:00:00 (UTC)
                                  // ((2^24)/2)-1 = 8388607 MAX
                                  // 8388607 / 365.24 = ~22967.38 years
                                  // (2^24)/2)* -1 = -8388608 MIN
                                  // -8388608 / 365.24 = ~-22967.38 years
signed long m_lLeapsecsNegLow:8;  // Negative Leap-seconds low word
signed short m_nLeapsecsNegHigh:7; // Negative leap-seconds high word
                                   // Negative TAI-UTC minus initial 10s
                                   // ((2^15)/2)* -1 = -16384 MIN
signed short m_nLeapsecsHigh:9;   // Positive leap-seconds high word
signed char m_nLeapsecsLow:6;     // Positive leap-seconds low word
                                  // TAI-UTC minus initial 10s
                                  // ((2^15)/2)-1 = 16383 MAX
signed char m_nReserved:2;
} DateTaiUtc_st;
```

The origin (epoch) of the date and TAI-UTC values shall be:

441762480s TAI = 1970-01-01 00:00:10(TAI) = 1970-01-01T00:00:00(UTC) = MJD 40587. (called UTC1970). Thus seconds-since-UTC1970 = (1970DayNumber * 86400s) + positive leap-seconds + negative leap-seconds.

Time zone and UTC offset metadata are included according to the Time Zone API. See TzDatabaseApi.h

The optional CBFEDstBias_st structure shall be appended to provide required DST metadata if applicable. See CBFELocalDate_st::m_bDSTExt and CBFEDstBias_st.

The optional CBFEDstTransDay_st structure shall be appended to provide required DST changes if applicable. See CBFELocalDate_st:: m_bDstTransDayExt and CBFEDstTransDay.

The optional CBFEUtcShift_st structure shall be appended to provide required UTC-Offset metadata if applicable. See CBFELocalDate_st::m_bUtcShiftExt and CBFEUtcShift.

The corresponding CCF element is Date Element. See CCF.h, Date Element.

The presence of a CBFELocalDate_st is indicated by
`CBFETime_st:: m_bLocalDateExt:1`

```
typedef struct CBFELocalDate_st  // 16 bytes, 128-bit
{
DateTaiUtc_st m_DateTaiUtc_st;   // UTC1970 Day number
                                 // and Leapsecs values
                                 // See TaiUtcApi.h - 7 bytes, 52-bit
TZDTimeZoneID_st m_TZDTimeZoneID_st; // tz datebase zone id
                                     // see TzDatabaseApi.h - 4 bytes, 32-bit
signed long m_lUTCOffset:17;    // UTC Offset in seconds (TzDb STDOFF)
                                // 17 bits signed = min -65536 / 3600 =
                                // -18.20444444 hrs
                                // 17 bits signed = max  65535 / 3600 =
                                // 18.20416667 hrs
unsigned long m_eLsTodCntMode:3; // Enumerated Leap-second Count Mode
                                 // See CBFLsTodCntMode_et
unsigned long m_eDstMode:2;      // DST count mode enum
                                 // See CBFDstCountMode_et
unsigned long m_bDstBiasExt:1;   // CBFDstBias_st present
                                 // following this
                                 // TBOLocalTime_st struct
unsigned long m_bDstTransDayExt:1; // Dst Transition Day_st extension
                                   // see CBFDstTransDay_st
unsigned long m_bUtcShiftExt:1;  // CBFUtcShift_st present
                                 // see CBFUtcShift_st
```

```
unsigned char m_bIsLeapSecond:1;         // this local second is a leap-second
unsigned char m_bIsLeapSecondDay:1;      // today local is a leap-second day
unsigned char m_bIsLeapSecondNegative:1; // leap-second is negative
unsigned char m_lReserved:4;
} CBFELocalDate_st;
```

4.1.4.1 Time Zone Identity

TzDb encodes the time zone identity as strings, such as “America/New_York”, “Europe/Moscow”.
CCT encodes these as index numbers, derived from The TzDb source files.

```
typedef struct TZDTimeZoneID_st  // 4 bytes, 32 bits
{
unsigned short m_unZoneIdx:10;   // tz datebase zone index
                                 // 2^10 = 1024 MAX
// same variables as TZDDataRelease_st but local here for byte alignment
unsigned short m_unTzDataReleaseLetter:5; // 26 release letters a-z
                                          // 2^5 = 32 MAX
unsigned short m_bCBFLocationExt:1;       // CBFLocation_st present
unsigned short m_unTzDataReleaseYear:12;  // UTC1970 zero based year number
                                          // 1970 + 3465 = year 5435
                                          // 2^12 = 4096 MAX
unsigned short m_bCBFAbbrExt:1;         // CCbf::m_aCBFChar_st16PosixAbbr[16];
unsigned short m_bCBFAbbrChangeExt:1;   // CCbf::m_aCBFChar_st16PosixAbbr[16];
unsigned short m_unReserved:2;
} TZDTimeZoneID_st;
```

See Common Calendar Time Zone API

4.1.4.2 Posix Abbreviated Name

Tz Database assigns "time zone abbreviations" to the Posix TZ environment variable.

For example, in time zone "America/New_York", when DST is not in effect,
the TZ environment variable provided by Tz Database is "EST", and when DST is in effect, “EDT”.

Charaters are encoded in a variable length character string, CBFEChar_st

```
typedef struct CBFEChar_st
{
unsigned char m_Char:7;
unsigned char m_Next:1;
} CBFEChar_st;
```

Each character is limited to ASCII values 0 - 126.
CBFEChar_st.m_Char:7 holds the character.
CBFEChar_st.m_Next:1 flags if another character follows.
  if CBFEChar_st.m_Next == 1, another character follows
  if CBFEChar_st.m_Next == 0, no more characters

CCT carries these encodings in class CCbf as lower-case characters in a 16 byte CBFEChar_st array;
see CCbf::m_aCBFEChar_st16Abbr;

Only the populated CBFEChar_st bytes of this array are written to the CBFE binary format.

For example, for the TZ environment variable "EST"
three CBFEChar_st bytes are written:
CBFEChar_st.m_Char = 'e';
CBFEChar_st.m_Next = 1; // have another char

CBFEChar_st.m_Char = 's';
CBFEChar_st.m_Next = 1; // have another char
CBFEChar_st.m_Char = 't';
CBFEChar_st.m_Next = 0; // last char

See CBFE.h CBFEChar_s
See CCbf::SetCBFEChar_stStringLowerCase()
See CCbf::GetCBFEChar_stString()

4.1.4.3 Posix Abbreviation Name Change

Some Tz Database transitions occur only because the Posix TZ environment variable changes. This is a rare situation, but supported by the presence CBFEAbbrChange_st as flagged by CBFELocalDate_st::TZDTimeZoneID_st:m_bCBFEAbbrChangeExt.

Only the populated CBFEChar_st bytes of these arrays are written to the CBFE binary format.

```
typedef struct CBFEAbbrChange_st   // 37 bytes, 296-bit
{
unsigned long m_ulAbbrChangeTime:17; // Abbr change time-of-day
                      // 86400 = 24 hours
                      // 17 bits unsigned max 131071 / 3600 = 36.40861111 hrs
CBFEChar_st m_aCBFEChar_st16Abbr_Before[16];
CBFEChar_st m_aCBFEChar_st16Abbr_After[16];

unsigned char m_lReserved:6;
} CBFEAbbrChange_st;
```

The CBFE member is `CBFEAbbrChange_st;`
The presence of the Posix Abbreviation Name Change is indicated by `CBFELocalDate_st::TZDTimeZoneID_st:m_bCBFEAbbrChangeExt`

4.1.4.4 Time-of-Day Count Mode

CCT provides means to construct the YMDhms counting sequence through Common Calendar Character Format (CCF) to support either: A) the widely used common practice of introducing leap-seconds simultaneous with UTC or B) an alternate scheme introducing the leap-second at the end of the local day (rolling leap-second).

CBFETodCountMode_et instructs how the CBFE binary representation is to be converted to the CCF YMDhms counting sequence representation. Options include support for UTC, POSIX, or NTP YMDhms sequences.

See Common Calendar Local Timescales.

```
typedef enum CBFLsTodCntMode_et
{
TOD_LEAPSECOND_NONE = 0,     // Not Time-of-day
                             // Time has no relation to Date,
                             // Time has zero or "arbitrary" epoch
                             // CCF char indicator "a" (arbitrary)
TOD_LEAPSECOND_MIDNIGHT,     // Leap-seconds introduced at
                             // midnight on local timescales
                             // (Rolling leap-second)
                             // CCF char indicator "m" (midnight)
TOD_LEAPSECOND_UTC_UTC,      // Leap-seconds introduced
                             // simultaneous with UTC on
                             // local timescales
                             // leap-second label
                             // 23:59:60
                             // CCF char indicator "u" (utc)
TOD_LEAPSECOND_UTC_NTP,      // Leap-seconds introduced
                             // simultaneous with UTC on
                             // local timescales
```

```
                              // leap-second label
                              // 23:59:59 ("freeze")
                              // CCF char indicator "n" (ntp)
TOD_LEAPSECOND_UTC_POSIX,     // Leap-seconds introduced
                              // simultaneous with UTC on
                              // local timescales
                              // leap-second label
                              // 00:00:00 ("roll over and reset")
                              // CCF char indicator "p" (posix)
TOD_LEAPSECOND_24HOUR_DAY,    // 86400-second-days of calendar
                              // (leap-seconds unknown or unavailable)
                              // CCF char indicator "g" (gregorian)
TOD_LEAPSECOND_NA             // not set or logic error (default)
} CBFLsTodCntMode_et;
```

4.1.4.5 UTC Offset Shift

Many time zones have shifted their UTC-offset independent of DST shifts. Tz Database calls this "STDOFF" for "standard time offset". When this occurs CBFEUtcShift_st provides metadata to represent these UTC-offset shifts.

The presence of a CBFEUtcShift_st is indicated by `CBFELocalDate_st::m_bUtcShiftExt:1`

```
typedef struct CBFEUtcShift_st     // 5 bytes, 40-bit
{
unsigned short m_unUtcShiftTimeLow:16; // Utc offset shift time-of-day in
seconds
signed short m_nUtcShiftLow:16;     // Utc offset shift in seconds

unsigned char m_eUtcShiftDay:2;     // Utc offset shift day enum
                                    // See TzDatabaseApi.h TZDUtcShiftDay_et
signed char m_nUtcShiftHigh:2;      // Utc offset shift in seconds
                                    // 18 bit min -131072 / 3600 = -
36.40888889 hr
                                    // 18 bit max  131071 / 3600 =
36.40861111 hr
unsigned char m_unUtcShiftTimeHigh:1;  // Utc offset shift time-of-day in
                                       // seconds
                                       // 86400 = 24 hours
                                       // 17 bits unsigned max 131071 / 3600 =
                                       // 36.40861111 hrs
unsigned char m_unReserved3:3;
} CBFEUtcShift_st;
```

4.1.5 Daylight Saving Time

Where Daylight Saving is observed the optional CBFEDstBias_st struct extension shall be added. The presence of CBFEDstBias_st is indicated by CBFELocalDate_st:: m_bDstBiasExt:1.

4.1.5.1 Daylight Saving Count Mode

Daylight Saving offsets may be applied in two modes:

“Conventional” where the DST shift occurs at some mid time-of-day

“Uninterrupted” where the DST shift at midnight

```
typedef enum CBFEDstCountMode_et
{
DSTCOUNTMODE_NOTAPPLICABLE = 0,
DSTCOUNTMODE_CONVENTIONAL, // conventional count mode
DSTCOUNTMODE_UNINTERRUPTED // uninterrupted count mode
```

```
} CBFEDstCountMode_et;
```

4.1.5.2 Daylight Saving Bias

If a DST shift is in effect its value, called “DstBias”, is held by CBFEDstBias_st:: m_nDstBias

Presence is signaled by CBFELocalDate_st:: m_bDstBiasExt:1

```
typedef struct CBFEDstBias_st  // 2 bytes, 16-bit
{
signed short m_nDstBias;     // DST bias in effect
                             // min -32768 / 3600 = -9.102222222 hr
                             // max  32767 / 3600 = 9.101944444 hr
} CBFEDstBias_st;
```

4.1.5.3 Daylight Saving Transition Day

If a DST shift is to occur during this day the CBFEDstTransDay_st shall appear.

Presence is signaled by CBFELocalDate_st:: m_bDstTransDayExt:1

```
typedef struct CBFEDstTransDay_st  // 5 bytes, 40-bit
{
unsigned short m_unDstTransTimeLow:16;// DST transition time of day
                       // 17 bit unsigned max 131071 = 36.40861111 hr
signed short m_nDstBiasChangeLow:16;  // DST bias transition value
                       // 18 bit min -131072 / 3600 = -36.40888889 hr
                       // 18 bit max  131071 / 3600 =  36.40861111 hr
signed char m_nDstBiasChangeHigh:2;
unsigned char m_unDstTransTimeHigh:1;
unsigned char m_lReserved5:5;
} CBFEDstTransDay_st;
```

Enumerated values used by CBFEDst_st variables are defined by Tz Database API. See TzDatabaseApi.h

```
typedef enum TZDDstChangeDay_et
typedef enum TZDDstBias_et
```

4.2 Construction of CBFE

CBFETime_st structure is mandatory and the anchor of the binary image. Optional counter extensions may be used to increase its counter range for any configuration.

The state of the CBFETime_st::m_bIsInterval flag indicates if the CBFE represents a time-point (m_bIsInterval == false) or a time interval (m_bIsInterval == true).

4.2.1 As Time-point:

If m_bIsInterval == false a CBFETime_st represents a time-point.

Used without the optional CBFELocalDate_st extension, the values of a CBFETime_st represent a time point within a 24 hour period (86400 seconds) with no reference to any date or other timescale. It represents a zero-based count of time units from an origin marked as "zero". It is a timestamp of a simple timer, like a "game clock" or "stop-watch".

Combined with the optional CBFELocalDate_st extension the CBFE represents a leap-second accurate local date and time. The CBFETime_st counter value represents time-of-day (24 hours plus one leap-second if applicable) on the calendar date indicated by the CBFELocalDate_st member values. The presence of CBFELocalDate_st is indicated by the CBFETime_st::m_bLocalDateExt flag.

If DST is observed in the time zone, the CBFEDst_st extension shall be present, providing the necessary DST information. The presence of CBFEDst_st is indicated by the CBFELocalDate_st::m_bDSTExt flag.

4.2.1.1 As UTC accurate local time-point timestamp :

CBFETime_st – mandatory
CBFETime_st::m_bIsInterval == false
CBFECounterHigh16_st and/or CBFECounterHigh32_st optional
CBFELocalDate_st – local date extension
CBFELocalDate_st::m_eTODMode == one of CBFETodCountMode_et:

- TOD_LEAPSECOND_UTC_UTC - Leap-seconds introduced simultaneous with UTC on local timescales, leap-second label 23:59:60, CCF TOD Count Mode char indicator "u" (utc)
- TOD_LEAPSECOND_UTC_NTP - Leap-seconds introduced simultaneous with UTC on local timescales, leap-second label 59:59:59 ("freeze"), CCF TOD Count Mode char indicator "n" (ntp)
- TOD_LEAPSECOND_UTC_POSIX - Leap-seconds introduced simultaneous with UTC on local timescales, leap-second label 00:00:00 ("roll over and reset"), CCF TOD Count Mode char indicator "p" (posix)
- TOD_LEAPSECOND_MIDNIGHT - Leap-seconds introduced at midnight on local timescales, Leap-second label 23:59:60, CCF TOD Count Mode char indicator "m" (midnight)

CBFEDst_st – Daylight Saving Time metadata extension (if applicable)

4.2.1.2 As UTC accurate local date with time portion having no relation to the date:

CBFETime_st – mandatory
CBFETime_st::m_bIsInterval == false
CBFECounterHigh16_st and/or CBFECounterHigh32_st optional
CBFELocalDate_st – local date extension
CBFEDstBia_st – Daylight Saving Time metadata extension (if applicable)
CBFEDstTransDay_st - Daylight Saving Change Day
CBFELocalDate_st::m_eTODMode == CBFETodCountMode_et ==
TOD_NONE - Not Time-of-day, Time has no relation to Date, Time has zero or "arbitrary" epoch
CCF TOD Count Mode char indicator "a"

4.2.1.3 As time-point timestamp < 86400 seconds:

CBFETime_st – mandatory
CBFETime_st::m_bIsInterval == false
CBFECounterHigh16_st and/or CBFECounterHigh32_st optional

4.2.1.4 As time-point timestamp >= 86400 seconds:

CBFETime_st – mandatory
CBFETime_st::m_bIsInterval == false
CBFECounterHigh16_st and/or CBFECounterHigh32_st optional
CBFE24HourPeriod_st – 24 hour period counter extension

4.2.2 As Time Interval

If CBFETime_st::m_bIsInterval == true the CBFE represents an interval, rather than a time-point.

Used without the optional CBFE24HourPeriod_st extension the CBFE represents an interval less than 24 hours (< 86400 seconds).

If optional CBFE24HourPeriod_st extension is present the CBFE represents an interval equal to or greater than 24 hours (>= 86400 seconds). The CBFE24HourPeriod_st::m_ul24HourPeriods value represents a count of 24 hour periods (1 == 86400 seconds) and the CBFETime_st counter value represents a count of 86400 seconds within that 24 hour period. The CBFE24HourPeriod_st shall be present if the state of CBFETime_st::m_bIsInterval == true and the total counter value exceeds 86400.

Note intervals >= 86400 seconds are not accurate UTC date and time because Leap-seconds are not accounted for; intervals are made up of 24 hour (86400 second) periods, not UTC accurate days.

4.2.2.1 As time interval < 86400 seconds:

CBFETime_st – mandatory
CBFETime_st::m_bIsInterval == true
CBFECounterHigh16_st and/or CBFECounterHigh32_st optional

4.2.2.2 As time interval >= 86400 seconds:

CBFETime_st – mandatory
CBFETime_st::m_bIsInterval == true

CBFECounterHigh16_st and/or CBFECounterHigh32_st optional
CBFE24HourPeriod_st – 24 hour period counter extension

4.2.3 Assembly Order

A CBFE shall be formed with the mandatory CBFETime_st structure with optional components concatenated in the following order:

- To form a time-point timestamp < 86400 seconds:

```
CBFETime_st - mandatory and CBFETime_st::m_bIsInterval == false
  CBFECounterHigh16_st and/or CBFECounterHigh32_st optional
  extensions to form 48, 64, or 80 bit counter
```

- To form a time-point timestamp >= 86400 seconds:

```
CBFETime_st - mandatory and CBFETime_st::m_bIsInterval == false
  CBFECounterHigh16_st and/or CBFECounterHigh32_st optional
  extensions to form 48, 64, or 80 bit counter
CBFE24HourPeriod_st - optional 24 hour period counter
```

- To form a UTC accurate local time-point timestamp:

```
CBFETime_st - mandatory and CBFETime_st::m_bIsInterval == false
  CBFECounterHigh16_st and/or CBFECounterHigh32_st optional
  extensions to form 48, 64, or 80 bit counter
CBFELocalDate_st - optional local date extension
CBFEDst_st - optional Daylight Savings Time metadata extension if applicable
CBFEUtcShift_st - optional UTC-offset shift metadata extension if applicable
Variable length Posix TZ environment string of CBFEChar_st
                in array CCbf::m_aCBFEChar_st16Abbr[]
```

Interval and LocalDate are exclusive

- To form a interval < 86400 seconds:

```
CBFETime_st - mandatory and CBFETime_st::m_bIsInterval == true
  CBFECounterHigh16_st and/or CBFECounterHigh32_st optional
  extensions to form 48, 64, or 80 bit counter
```

- To form an interval >= 86400 seconds:

```
CBFETime_st - mandatory and CBFETime_st::m_bIsInterval == true
  CBFECounterHigh16_st and/or CBFECounterHigh32_st optional
  extensions to form 48, 64, or 80 bit counter
CBFE24HourPeriod_st - optional day duration counter
```

CBFE assembly order in pseudocode -

```
CBFETime_st - mandatory primary component

if(m_CBFETime_st.m_eCounterSize == COUNTERSIZE_48)
  append CBFECounterHigh16_st
if(m_CBFETime_st.m_eCounterSize == COUNTERSIZE_64)
  append CBFECounterHigh32_st
if(m_CBFETime_st.m_eCounterSize == COUNTERSIZE_80)
  append CBFECounterHigh16_st
  append CBFECounterHigh32_st

// Interval and LocalDate are exclusive
if(m_CBFETime_st.m_bIsInterval)
 {
 if(m_CBFETime_st.m_b24HourPeriodExt)
  append CBFE24HourPeriod_st
 }
else
 {
 if(m_CBFETime_st.m_bLocalDateExt)
  append CBFELocalDate_st
 if(m_CBFELocalDate_st.m_bUtcShiftExt)
```

```
  append CBFEUtcShift_st
 if(m_CBFELocalDate_st.m_bDSTExt)
  append CBFEDst_st
 if(m_CBFELocalDate_st.m_TZDTimeZoneID_st.m_bCBFEAbbrExt)
  append number of CBFEChar_st in array m_aCBFEChar_st16Abbr[]
 if(m_CBFELocalDate_st.m_TZDTimeZoneID_st.m_bCBFELocationExt)
  append CBFELocation_st
 }
```

See `CCbf::AssembleCbf(char* pcaBinayCbf, int* piLen);`

Assembled CBFE size guide:

- Minimum size 8 bytes (mandatory CBFETime_st)
- Typical size with date and time may be the range of 30 - 34 bytes
- Maximum size with date, time and location could be 76 bytes

4.3 Geostamp

CCTC may include location, the geographical coordinates. This transforms a CCTC timestamp into a GeoStamp. The Geostamp specification was developed in collaboration with Son Voba of Sync-n-Scale to support "tractability provenance".

A Geostamp consists of geographic coordinates and a CCTC timestamp. Geostamps are technically accurate, making them suitable for general and legal purposes where time recording is used for tracking and auditing and a wide range of spatial-temporal geographic information systems (4D GIS) applications in machine learning, artificial intelligence, data analytics and blockchain distributed ledgers.

Like CCTC, Geostamps can be formed in either a binary or character format. The binary format supports efficient machine interoperability while the character format is human readable making their meaning accessible to those less familiar with the intricacies of timekeeping and geographic representations.

Coordinates are carried in the binary CBF CBFLocation_st structure and reflected in CCFC character format in the Location Element field.

If user has requested to include location in the CCTC timestamp.
m_CCTCParams_st.m_bUserIncludeLocation == true

The presence of location data is signaled by:
m_CCTCLocalDate_st.m_CCTCZoneID_st.m_bCBFLocationExt == true.

CCTC supports two forms of location:

- The time zone default location as provided by TzDb zone.tab. Latitude and longitude of the timezone's principal location in ISO 6709 sign-degrees-minutes-seconds format, Altitude is not given by TzDb zone.tab.
- The Latitude, Longitude and Altitude as supplied by National Marine Electronics Association (NMEA) NMEA 0183 GPGGA sentences. Latitude and longitude in degrees, minutes and decimal fractions of minutes. Altitude in meters and decimal fractions.

If m_CCTCParams_st.m_CBFLocation_st.m_bSourceIsExtern == false the data is in TzDb zone.tab ISO 6709 form.
See CTzDataParse::GeoOp_stToCBFLocation_stUTIL()

If the application has called CCctC::SetLocation() the data is in NMEA GPGGA form and m_CCTCParams_st.m_CBFLocation_st.m_bSourceIsExtern == true.
See CCctC::SetLocation()
See CTzDataParse::LatLgnAltToCBFLocation_stUTIL()

4.3.1 Geographic Coordinates

The CBFcLocation_st struct carries geographic coordinates.

```
typedef struct CBFLocation_st  // 14 bytes
{
signed long m_i21Lat_uMin:21;  // micro-minutes -100000 to 100000 range
```

```
                                // [((2^21)/2)-1  =  1048575 MAX]
signed long m_i9Lat_Deg:9;     // degrees -90 to 90 range, negative is South
                                // [((2^9) / 2) - 1 = 255 MAX]
signed long m_Pad1:2;
signed long m_i21Lng_uMin:21;  // micro-minutes -100000 to 100000 range
                                // [((2^21)/2)-1  =  1048575 MAX]
signed long m_i9Lng_Deg:9;     // degrees -180 to 180 range, negative is West
                                // [((2^9) / 2) - 1 = 255 MAX]
signed long m_Pad2:2;
signed long m_i25Alt_cm:25;    // signed centimeters range [((2^25)/2)*-1 MIN
                                // -16777216, ((2^25)/2)-1 MAX 16777215
signed long m_i7Lng_Min:7;     // minutes 0 to 60 range
                                // [((2^7) / 2) - 1 = 63 MAX]

unsigned char m_i7Lat_Min:7;   // minutes 0 to 60 range
                                // [((2^7) / 2) - 1 = 63 MAX]
unsigned char m_bSourceIsExtern:1; // flag is external location, otherwise is
                                    // TzDb zone.tab location
unsigned char m_bIsValidLat:1; // flag Latitude value valid
unsigned char m_bIsValidLng:1; // flag Longitude value valid
unsigned char m_bIsValidAlt:1; // flag Altitude value valid
unsigned char m_Pad3:5;

} CBFLocation_st;
```

See TzDatabaseApi.h, CBFCLocation_st
See CCctE.h, CCctE.cpp class CCctE
See CCbfE.h, CCbfE.cpp class CCbcE

4.4 CCTE RIFF Wrapper

One or more CCTEs may be contained in a CCTE RIFF Wrapper.

The four-cc of the CBFE RIFF shall be " CCTE"
The four-cc of each CBFE chunk shall be " ccte"

See CCTRiffCCct.h

```
#define FOURCC_CCTE "CCTE" // RIFF CCTE header
#define FOURCC_ccte "ccte" // RIFF CCTE chunk
```

See CCTRiffCCct.h and CCTRiffCCct.cpp

The CBFE RIFF Wrapper can be opened, populated by one or more assembled CCBs, and closed by

```
CCctE::WriteCRiff_Ccbf();
```

The CBFE RIFF Wrapper can be opened, populating one or more CCbf classes, and closed by

```
CCctE::ReadCRiff_CCct();
```

5 Common Calendar Enhanced Character Format (CCFE)

Common Calendar Character Format (CCF) specifies a character based machine-readable format of date and time in a YMDhms form to impart familiar meaning to human users. CCF reflects equivalent data of the CBFE binary format providing complete symmetrical conversion between the two.

CCF construction does not rely on external metadata, only the values contained in a CBFE. CCF contains sufficient information to populate a binary CBFE without reference to any external information.

A CCF, like its corresponding CBFE format, can have one of five meanings:

- time point with UTC accurate local date and time-of-day
- time point with UTC accurate local date with time portion having no relation to the date
- time point less than 24 hours with no relation to date
- time point 24 hours or greater with no relation to date
- time interval less than 24 hours (< 86400 seconds)

- time interval 24 hours or greater (>= 86400 seconds)

In the case of representation of a time-point of UTC accurate local date and time-of-day a CCF and its corresponding CBFE shall contain the local date, time-of-day, and local metadata as known to the emitting system when generated in accordance with the rules and guidelines set out in Common Calendar Local Timescales.

Examples of these configurations are shown in *Annex C - Example Listing from CCT Reference Implementation.* This listing is generated by the CCT reference implementations. See `CCTDemoConsole, CCTDemoConsole.cpp, CCTDemoTests.cpp, TestSelectedConfigurationsAndShowCbfValues()`.

5.1 ISO 8601 Variation

The CCF format is based on the guidelines set out in ISO 8601 with important variations. The 8601 scheme is augmented in several ways including:

- Formatting of hms shall use the full hh:mm:ss form
- If date is given formatting of date and time shall use the full YYYY-MM-DDThh:mm:ss form and shall include all required applicable metadata elements.
- Clock rate is added.
- Leap-second value is added.
- Time zone , DST and UTC-offset shift metadata is added
- The character "Z" is used to delimit the time zone identification data field, replacing the ISO 8601 use of “Z” for "Zulu"

CCFE does not support any form of partial representation of local date and time such as date only, "YYYY-DD-MM", or date and time only, "YYYY-MM-DDThh:mm:ss" because these are ambiguous without accompanying metadata. If date is given, CCFE and CBFE support only complete and deterministic representation of local date and time including the required local time metadata. Other representations are outside the scope of CBFE and CCFE.

5.2 Character Set

CCFE shall be encoded using the ASCII character set excluding control characters and white space as detailed in *Annex A - CCFE Character Set*.

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
!"#$%&'()*+,-./:;<=>?@[]^_`{|}~
```

5.3 Hard Terminator

The CCFE string shall be terminated with an upper case "X" in all cases.

5.4 Data Field Elements

A CCFE is a variable length string and its total length depends on the included optional elements and their contents. Some elements are fixed length, others variable length.

CCFE shall be encoded as several optional data field elements delimited by a single designated upper-case letter and terminated with "X" to facilitate parsing. The length of variable length elements is bounded by its delimiting character and the next delimiting character or terminator. The syntax is variable length with no white space.

CCFE delimiter characters and encoding characters are explicitly defined in CCTLib/CCFE.h. The required elements and order of assembly are described in section *4.5 Assembly and Order*. Each element is described in sections below.

Delimiting Character	Element	Example fragment(s)
T	Time: • singly - time point < 86400s • with Date and TOD_LEAPSECOND_MIDNIGHT or TOD_LEAPSECOND_UTC_UTC	`T23:59:59n999999999`

	or TOD_LEAPSECOND_UTC_NTP or TOD_LEAPSECOND_UTC_POSIX - time-of-day of UTC accurate local date • with Date and TOD_NONE - accurate local date and time having no relation to the date • with Event - time point >= 86400s	
E	Event with Time - time point >= 86400s	E123T23:59:59n999999999
I	Interval: • singly - time Interval < 86400s • with Period - time interval >= 86400s	I23:59:59n999999999
P	Period with Interval - time interval >= 86400s	P123I23:59:59n999999999
D	Date	D2016-02-22
U	UTC Offset	U-5, U+5, U+02:30:17, U+3w01+02
Z	(Z)one (time zone)	Zamerica/new_york
A	Posix zone abbreviation	Aest
Q	Posix zone abbreviation Change	Qewt_ept+19
V	Tz Database tzdata release version	V2021a
L	Leap-seconds	L27
S	Daylight Saving bias, state and mode	Ss+01c
M	Time-of-day (TOD) Count Mode:	Mm
X	Hard terminator	X

5.4.1 Time Element

The time element represents a time point or time interval in the familiar hours:minutes:seconds form, that is; 60 seconds = 1 minute, 60 minutes = 1 hour, and 24 hours = one 24 hour period (86400 seconds). In the case of representing UTC accurate data and time, a leap-second day includes an additional second (61 seconds, "23:59:60") and has a duration of 86401 seconds.

The corresponding CBFE binary data structure is CBFETime_st. The CBFE member variable: CBFETime_st::m_bIsInterval flags if the data represents a time point (m_bIsInterval == false) or interval (m_bIsInterval == true).

The first eight characters of the time element shall be fixed length containing two-digit values of hours, minutes, and seconds, separated by colons.

If the character following the hh:mm:ss field is a delimiter or terminator, the resolution of the time measurement unit shall be seconds. The corresponding value of a CBFE CBFETime_st::m_eRateEnumeration member is CLOCK_0. Illustration: "00:00:00X"

For higher rates designated characters following the seconds field encode the decimal fractions of seconds enumerated clock rates and indicate the number of digits following the indicator character.

If the character following the hh:mm:ss field is one of the lower-case rate indicators the time measurement unit shall correspond to the value of a CBFERate_et enumeration in the CBFETime_st::m_eRateEnumeration member and the number of digits of the decimal fractions of seconds shall be as shown in the following table.

Character Encoding	CBFE rate CBFERate_et	rate resolution	digits
none	CLOCK_0	1/1 second	none
t	CLOCK_1	1/10 tenths	1
h	CLOCK_2	1/100 hundredths	2
m	CLOCK_3	1/1000 Millisecond	3
u	CLOCK_6	1/1000000 Microsecond	6
h	CLOCK_7	1/10000000 100-Nanosecond	7
n	CLOCK_9	1/1000000000 Nanosecond	9
p	CLOCK_12	1/1000000000000 Picosecond	12

f	CLOCK_15	1/1000000000000000 Femtosecond	15
a	CLOCK_18	1/1000000000000000000 Attosecond	18

Example fragment:
```
00:00:00h1234567X
        ^ rate indicator CLOCK_7
```

5.4.1.1 UTC accurate local date and time-of-day

If the Time Element appears in combination with the Date Element and the TOD Count Mode is TOD_LEAPSECOND_MIDNIGHT or TOD_LEAPSECOND_UTC_UTC or TOD_LEAPSECOND_UTC_NTP or TOD_LEAPSECOND_UTC_POSIX the Time Element shall represent the time-of-day portion of a complete UTC accurate local date and time-of-day time-point including DST and.or UTC-offset Shift metadata if applicable. See Date Element and DST Element.

The corresponding CBFE binary data structures are CBFETime_st., CBFELocalDate_st, and CBFEDst_st.

Delimiters	Description	CBFE member variable state
D and T plus applicable metadata delimiters	UTC accurate local date and time-of-day time point	`CBFETime_st::m_bIsInterval == false` `CBFELocalDate_st::m_eTODMode == TOD_LEAPSECOND_MIDNIGHT or TOD_LEAPSECOND_UTC_UTC or TOD_LEAPSECOND_UTC_NTP or TOD_LEAPSECOND_UTC_POSIX`

Example:
```
D1972-06-30T19:59:60U-04Zamerica/new_yorkAedtV2021aL00*Ss+01cMuX
```

5.4.1.2 UTC accurate local date and time having no relation to date

If the time element appears in combination with the Date Element and the TOD Count Mode is TOD_NONE, character Ma ,it shall represent a time having no relation to the UTC accurate local date. The DST Element is not allowed. See Date Element. No

The corresponding CBFE binary data structures are CBFETime_st., CBFELocalDate_st, and CBFEDst_st.

Delimiters	Description	CBFE member variable state
D and T plus applicable metadata delimiters	Time-of-day unrelated to UTC date	`CBFETime_st::m_bIsInterval == false` `CBFELocalDate_st::m_eTODMode == TOD_NONE`

Example:
```
D1972-06-30T12:34:56U-04Zamerica/new_yorkAedtV2021aL00MaX
```

5.4.1.3 Time point less than 86400s

If the Time Element appears alone delimited by uppercase "T" it shall represent a time point less than 86400s.

Delimiter	Description	CBFE member variable state
T	Time point ((T)ime) less than 24 hours (< 86400 seconds) with no relation to date	`CBFETime_st::m_bIsInterval == false`

Example:
`T23:59:59X` - time-point (Time) in seconds

5.4.1.4 Time point equal or greater than to 86400s

If the Time Element appears in combination with an Event Element it shall represent a time point greater than or equal to 86400s and be delimited by uppercase "T". See Event Element.

Delimiters	Description	CBFE member variable state

E and T	Time point ((E)vent) 24 hours or greater (>= 86400 seconds) with no relation to date	CBFETime_st::m_bIsInterval == false

Example:
E123T23:59:59X - time-point (Event) >= 24 hours in seconds

5.4.1.5 Interval less than 86400s

If the Time Element appears alone delimited by uppercase "I" it shall represent an interval less than 86400s.

Delimiter	Description	CBFE member variable state
I	Interval ((I)nterval) less than 24 hours (< 86400 seconds)	CBFETime_st::m_bIsInterval == true.

Example:
I23:59:59u999999X - interval (Interval) < 24 hours in microseconds

5.4.1.6 Interval equal or greater than 86400s

If the Time Element appears in combination with an Period Element it shall represent an interval equal or greater than 86400s and be delimited by uppercase "I". See Period Element.

Delimiters	Description	CBFE member variable state
P and I	Interval ((P)eriod) 24 hours or greater (>= 86400 seconds)	CBFETime_st::m_bIsInterval == true.

Example:
P123I23:59:59u999999X - interval (Period) >= 24 hours in microseconds

Examples summary:
```
T23:59:59X                - time-point (Time) in seconds
T23:59:59m999X            - time-point (Time) in milliseconds
T23:59:59n999999999X      - time-point (Time) in nanoseconds
I23:59:59X                - interval (Interval) < 24 hours in seconds
I23:59:59u999999X         - interval (Interval) < 24 hours in microseconds
E1T23:59:59X              - time-point (Event) >= 24 hours in seconds
E12T23:59:59m999X         - time-point (Event) >= 24 hours in  milliseconds
E123T23:59:59n999999999X  - time-point (Event) >= 24 hours in nanoseconds
P1I23:59:59X              - interval (Period) >= 24 hours in seconds
P2I23:59:59u999999X       - interval (Period) >= 24 hours in microseconds
D1972-06-30T19:59:60U-05:00Znew_yorkV2016cL10*Sc4sMuX - Local Date and time
```

See CBFE.h,
```
    CBFETime_st
    CBFERATE_et
```
See CCct.h
```
   CCct::SetIntervalFromSecondsFrac_st()
   CCct::Set24HourTimepointFromSecondsFrac_st
```
See CCcf.cpp,
```
   CCcf::SetTimeFromCCbf()
   CCcf::ParseTimeSetCCbf()
```

5.4.2 Event Element

Encodes a 24 hour period (86400 seconds) count value. If given, shall be used in combination with a Time Element to represent a time interval equal to or greater than 86400 seconds. See Time Element.

The term "24 hour period" is used in this context to avoid the word "day" because only UTC, with its occasional 86401 second leap-second days, represents accurate calendar dates. An Event Element indicates a count of fixed-length 86400 second periods, not UTC days.

The corresponding CBFE binary data structure is CBFE24HourPeriod_st.

This is a range of approximately 27378 years, far greater than the approximate 3000 year range supported by Common Calendar. The corresponding CBFE 21-bit data member CBFE24HourPeriod_st:: m_ul24HourPeriods has a range of 2^21-bits = 2097152 MAX.

Delimiters	Description	CBFE member variable state
E and T	Time point ((E)vent) 24 hours or greater (>= 86400 seconds) with no relation to date	CBFETime_st::m_bIsInterval == false

Examples:

```
E1T23:59:59X             - time-point (Event) >= 24 hours in seconds
E12T23:59:59m999X        - time-point (Event) >= 24 hours in  milliseconds
E123T23:59:59n999999999X - time-point (Event) >= 24 hours in nanoseconds
```

See CBFE.h, CBFE24HourPeriod_st
CBFE24HourPeriod_st::m_ul24HourPeriods
See CCct.h
`CCct::SetIntervalFromSecondsFrac_st()`
`CCct::Set24HourTimepointFromSecondsFrac_st`
See CCcf.cpp, CCcf::SetCcfFromCCbf()
CCcf::SetCcfFromCCbf_Interval()
CCcf::SetCcfFromCCbf_Timepoint()
CCcf::ParseIntervalSetCCbf()
CCcf::Parse24HourIntervalSetCCbf()
CCcf::Parse24HourEventSetCCbf()

5.4.3 Period Element

Encodes a 24 hour interval (86400 seconds) value. If given, shall be used in combination with an Interval Element to represent a interval equal to or greater than 86400 seconds. See Interval Element.

The corresponding CBFE binary data structure is CBFE24HourPeriod_st.

Delimiters	Description	CBFE member variable state
P and I	Interval ((P)eriod) 24 hours or greater (>= 86400 seconds)	CBFETime_st::m_bIsInterval == true.

Examples:

```
P1T23:59:59X             - interval (Period) >= 24 hours in seconds
P2T23:59:59u999999X      - interval (Period) >= 24 hours in microseconds
```

See CBFE.h, CBFE24HourPeriod_st
CBFE24HourPeriod_st::m_ul24HourPeriods
See CCct.h
`CCct::SetIntervalFromSecondsFrac_st()`
`CCct::Set24HourTimepointFromSecondsFrac_st`
See CCcf.cpp, CCcf::SetCcfFromCCbf()
CCcf::SetCcfFromCCbf_Interval()
CCcf::SetCcfFromCCbf_Timepoint()
CCcf::ParseIntervalSetCCbf()
CCcf::Parse24HourIntervalSetCCbf()
CCcf::Parse24HourEventSetCCbf()

5.4.4 Date Element

Encodes the local date.

The Date Element shall be fixed length 11 characters including delimiter in the form DYYYY-MM-DD, as shown in the following table.

Delimiter	Year (YYYY)	Dash	Month (MM)	Dash	Day (DD)
D	4 digit year (YYYY)	-	2 digit month (MM) with leading zeros	-	2 digit day of month (DD) with leading zeros

Example fragment:

If the optional YYYY-MM-DD Date Element is present it shall represent the complete local calendar date and time-of-day together with sufficient metadata to fully describe local data and time. The optional DST Element shall add Daylight Saving parameters if applicable and the UTC-Offset Shift Element shall add UTC-Offset shift parameters if applicable.

If date is given it shall be accompanied by

- Time Element (time-of-day)
- Local UTC Offset Element
- Time Zone Element
- Tz Database Version Element
- Leap-seconds Element
- DST Element (if applicable)
- UTC-Offset Shift Element (if applicable)
- TOD Count Mode Element

A Date element together with its required Time element and metadata elements describe a time-point, indicated by CBFE `CBFETime_st::m_bIsInterval == false`.

A CBF shall include the CBFLocalDate_st structure, indicated by CBFTime_st::m_bLocalDateExt == true

If the YYYY-MM-DD date element is present the hh:mm:ss time element values shall represent the time-of-day portion of a complete UTC accurate local date and time on the date indicated by the YYYY-MM-DD values of the date element, including common-days of 86400-seconds and leap-second-days of 86401 seconds. DST information shall be provided by the DST element, if applicable.

The sequence of YMDhms values shall be governed by the TOD Count Mode CBFTodCountMode_et enumerated values of CBFLocalDate_st::m_eTODMode and DST Count Mode CBFDstCountMode_et CBFDst_st::m_eDSTCountMode (if applicable) in effect.

See Common Calendar Local Timescales,:
4.1 Daylight Saving Time (DST) Count Mode.
4.2 Time-of-day (TOD) Count Mode and leap-second Introduction

Identically to CBF CBFLocalDate_st and CBFDst_st parameters, the values and metadata of the date and DST elements (if applicable) shall represent local time as known to the emitting system in all cases.

Construction of CCFE YMDhms from CBF binary data shall employ the algorithms described by the YMDhms API together with logic to apply DST and/or UTC-offset shifts if applicable. The YMDhms values will depend on CBFLocalDate_st::m_eTODMode and CBFDst_st::m_eDSTCountMode if applicable.

CCFE delimiter characters and encoding characters are explicitly defined in CCTLib/CCFE.h.

The details of the CBF to CCFE conversion are shown in:
CCcf.cpp, CCcf::SetCcfFromCCbf()
CCcf::SetCcfFromCCbf_TOD_LEAPSECOND_UTC()
CCcf::SetCcfFromCCbf_TOD_LEAPSECOND_MIDNIGHT()

Parsing of a CCFE string and populating a CBF binary is shown in: CCcf.cpp, CCcf::ParseDateSetCbf()

The corresponding CBF components are:
CBFTime_st
CBFLocalDate_st
CBFDst_st (if applicable).
CBFUtcShift_st (if applicable).

CBF CBFLocalDate_st records the date in a compound counter:
CBFLocalDate_st::m_DateTaiUtc_st.m_ui1970DayNumber
CBFLocalDate_st::m_DateTaiUtc_st.m_ui1970TAI_UTC

See CCcf.cpp, CCcf::SetCcfFromCCbf()
CCcf::SetCcfFromCCbf_TOD_LEAPSECOND_MIDNIGHT()
CCcf::SetCcfFromCCbf_TOD_LEAPSECOND_UTC()
CCcf::ParseDateSetCbf()

Examples:

```
D1972-06-30T19:59:60U-04Zamerica/new_yorkAedtV2021aL00*Ss+01cMuX
```
- 1972-06-30T19:59:60 seconds, date and time
- U-04 - UTC-offset
- Zamerica/new_yorkAedtV2021aL0 - time zone
- Aedt - Posix environment time zone abbreviation
- V2021a - Tz Database version
- L0 - Leap-seconds
- * - Is Leap-second indicator
- Ss+01c - DST
  - s - DST in effect (summer time)
  - +01 - 01:00 DST bias
  - c - DST Count Mode DSTCOUNTMODE_CONVENTIONAL
- Mu - TOD Count Mode TOD_LEAPSECOND_UTC_UTC
- X - terminator

```
D1972-07-01T00:59:60U+1Zeurope/berlinAcetV2021aL00*SwcMuX
```
- 1972-07-01T01:59:60 seconds, date and time
- U+1 - UTC offset
- Zeurope/berlin - time zone
- V2021a - Tz Database version
- Acet - Posix environment time zone abbreviation
- L0 - Leap-seconds
- * - Is Leap-second indicator
- Swc - DST
  - w - DST not in effect (winter time)
  - c - DST Count Mode DSTCOUNTMODE_CONVENTIONAL
- Mu - TOD Count Mode TOD_LEAPSECOND_UTC_UTC
- X - terminator

5.4.5 TOD Count Mode Element

Encodes the TOD Count Mode value.

TOD Count Mode indicates the relation between the DYYY-MM-DD and hh:mm:ss portions of the CCFE and the resulting sequence of YMDhms representation.

Required if, and applicable only if, the Date Element "D" is given.

The TOD Count Mode Element shall be a fixed length 2 character string including the M (TOD (M)ode) delimiter and a single lower-case encoding character as show below.

Delimiter	TOD Count Mode Indicator
M	one character encoding as shown in the following table

TOD Count Mode shall be indicated by a single lower-case character as shown in the following table.

Character Encoding	TOD Count Mode CBFTodCountMode_et	Description
a	TOD_NONE	Time has no relation to date or time-of-day
m	TOD_LEAPSECOND_MIDNIGHT	Leap-seconds introduced at midnight on local timescales (Rolling leap-second)
u	TOD_LEAPSECOND_UTC_UTC	Leap-seconds introduced simultaneous with UTC on local timescales. Leap-second label 23:59:60 (UTC specification)
n	TOD_LEAPSECOND_UTC_NTP	Leap-seconds introduced simultaneous with UTC on local timescales. Leap-second label 59:59:59 ("freeze")

p	TOD_LEAPSECOND_UTC_POSIX	Leap-seconds introduced simultaneous with UTC on local timescales. Leap-second label 00:00:00 ("roll over and reset")
g	TOD_24HOUR_DAY_DATE	86400-second-days of calendar (leap-seconds unknown or unavailable)
	TOD_NA	not set or logic error (default)

Example fragments:
`Ma` - TOD_NONE
`Mm` - TOD_LEAPSECOND_MIDNIGHT
`Mu` - TOD_LEAPSECOND_UTC_UTC
`Mn` - TOD_LEAPSECOND_UTC_NTP
`Mp` - TOD_LEAPSECOND_UTC_POSIX
`Mg` - TOD_24HOUR_DAY_DATE

CCFE delimiter characters and encoding characters are explicitly defined in CCTLib/CCFE.h.

The corresponding CBF member is CBFLocalDate_st::m_eTODMode.

See CBF.h, CBFTodCountMode_et
CBFLocalDate_st::m_eTODMode
See CCcf.cpp, CCcf::SetCcfFromCCbf()
CCcf::ParseTODModeSetCCbf()

5.4.6 UTC-Offset Element

Encodes the local time UTC offset from UTC

The value represents the UTC-offset of local time from UTC. It is the sum of the current UTC-offset of CCbf::CBFLocalDate_st.m_lUTCOffset (called STDOFF in Tz Database) plus DST bias if applied, and any UTC-offset shift if applicable. It is sometimes called "current local offset", and this is called GMTOFF in Tz Database. It is consistent with ISO 8601 representations.

The value is a variable length +-hh:mm:ss representation. A sub-field may appear to encode (rare) UTC-offset shifts. See UTC-offset Shifts.

The local UTC-Offset element shall be delimited with upper-case "U" followed by "+" (East) or "-" (West) followed by a variable length hh:mm:ss representation.

A two digit hour shall always be present.
If minutes is non-zero a two digit minute shall be present separated from hours by a ":" (colon).
If seconds is non-zero a two digit seconds shall be present separated from minutes by a ":" (colon).

Delimiter	UTC-offset sign	UTC-offset	
U	+ or -	2 digit hour	all cases
		":" 2 digit minute	If minutes is non-zero
		":" 2 digit second	If seconds is non-zero

Example fragments:

```
U+00 U-05 U+01 U-10 U+14
U-00:01 U+07:30
U-00:01:15 U+07:30:23
```

The corresponding CBF member is CBFLocalDate_st::m_lUTCOffset

See CBF.h, CBFLocalDate_st::m_lUTCOffset
See CCcf.cpp, CCcf::SetUTCOffsetAndZoneFromCCbf()
CCcf::ParseUTCOffsetSetCCbf()

5.4.6.1 UTC-Offset Shift Sub-fields

Encodes (rare) cases where a time zone has shifted the UTC-offset (STDOFF).

5.4.6.1.1 UTC-offset shift East or West

The UTC-Offset Shift Sub-field encodes the UTC-offset direction (East or West), shift amount and shift time-of-day.

e (TZDUtcShiftDay_et UTCSHIFT_EAST, East is positive shift)
w (TZDUtcShiftDay_et UTCSHIFT_WEST, West is negative shift)
hh:mm:ss variable length UTC-offset shift amount
+hh:mm:ss variable length UTC-offset shift time-of-day (local time)

if UTC-offset shift is East (UTCSHIFT_EAST) a lower-case "e" shall be appended.
if UTC-offset shift is West (UTCSHIFT_WEST) a lower-case "w" shall be appended.

Character Encoding	Description	CBF enumeration TZDUtcShiftDay_et
e	UTC-offset shift East - positive	UTCSHIFT_EAST
w	UTC-offset shift West - negative	UTCSHIFT_WEST

5.4.6.1.2 UTC-offset shift amount

if UTC-offset shift is positive the value shall be preceded by "+".
if UTC-offset shift is negative the value shall be preceded by "-".
A two digit hour shall always be present.
If minutes is non-zero a two digit minute shall be present separated from hours by a ":" (colon).
If seconds is non-zero a two digit seconds shall be present separated from minutes by a ":" (colon).

	UTC-offset shift sign	UTC-offset shift	
	+ or -	two digit hour	all cases
		":" two digit minute	If minutes is non-zero
		":" two digit second	If seconds is non-zero

5.4.6.1.3 UTC-offset shift time-of-day (local time)

if UTC-offset shift time-of-day shall be preceded by "+".
A two digit hour shall always be present.
If minutes is non-zero a two digit minute shall be present separated from hours by a ":" (colon).
If seconds is non-zero a two digit seconds shall be present separated from minutes by a ":" (colon).

UTC-offset shift time-of-day delimiter	UTC-offset shift	
+	two digit hour	all cases
	":" two digit minute	If minutes is non-zero
	":" two digit second	If seconds is non-zero

typedef enum TZDUtcShiftDay_et
See CBF.h, CBFLocalDate_st::m_bUtcShiftExt
See CBF.h, CBFUtcShift_st
See CCcf.cpp, CCcf::SetUTCOffsetAndZoneFromCCbf()
CCcf::ParseUTCOffsetSetCCbf()
See TzDatabaseApi.h

Example fragments:
`U+00w00:01:15+00:01:15` - shift West by 00:01:15 at +00:01:15
`U+02e01+03` - shift East by 01:00:00 at +03:00:00
Example:
`D2011-03-27T01:59:59U+03e01+02Zeurope/moscowAmskV2021aL24MuX`
`D2011-03-27T03:00:00U+04e01+02Zeurope/moscowAmskV2021aL24MuX`

5.4.7 Time Zone Element

Encodes the Tz Database time zone identifier name.

The Time Zone Element shall be delimited with Z ((Z)one) followed by a variable length string of lower-case Tz Database time zone name identifiers including and forward slash "/". For example, "America/New_York" becomes "america/new_york".

Delimiter	Time Zone name
Z	variable length string as lower-case of Tz Database time zone name identifiers

Example fragments:
```
Zetc/utc
Zamerica/new_york
Zamerica/los_angles
Zeurope/london
```

Example
```
D2024-03-10T03:00:00m000U-04Zamerica/new_yorkAedtV2021aL27S01t01a02cMuX
```

CCFE delimiter characters and encoding characters are explicitly defined in CCTLib/CCFE.h.

The corresponding CBF member is CBFLocalDate_st::m_TZDTimeZone_st.m_uneTZDZoneName_et

See TzDatabaseApi.h, TZDZoneName_et
See CTzZoneTable.cpp, CTzZoneTable::m_CTzZone_stTable[]
See CBF.h, CBFLocalDate_st::m_TZDTimeZone_st
See CCcf.cpp,
CCcf::SetUTCOffsetAndZoneFromCCbf()
CCcf::ParseZoneSetCCbf()
See Common Calendar Time Zone API

5.4.8 Posix Abbreviation Name Element

Encodes the Posix-time TZ environment variable abbreviated time zone name as defined by Tz Database.

The Posix Abbreviation Name Element shall be delimited with A ((A)bbriveation) followed by a variable lower case string.

Delimiter	Posix Abbreviated Time Zone name
A	variable length string as lower-case of Posix abbreviated time zone name

Example fragments:
```
Aest
Aedt
```

Example
```
D2024-03-10T03:00:00m000U-04Zamerica/new_yorkAedtV2021aL27S01t01a02cMuX
```

The corresponding CBF member is m_aCBFChar_st16Abbr[16];

See CCcf.cpp,
CCcf::SetUTCOffsetAndZoneFromCCbf()
CCcf::ParseDstSetCCbf()

5.4.9 Posix Abbreviation Name Change Element

Encodes the Posix-time abbreviated name changes.

Most transitions have some change of parameters and the abbreviated name naturally follows. But some examples have *only* a change to the abbreviated name. This type of transition is supported by this element.

The Abbreviation Name Element shall be delimited with Q followed by a variable lower case string.

Delimiter	Posix Abbreviated Time Zone name
Q	variable length string as lower-case of Posix abbreviated time zone name

Example fragments:
```
Qewt_ept+00Aedt
Qewt_ept+00
```

Examples

```
D1945-08-14T18:59:59U-04America/new_yorkAewtQewt_ept+00V2021aL00S01cMuX
D1945-08-14T19:00:00U-04America/new_yorkAeptQewt_ept+00V2021aL00S01cMuX
```

The corresponding CBF member is CBFAbbrChange_st;

The presence of the Posix Abbreviation Name Change is indicated by
CBFLocalDate_st::TZDTimeZoneID_st:m_bCBFAbbrChangeExt

5.4.10 IANA Time Zone Database Version Element

Encodes the IANA Time Zone Database tzdata source files release year and release letter.

The IANA Time Zone Database Version Element shall be delimited with V ((V)ersion) followed by a four digit year and single character release letter.

Delimiter	IANA tzdata files release year	IANA tzdata files release letter
V	four digit year (YYYY)	one lower-case character

Example fragments:
```
V2021a
V2022d
```

Example
```
D2024-03-10T03:00:00m000U-04America/new_yorkAedtV2021aL27S01t01a02cMuX
```

CCFE delimiter characters and encoding characters are explicitly defined in CCTLib/CCFE.h.

The corresponding CBF members are
CBFLocalDate_st::m_TZDTimeZone_st.m_unTzDataReleaseYear
CBFLocalDate_st::m_TZDTimeZone_st.m_unTzDataReleaseLetter

See TzDatabaseApi.h, TZDDataRelease_st
TZDTimeZone_st
See CBF.h, CBFLocalDate_st::m_TZDTimeZone_st
See CCcf.cpp, CCcf::SetUTCOffsetAndZoneFromCCbf()
CCcf::ParseTzVersionSetCCbf()

5.4.11 Leap-seconds Element

Encodes the leap-second value. The leap-second value is TAI-UTC minus the initial 10s calibration offset between TAI and UTC. *The minimum leap-seconds value is 0 at and before the UTC1972 origin.*

The TOD Count Mode Element describes the method of leap-second introduction resulting in the YMDhms sequence. See TOD Count Mode Element.

If the TOD Count Mode Element value is TOD_NONE or TOD_24HOUR_DAY_DATE the leap-seconds Element shall not appear.

The leap-seconds Element shall be delimited with L ((L)eap) followed by a variable 2-n digit leap-second value. If the current date is a positive leap-second Day a plus sign shall be appended. If the current date is a negative leap-second Day a minus sign shall be appended. If the current time point lies at or within a leap-second an asterisk shall be appended.

Delimiter	Leap-second value	if IsLeapSecond	if IsLeapSecondDay (positive Leap-second)	if IsLeapSecondDay (negative Leap-second)
L	2-n digits	append * (asterisk)	append + (plus)	append - (minus)

Example fragments:
```
L25  - TAI-UTC offset minus 10s initial calibration
L25+ - Is Leap-second Day (positive leap-second)
L25* - Is Leap-second
L25- - Is Leap-second Day (negative leap-second)
```

Examples

```
------ 2015 Leap-second Day with TOD_LEAPSECOND_UTC_UTC mode ------
```

```
1) second before leap-second day    - not LS day -    L25
2) first second of leap-second day    is  LS day - +  L25+
3) second before leap-second          is  LS day - +  L25+
4) leap-second                        is  LS     - *  L25*
5) second after leap-second           is  LS day - +  L26+
6) last second of leap-second day     is  LS day - +  L26+
7) second after leap-second day       not LS day -    L26

1) D2015-06-29T23:59:59U-04Zamerica/new_yorkAedtV2021aL25Ss+01cMuX
2) D2015-06-30T00:00:00U-04Zamerica/new_yorkAedtV2021aL25+Ss+01cMuX
3) D2015-06-30T19:59:59U-04Zamerica/new_yorkAedtV2021aL25+Ss+01cMuX
4) D2015-06-30T19:59:60U-04Zamerica/new_yorkAedtV2021aL25*Ss+01cMuX
5) D2015-06-30T20:00:00U-04Zamerica/new_yorkAedtV2021aL26+Ss+01cMuX
6) D2015-06-30T23:59:59U-04Zamerica/new_yorkAedtV2021aL26+Ss+01cMuX
7) D2015-07-01T00:00:00U-04Zamerica/new_yorkAedtV2021aL26Ss+01cMuX
```

CCFE delimiter characters and encoding characters are explicitly defined in CCTLib/CCFE.h.

The corresponding CBF members are

```
CBFLocalDate_st::m_DateTaiUtc_st.m_nLeapsecsHigh;    // Positive leap-seconds
CBFLocalDate_st::m_DateTaiUtc_st.m_nLeapsecsLow;     // Positive leap-seconds
CBFLocalDate_st::m_DateTaiUtc_st.m_nLeapsecsNegHigh; // Negative Leap-seconds
CBFLocalDate_st::m_DateTaiUtc_st.m_lLeapsecsNegLow;  // Negative Leap-seconds
CBFLocalDate_st::m_bIsLeapSecond;          // is Leap-second
CBFLocalDate_st::m_bIsLeapSecondDay;       // is Leap-second day
CBFLocalDate_st::m_bIsLeapSecondNegative; // Leap-second is negative
```

See TaiUtcApi.h, DateTaiUtc_st
See CBF.h, `CBFLocalDate_st::m_DateTaiUtc_st`
`CBFLocalDate_st::m_bIsLeapSecond`
`CBFLocalDate_st::m_bIsLeapSecondDay`
`CBFLocalDate_st::m_bIsLeapSecondNegative`
See CCcf.cpp, `CCcf::SetTAIUTCFromCCbf()`
`CCcf::ParseTAIUTCSetCCbf()`

5.4.12 Daylight Saving Time (DST) Element

Encodes Daylight Saving Time (DST) parameters if applicable.

The DST element shall *not* appear if DST bias equals zero except when DST Transition Day sub-fields may be applicable.

The DST element shall be a variable length string delimited with upper-case "S" (daylight (S)aving) followed by appropriate sub-fields.

Delimiter	Daylight Saving Time
S	DST Bias if applicable and DST Transition Day if applicable

Note that CCT does not have a “tm_isdst” flag, as in traditional Posix-time and TzDb struct tm. Instead, only the DST Bias and DST Transition Day with DST Bias Change and DST Transition Change Time-of-day sub-fields are needed. This then supports unusual situations such as “double summertime” , DST bias and changes not equal to 1-hour. If traditional “tm_isdst” is needed an application may interrogate a CCT timestamp; if DST Bias is not zero, tm_isdst = true.

See CCcf.cpp, CCcf::SetDstMetadataFromCCbf()
CCcf::ParseDstSetCCbf()

5.4.12.1 Daylight Saving Time (DST) Bias Sub-field

Encodes the value of the Daylight Saving Time (DST) offset (called DST Bias) if DST is in effect.

If the DST bias amount is positive the value shall *not* include a sign indicator
If the DST bias amount is negative the value shall be preceded by "-".

A two digit hour shall always be present.

If minutes is non-zero a two digit minute shall be present separated from hours by a ":" (colon).
If seconds is non-zero a two digit seconds shall be present separated from minutes by a ":" (colon).

DST Bias	DST Bias value	
+ or -	two digit hour	all cases
	":" two digit minute	If minutes is non-zero
	":" two digit second	If seconds is non-zero

Corresponding CBF member CBFDstBias_st
See CBF.h, CBFDstBias_st

Example fragments:
```
S01c
S01:15c
S01:30:01c
S-01:30:01c
```

Example
```
D2024-04-00T00:00:00m000U-04Zamerica/new_yorkAedtV2021aL27S01cMuX
```

5.4.12.2 DST Transition Day Sub-fields

If the day has a DST transition the DST Bias element shall appear (its value may be zero bias) together with the DST Transition Day sub-fields

5.4.12.2.1 DST Bias Change Sub-field

Indicates the value of the DST change.
Shall be delimited by lower case “t” followed by and hms indicator. (“t” for “transition”)

A two digit hour shall always be present.
If minutes is non-zero a two digit minute shall be present separated from hours by a ":" (colon).
If seconds is non-zero a two digit seconds shall be present separated from minutes by a ":" (colon).

two digit hour	all cases
":" two digit minute	If minutes is non-zero
":" two digit second	If seconds is non-zero

Corresponding CBF member is
CBFDstTransDay_st:: m_nDstBiasChangeLow:16/ m_nDstBiasChangeHigh:2
See CBF.h, CBFDstTransDay_st

5.4.12.2.2 DST Transition Change Time-of-day Sub-field

Indicates the time-of-day of the DST transition with respect to local time.

Shall be delimited by lower case “a” followed by an hms indicator. (“a” for “at”)

A two digit hour shall always be present.
If minutes is non-zero a two digit minute shall be present separated from hours by a ":" (colon).
If seconds is non-zero a two digit seconds shall be present separated from minutes by a ":" (colon).

two digit hour	all cases
":" two digit minute	If minutes is non-zero
":" two digit second	If seconds is non-zero

Corresponding CBF member is CBFDst_st::m_ulDSTChangeTime

Example fragments

```
S01t01a02c
S01t-01a02c
S00t-01a02c
S-01t01a01c
S00t02a00:01c
```

```
S02t-02a00:01c
S02t-01a04:31:19c
S01t-01a24c
S00t00:00:01a23:59:58c
```

Examples

D2024-03-09T23:59:59U-05Zamerica/new_yorkAestV2021aL27MuX
D2024-03-10T01:59:59U-05Zamerica/new_yorkAestV2021aL27S00t01a02cMuX
D2024-03-10T03:00:00U-04Zamerica/new_yorkAedtV2021aL27S01t01a02cMuX
D2024-03-11T00:00:00U-04Zamerica/new_yorkAedtV2021aL27S01cMuX

D2024-11-02T23:59:59U-04Zamerica/new_yorkAedtV2021aL27S01cMuX
D2024-11-03T01:59:59U-04Zamerica/new_yorkAedtV2021aL27S01t-01a02cMu
D2024-11-03T01:00:00U-05Zamerica/new_yorkAestV2021aL27S00t-01a02cMuX
D2024-11-04T00:00:00U-05Zamerica/new_yorkAestV2021aL27MuX

D2024-03-30T23:59:59U+00Zeurope/dublinAgmtV2021aL27S-01cMuX
D2024-03-31T00:59:59U+00Zeurope/dublinAgmtV2021aL27S-01t01a01cMuX
D2024-03-31T02:00:00U+01Zeurope/dublinAistV2021aL27S00t01a01cMuX
D2024-04-01T00:00:00U+01Zeurope/dublinAistV2021aL27MuX

D2024-10-26T23:59:59U+01Zeurope/dublinAistV2021aL27MuX
D2024-10-27T01:59:59U+01Zeurope/dublinAistV2021aL27S00t-01a02cMuX
D2024-10-27T01:00:00U+00Zeurope/dublinAgmtV2021aL27S-01t-01a02cMuX
D2024-10-28T00:00:00U+00Zeurope/dublinAgmtV2021aL27S-01cMuX

5.4.12.3 DST Count Mode Sub-field

Encodes the DST Count Mode in effect.

CCT supports two DST Count Modes, either "conventional", where the DST transition occurs at the typical time-of-day given by tzdb rules, or "uninterrupted", where the DST transition occurs at the end of the day. See Common Calendar Local Timescales, Daylight Saving Time (DST) Count Mode..

If the Count Mode is "conventional" (DSTCOUNTMODE_CONVENTIONAL) a lower-case "c" shall be appended.

If the Count Mode is "uninterrupted" (DSTCOUNTMODE_UNINTERRUPTED) a lower-case "c" shall be appended.

Character Encoding	CBF enumeration CBFDstCountMode_et
c	DSTCOUNTMODE_CONVENTIONAL
u	DSTCOUNTMODE_UNINTERRUPTED

Corresponding CBF member CBFDst_st::m_eDSTCountMode
See CBF.h, CBFDstCountMode_et

Examples:

```
------------------- DST Onset Transition Day --------------------
1) second before Onset day     DST is not in effect - w  Swc
2) first second of Onset day   DST is not in effect - w  Swo02+01c
3) second before Onset         DST is not in effect - w  Swo02+01c
4) Onset transition            DST is     in effect - s  Sso02+01c
5) last second of Onset day    DST is     in effect - s  Sso02+01c
6) second after Onset Day      DST is     in effect - s  Ss+01c

1) D2015-03-07T23:59:59U-05Zamerica/new_yorkAestV2021aL25SwcMuX
2) D2015-03-08T00:00:00U-05Zamerica/new_yorkAestV2021aL25Swo02+01cMuX
3) D2015-03-08T01:59:59U-05Zamerica/new_yorkAestV2021aL25Swo02+01cMuX
4) D2015-03-08T03:00:00U-04Zamerica/new_yorkAedtV2021aL25Sso02+01cMuX
5) D2015-03-08T23:59:59U-04Zamerica/new_yorkAedtV2021aL25Sso02+01cMuX
6) D2015-03-09T00:00:00U-04Zamerica/new_yorkAedtV2021aL25Ss+01cMuX
```

```
------------------- DST Retreat Transition Day -------------------
1) second before Retreat day   DST is     in effect - s  Ss+01c
2) first second of Retreat day DST is     in effect - s  Ssr02+01c
3) second before Retreat       DST is     in effect - s  Ssr02+01c
4) Retreat transition          DST is not in effect - w  Swr02+01c
5) last second of Retreat day  DST is not in effect - w  Swr02+01c
6) second after Retreat Day    DST is not in effect - w  Swc

1) D2015-10-31T23:59:59U-04Zamerica/new_yorkAedtV2021aL26Ss+01cMuX
2) D2015-11-01T00:00:00U-04Zamerica/new_yorkAedtV2021aL26Ssr02+01cMuX
3) D2015-11-01T01:59:59U-04Zamerica/new_yorkAedtV2021aL26Ssr02+01cMuX
4) D2015-11-01T01:00:00U-05Zamerica/new_yorkAestV2021aL26Swr02+01cMuX
5) D2015-11-01T23:59:59U-05Zamerica/new_yorkAestV2021aL26Swr02+01cMuX
6) D2015-11-02T00:00:00U-05Zamerica/new_yorkAestV2021aL26SwcMuX
```

5.5 Assembly and Order

A CCFE string shall begin with one of D (Date), T (Time), E (Event), I (Interval), or P (Period).

Required elements and order of presentation for each of the supported meanings are specified in sections below.

5.5.1 UTC accurate local date and time-of-day

To represent a UTC accurate local date and time time point a CCFE shall include elements in the order shown in the following table:

Order	Delimiter	Element	
1	D	Date	required
2	T	Time-of-day	required
3	U	UTC offset	required
4	Z	Zone (time zone)	required
5	A	Posix Abbreviation	required
5	V	Version of Tz Database	required
7	L	TAI-UTC (Leap-seconds)	required
8	S	DST	if applicable
9	M	TOD Count Mode shall be TOD_LEAPSECOND_MIDNIGHT or TOD_LEAPSECOND_UTC_UTC or TOD_LEAPSECOND_UTC_NTP or TOD_LEAPSECOND_UTC_POSIX	required
10	X	terminator	required
	E	Event (24 hour period)	excluded
	I	Interval (24 hour period)	excluded
	P	Period (24 hour period)	excluded

D T U Z A V L S M X required
E I P excluded

Corresponding CBF variable states

```
   CBFTime_st::m_bIsInterval = false
   CBFTime_st::m_bLocalDateExt == true
   CBFTime_st::m_b24HourPeriodExt == false
   CBFLocalDate_st::m_eTODMode =
      TOD_LEAPSECOND_MIDNIGHT or
      TOD_LEAPSECOND_UTC_UTC or
      TOD_LEAPSECOND_UTC_NTP or
      TOD_LEAPSECOND_UTC_POSIX
   CBFLocalDate_st::m_bDSTExt = true
   CBFDst_st::m_eDSTCountMode =
      DSTCOUNTMODE_CONVENTIONAL or
      DSTCOUNTMODE_UNINTERRUPTED
```

See Common Calendar Local Timescales, 4.2 Time-of-day (TOD) Count Mode and leap-second Introduction, and 4.1 Daylight Saving Time (DST) Count Mode

5.5.2 UTC accurate local date and time with no relation to the date

To represent a UTC accurate local date and a time with no relation to the date a CCFE shall include elements in the order shown in the following table:

Order	Delimiter	Element	
1	D	Date	required
2	T	Time-of-day	required
3	U	UTC offset	required
4	Z	Zone (time zone)	required
5	V	Version of Tz Database	required
6	L	TAI-UTC (Leap-seconds)	required
7	M	TOD Count Mode shall be TOD_NONE	required
8	X	terminator	required
	S	DST	excluded
	E	Event (24 hour period)	excluded
	I	Interval (24 hour period)	excluded
	P	Period (24 hour period)	excluded

D T U Z V L M X required
S E I P excluded

Corresponding CBF variable states

```
CBFTime_st::m_bIsInterval == false
CBFTime_st::m_bLocalDateExt == true
CBFTime_st::m_b24HourPeriodExt == false
CBFLocalDate_st::m_eTODMode == TOD_NONE
CBFLocalDate_st::m_bDSTExt == false
```

See Common Calendar Local Timescales, 4.2 Time-of-day (TOD) Count Mode

5.5.3 Time point less than 86400s

To represent a time point less than 24 hours (< 86400s) a CCFE shall include elements in the order shown in the following table:

Order	Delimiter	Element	
1	T	Time	required
2	X	terminator	required
		all others	excluded

T required
D E I U Z V L S M excluded

Indicated time shall be less than 86400 seconds.

Corresponding CBF variable states

```
CBFTime_st::m_bIsInterval == false
CBFTime_st::m_bLocalDateExt == false
CBFTime_st::m_b24HourPeriodExt == false
```

5.5.4 Time point equal or greater than 86400s

To represent a time point 24 hours or greater (>= 86400s) a CCFE shall include elements in the order shown in the following table:

Order	Delimiter	Element	
1	E	Event	
2	T	Time	required
3	X	terminator	required
		all others	excluded

E and T required

D I U Z V L S M excluded

Corresponding CBF variable states
```
CBFTime_st::m_bIsInterval == false
CBFTime_st::m_bLocalDateExt == false
CBFTime_st::m_b24HourPeriodExt == true
```

5.5.5 Interval less than 86400s

To represent an interval less than 24 hours (< 86400s) a CCFE shall include elements in the order shown in the following table:

Order	Delimiter	Element	
1	I	Time	required
2	X	terminator	required
		all others	excluded

T required
D E P U Z V L S M excluded

Indicated interval shall be less than 86400 seconds.

Corresponding CBF variable states
```
CBFTime_st::m_bIsInterval == true
CBFTime_st::m_bLocalDateExt == false
CBFTime_st::m_b24HourPeriodExt == false
```

5.5.6 Interval equal or greater than 86400s

To represent an interval 24 hours or greater (>= 86400s) a CCFE shall include elements in the order shown in the following table:

Order	Delimiter	Element	
1	P	Period	
2	I	Time	required
3	X	terminator	required
		all others	excluded

P and T required
D E I U Z V L S M excluded

Corresponding CBF variable states
```
CBFTime_st::m_bIsInterval == true
CBFTime_st::m_bLocalDateExt == false
CBFTime_st::m_b24HourPeriodExt == true
```

5.6 Geostamp

CCTC may include location, the geographical coordinates. This transforms a CCTC timestamp into a GeoStamp.

The corresponding CBFC component is CCbfC::m_CBFLocation_st. as declared in TzDatabaseApi.h, CBFLocation_st. The presence of location data is signaled by:
m_CCTCLocalDate_st.m_CCTCZoneID_st.m_bCBFLocationExt == true

CCTC supports two forms of location:

If m_CCTCParams_st.m_CBFLocation_st.m_bSourceIsExtern == false the data is in time zone's principal location in ISO 6709 form as given by TzDb zone.tab. The character formatting shall be:

Latitude and longitude as sign-degrees-minutes-seconds format, either ±DDMMÂ±DDDMM or ±DDMMSSÂ±DDDMMSS.

Latitude is preceded by a sign character.
A plus sign (+) denotes northern hemisphere or the equator.
A minus sign (-) denotes southern hemisphere.

Longitude is preceded by a sign character.
A plus sign (+) denotes East or the prime meridian.

A minus sign (-) denotes West.

Altitude is not given by TzDb zone.tab.

If the application has called CCctC::SetLocation() the data is in NMEA GPGGA form and m_CCTCParams_st.m_CBFLocation_st.m_bSourceIsExtern == true.

2 - The coordinates as supplied by NMEA 0183 GPGGA sentences. Latitude and longitude in degrees, minutes and decimal fractions of minutes.

Latitude is preceded by a sign character.
A plus sign (+) denotes northern hemisphere or the equator.
A minus sign (-) denotes southern hemisphere.

Longitude is preceded by a sign character.
A plus sign (+) denotes East or the prime meridian.
A minus sign (-) denotes West.

5.6.1 Location Element

Encodes geographic coordinates.

CCTC carries coordinates in the form specified by National Marine Electronics Association (NMEA), NMEA 0183 Interface Standard, GPGGA., GGA Global Positioning System Fix Data. Time, Position and fix related data for a GPS receiver. The NMEA data is translated into the CBFLocation_st structure in the CBFC binary format. The CBFC data is reflected in the CCFC character format in the Location Element field.

The Location Element shall be a variable length string delimited with upper-case "C" ((C)oordinates) followed by appropriate sub-fields

Delimiter	Sub-fields			
	External	Latitude	Longitude	Altitude
C	external source or tzdb defaults	degrees, minutes, micro-minutes	degrees, minutes, micro-minutes	meters, centimeters

CCTC supports coordinates from two sources, either input from external source such as GPS, or from TzDb time zone default coordinates as given in tzdb zone.tab.

if external source a lower-case "e" ((e)xternal) shall be appended.
if not external source a lower-case "z" ((z)one) shall be appended.

The latitude sub-field shall be delimited by lower-case "t". (la(t)itude) followed by "+" or "-", 2 digits of degrees, 2 digits of seconds, a "." (period) separator, and 6 digits microminutes.

The longitude sub-field shall be delimited by lower-case "g". (lon(g)itude) followed by "+" or "-", 2 digits of degrees, 2 digits of seconds, a "." (period) separator, and 6 digits microminutes.

The altitude sub-field shall be delimited by lower-case "a". ((a)titude) followed by "+" or "-", 1-n meters, (period) separator, and 6 digits centimeters.

Corresponding CBFC member CBFLocation_st
See TzDatabaseAPI.h, CBFLocation_st
See CCctC.h, CCctC.cpp class CCctC
int CCctC::SetLocation(char* psLatitude, char* psLongitude, char* psAltitude);

Example fragments:
As TzDb zone.tab:
`Czt+404251g-0740023`
As NMEA 0183 GPGGA:
`Cet+4042.85000g-00740.38333a123.45`

Examples:
As TzDb zone.tab:
`D2024-11-03T01:59:59U-04Zamerica/new_yorkV2024aMuCzt+404251g-0740023X`
As NMEA 0183 GPGGA:
`D2024-11-03T01:59:59U-04Zamerica/new_yorkV2024aMuCet+4042.85000g-00740.38333a123.45X`

Annex A - CCT Standard Rates

CCT standard rates are enumerated in CCTRateLib, CRateTable.h

```
typedef enum CBFRate_et
{
 CLOCK_UNKNOWN = 0,
 CLOCK_0  , // 1/1 second
 CLOCK_1  , // 1/10 tenths of second
 CLOCK_2  , // 1/100 hundredths of second
 CLOCK_3  , // 1/1000 millisecond
 CLOCK_4  , // 1/10000 10ths of millisecond, 100 microsecond
 CLOCK_5  , // 1/100000 100ths of millisecond, 1000 microsecond
 CLOCK_6  , // 1/1000000 microsecond
 CLOCK_7  , // 1/10000000 10ths of microsecond, 100 nanosecond
 CLOCK_8  , // 1/100000000 100ths of microsecond, 1000 nanoseconds
 CLOCK_9  , // 1/1000000000 nanosecond
 CLOCK_10 , // 1/10000000000 10ths of nanosecond, 100 picoseconds
 CLOCK_11 , // 1/100000000000 100ths of nanosecond, 1000 picoseconds
 CLOCK_12 , // 1/1000000000000 picosecond
// CLOCK_15 , // 1/1000000000000000 femtosecond
// CLOCK_18 , // 1/1000000000000000000 attosecond
// CLOCK_21 , // 1/10000000000000000000000 zeptosecond
// CLOCK_24 , // 1/100000000000000000000000000 yoctosecond
// CLOCK_44 , // 1/10tothe44th Planck time
// room for more CLOCK_
 VID_UNKNOWN = 20,
 VID_1000_12,
 VID_1000_12_5,
 VID_1000_15,
 VID_1000_16,
 VID_1000_18,
 VID_1000_24,
 VID_1000_25,
 VID_1000_30,
 VID_1000_32,
 VID_1000_36,
 VID_1000_48,
 VID_1000_50,
 VID_1000_60,
 VID_1000_64,
 VID_1000_72,
 VID_1000_96,
 VID_1000_100,
 VID_1000_120,
 VID_1000_128,
 VID_1000_144,
 VID_1000_192,
 VID_1000_200,
 VID_1000_240,
 VID_1000_256,
 VID_1000_288,
 VID_1001_12 = 62,
 VID_1001_12_5,
 VID_1001_15,
 VID_1001_16,
 VID_1001_18,
 VID_1001_24,
 VID_1001_25,
 VID_1001_30,
 VID_1001_32,
 VID_1001_36,
 VID_1001_48,
 VID_1001_50,
 VID_1001_60,
 VID_1001_64,
 VID_1001_72,
```

```
  VID_1001_96,
  VID_1001_100,
  VID_1001_120,
  VID_1001_128,
  VID_1001_144,
  VID_1001_192,
  VID_1001_200,
  VID_1001_240,
  VID_1001_256,
  VID_1001_288,
  AUD_UNKNOWN = 103,
  AUD_30720,
  AUD_31968,
  AUD_32000,
  AUD_32032,
  AUD_33333,
  AUD_42336,
  AUD_44055,
  AUD_44100,
  AUD_44144,
  AUD_45937,
  AUD_46080,
  AUD_47952,
  AUD_48000,
  AUD_48048,
  AUD_50000,
  AUD_61440,
  AUD_63936,
  AUD_64000,
  AUD_64064,
  AUD_66666,
  AUD_84672,
  AUD_88111,
  AUD_88200,
  AUD_88288,
  AUD_91875,
  AUD_92160,
  AUD_95904,
  AUD_96000,
  AUD_96096,
  AUD_100000,
  AUD_122880,
  AUD_127872,
  AUD_128000,
  AUD_128128,
  AUD_133333,
  AUD_169344,
  AUD_176223,
  AUD_176400,
  AUD_176576,
  AUD_183750,
  AUD_184320,
  AUD_191808,
  AUD_192000,
  AUD_192192,
  AUD_200000
} CBFRate_et;
```

Annex B - CCFE Character Set

CCFE shall be encoded using the ASCII character set excluding control characters and white space.

Disallowed Characters

Character	Dec	Hex	Name
control characters	0 to 31	(0x00) to (0x1F)	control characters
space	32	(0x20)	space
delete	127	(0x7F)	DEL
128 or higher	128 to 255	(0x80) to (0xFF)	128 or higher

Active Characters

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
!"#$%&'()*+,-./:;<=>?@[]^_`{|}~
```

Character	Dec	Hex	Name
!	33	(0x21)	Exclamation
"	34	(0x22h)	Quote
#	35	(0x23)	Number
$	36	(0x24)	Dollar
%	37	(0x25)	Percent
&	38	(0x26)	Ampersand
'	39	(0x27)	Apostrophe
(	40	(0x28)	Open Parenthesis
)	41	(0x29)	Close Parenthesis
*	42	(0x2A)	Asterisk
+	43	(0x2B)	Plus
,	44	(0x2C)	Comma
-	45	(0x2D)	Hyphen
.	46	(0x2E)	Period
/	47	(0x2F)	Forward Slash
0	48	(0x30)	
1	49	(0x31)	
2	50	(0x32)	
3	51	(0x33)	
4	52	(0x34)	
5	53	(0x35)	
6	54	(0x36)	
7	55	(0x37)	
8	56	(0x38)	
9	57	(0x39)	
:	58	(0x3A)	Colon
;	59	(0x3B)	Semicolon
<	60	(0x3C)	Less than
=	61	(0x3D)	Equal
>	62	(0x3E)	Greater than

?	63	(0x3F)	Question
@	64	(0x40)	at symbol
A	65	(0x41)	
B	66	(0x42)	
C	67	(0x43)	
D	68	(0x44)	
E	69	(0x45)	
F	70	(0x46)	
G	71	(0x47)	
H	72	(0x48)	
I	73	(0x49)	
J	74	(0x4A)	
K	75	(0x4B)	
L	76	(0x4C)	
M	77	(0x4D)	
N	78	(0x4E)	
O	79	(0x4F)	
P	80	(0x50)	
Q	81	(0x51)	
R	82	(0x52)	
S	83	(0x53)	
T	84	(0x54)	
U	85	(0x55)	
V	86	(0x56)	
W	87	(0x57)	
X	88	(0x58)	
Y	89	(0x59)	
Z	90	(0x5A)	
[	91	(0x5B)	Open Bracket
\	92	(0x5C)	Back Slash
]	93	(0x5D)	Close Bracket
^	94	(0x5E)	Caret
_	95	(0x5F)	Underscore
`	96	(0x60)	Grave Accent
a	97	(0x61)	
b	98	(0x62)	
c	99	(0x63)	
d	100	(0x64)	
e	101	(0x65)	
f	102	(0x66)	
g	103	(0x67)	
h	104	(0x68)	
i	105	(0x69)	
j	106	(0x6A)	
k	107	(0x6B)	
l	108	(0x6C)	
m	109	(0x6D)	
n	110	(0x6E)	
o	111	(0x6F)	

p	112	(0x70)	
q	113	(0x71)	
r	114	(0x72)	
s	115	(0x73)	
t	116	(0x74)	
u	117	(0x75)	
v	118	(0x76)	
w	119	(0x77)	
x	120	(0x78)	
y	121	(0x79)	
z	122	(0x7A)	
{	123	(0x7B)	Open Brace
|	124	(0x7C)	Vertical Bar
}	125	(0x7D)	Close Brace
~	126	(0x7E)	Tilde

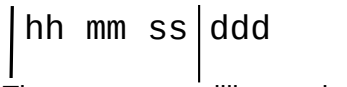
Annex C - CCFE Example Illustrations

Time point < 86400s in seconds

T01:00:00X

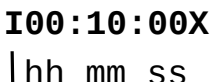
Time point < 86400s in milliseconds

T01:00:00m999X



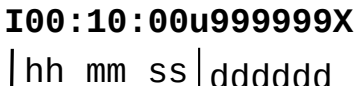
Interval < 86400s in seconds

I00:10:00X



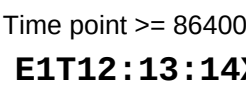
Interval < 86400s in microseconds

I00:10:00u999999X



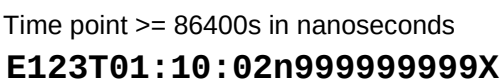
Time point >= 86400s in seconds

E1T12:13:14X



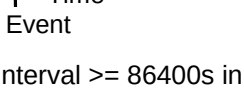
Time point >= 86400s in nanoseconds

E123T01:10:02n999999999X



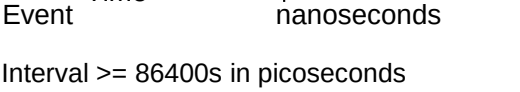
Interval >= 86400s in seconds

P2T22:23:24X



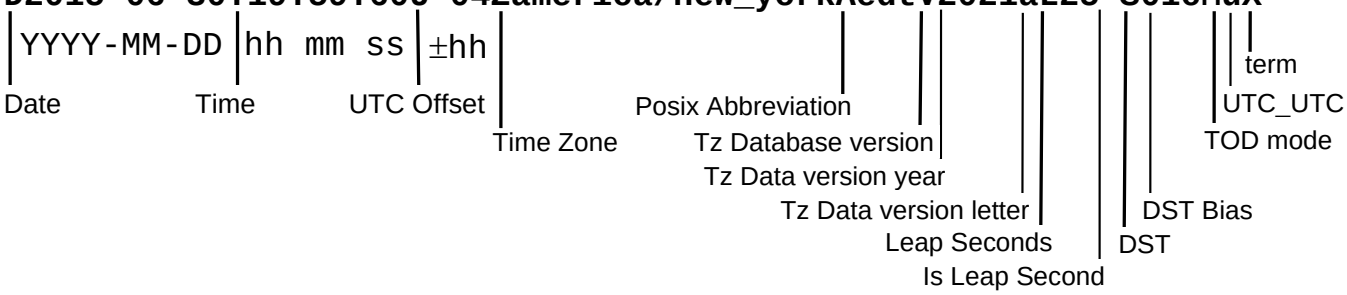
Interval >= 86400s in picoseconds

P12T01:10:02p999999999999X



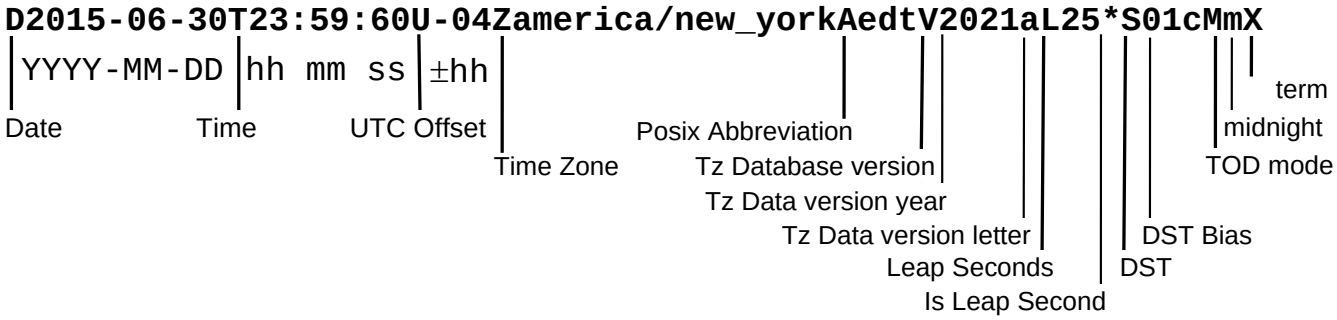
2015 Leap-second in New York with TOD_LEAPSECOND_UTC_UTC Count Mode

D2015-06-30T19:59:60U-04Zamerica/new_yorkAedtV2021aL25*S01cMuX



2015 Leap-second in New York with TOD_LEAPSECOND_MIDNIGHT Count Mode in seconds

D2015-06-30T23:59:60U-04Zamerica/new_yorkAedtV2021aL25*S01cMmX



Annex D - Example Listing from CCT Reference Implementation

```
CCT Version 3.0.0.0 2024-12-02 00:00:00

Time flies when you're having fun!


File out: ..\OutputFiles\CCTOut_BB_TESTSELECTEDCONFIGURATIONSANDSHOWCBFVALUES.txt


======= TestSelectedConfigurationsAndShowCbfValues() =======

------------------------ Common Calendar Character Format (CCF) ----------------
Date       Time     UTC Offset
|          |        |   Zone (Tz time zone name)
|          |        |   |                Abbreviation (Posix name)
|          |        |   |                |   Version (Tz Database release)
|          |        |   |                |   |     Leap-seconds
|          |        |   |                |   |     |   Saving (DST)
|          |        |   |                |   |     |   |DST bias
|          |        |   |                |   |     |   ||  Mode (TOD count mode)
|          |        |   |                |   |     |   ||  | X terminator
D2015-06-30T19:59:60U-04America/new_yorkAedtV2021aL25*S01cMuX
--------------------------------------------------------------------------------


--------------------------------------
CCF, CCbfE member values and CBF binary
--------------------------------------

-- Interval ((I)nterval) < 86400s --
I301:46:38m999X
CBFETime_st::
 m_eRateEnumeration CLOCK_3
 m_bLocalDateExt    FALSE
 m_b24HourPeriodExt  FALSE
 m_bCounterSign     positive
 m_eCounterSize     COUNTERSIZE_35
 m_ulCounterLow32   1086398999
CBFETime_st Counter  1086398999
CBF Total size      8 bytes  64 bits
CBF binary interchange bytes as hexadecimal
04 00 17 22 -c1 40 35 00  size 8

-- Interval ((P)eriod) >= 86400s --
P1I00:00:00m000X
CBFETime_st::
 m_eRateEnumeration CLOCK_3
 m_bLocalDateExt    FALSE
 m_b24HourPeriodExt  TRUE
 m_bCounterSign     positive
 m_eCounterSize     COUNTERSIZE_35
 m_ulCounterLow32   0
CBFETime_st Counter  0
CBFE24HourPeriod_st::
 m_unDayDuration    1
CBF Total size      12 bytes  96 bits
CBF binary interchange bytes as hexadecimal
04 02 00 00 00 00 35 00 01 00 00 00  size 12

-- Time point ((T)ime) < 86400s --
T301:46:38m999X
CBFETime_st::
 m_eRateEnumeration CLOCK_3
 m_bLocalDateExt    FALSE
 m_b24HourPeriodExt  FALSE
 m_bCounterSign     positive
```

```
 m_eCounterSize     COUNTERSIZE_35
 m_ulCounterLow32   1086398999
CBFETime_st Counter  1086398999
CBF Total size      8 bytes  64 bits
CBF binary interchange bytes as hexadecimal
04 00 17 22 -c1 40 34 00  size 8

-- Time point ((E)vent) >= 86400s --
E1T00:00:00m000X
CBFETime_st::
 m_eRateEnumeration CLOCK_3
 m_bLocalDateExt    FALSE
 m_b24HourPeriodExt  TRUE
 m_bCounterSign     positive
 m_eCounterSize     COUNTERSIZE_35
 m_ulCounterLow32   0
CBFETime_st Counter  0
CBFE24HourPeriod_st::
 m_unDayDuration    1
CBF Total size      12 bytes  96 bits
CBF binary interchange bytes as hexadecimal
04 02 00 00 00 00 34 00  size 8

-- Second preceding 1972 leap-second in UTC time zone --
D1972-06-30T23:59:59.999U+00Zetc/utcAutcV2024aL00+MuX
CBFETime_st::
 m_eRateEnumeration CLOCK_3
 m_bLocalDateExt    TRUE
 m_b24HourPeriodExt  FALSE
 m_bCounterSign     positive
 m_eCounterSize     COUNTERSIZE_35
 m_ulCounterLow32   86399999
CBFETime_st Counter  86399999
CBFELocalDate_st::
 m_l1970DayNumber    911
 m_nLeapsecs         0
 m_TZDTimeZoneID_st.m_unZoneIdx    idx[125] Etc/UTC
 m_TZDTimeZoneID_st.m_unTzDataReleaseYear   52
 m_TZDTimeZoneID_st.m_unTzDataReleaseLetter 0
 m_TZDTimeZoneID_st.m_bCBFLocationExt       FALSE
 m_TZDTimeZoneID_st.m_bCBFAbbrExt           TRUE
 m_TZDTimeZoneID_st.m_bCBFAbbrChangeExt     FALSE
 m_lUTCOffset         0
 m_eLsTodCntMode           TOD_LEAPSECOND_UTC_UTC
 m_eDstMode      DSTCOUNTMODE_NOTAPPLICABLE
 m_bIsLeapSecondDay   TRUE
 m_bIsLeapSecond      FALSE
 m_bIsLeapSecondNegative   FALSE
 m_bUtcShiftExt       FALSE
 m_bDstBiasExt            FALSE
 m_bDstTransDayExt    FALSE
CBFAbbr::
 utc
CBF Total size      27 bytes  216 bits
CBF binary interchange bytes as hexadecimal
04 01 -ff 5b 26 05 00 00 -8f 03 00 00 00 00 00 7d 00 34 10 00 00 04 00 02 -f5 -f4 63
size 27

-- 1972 leap-second in UTC time zone --
D1972-06-30T23:59:60.999U+00Zetc/utcAutcV2024aL00*MuX
CBFETime_st::
 m_eRateEnumeration CLOCK_3
 m_bLocalDateExt    TRUE
 m_b24HourPeriodExt  FALSE
 m_bCounterSign     positive
 m_eCounterSize     COUNTERSIZE_35
 m_ulCounterLow32   86400999
```

```
CBFETime_st Counter  86400999
CBFELocalDate_st::
 m_l1970DayNumber    911
 m_nLeapsecs         0
 m_TZDTimeZoneID_st.m_unZoneIdx    idx[125] Etc/UTC
 m_TZDTimeZoneID_st.m_unTzDataReleaseYear   52
 m_TZDTimeZoneID_st.m_unTzDataReleaseLetter 0
 m_TZDTimeZoneID_st.m_bCBFLocationExt       FALSE
 m_TZDTimeZoneID_st.m_bCBFAbbrExt           TRUE
 m_TZDTimeZoneID_st.m_bCBFAbbrChangeExt     FALSE
 m_lUTCOffset         0
 m_eLsTodCntMode           TOD_LEAPSECOND_UTC_UTC
 m_eDstMode      DSTCOUNTMODE_NOTAPPLICABLE
 m_bIsLeapSecondDay   TRUE
 m_bIsLeapSecond      TRUE
 m_bIsLeapSecondNegative   FALSE
 m_bUtcShiftExt       FALSE
 m_bDstBiasExt            FALSE
 m_bDstTransDayExt    FALSE
CBFAbbr::
 utc
CBF Total size      27 bytes  216 bits
CBF binary interchange bytes as hexadecimal
04 01 -e7 5f 26 05 00 00 -8f 03 00 00 00 00 00 7d 00 34 10 00 00 04 00 03 -f5 -f4 63
size 27

-- Second following 1972 leap-second in UTC time zone --
D1972-07-01T00:00:00.000U+00Zetc/utcAutcV2024aL01MuX
CBFETime_st::
 m_eRateEnumeration CLOCK_3
 m_bLocalDateExt    TRUE
 m_b24HourPeriodExt  FALSE
 m_bCounterSign     positive
 m_eCounterSize     COUNTERSIZE_35
 m_ulCounterLow32   0
CBFETime_st Counter  0
CBFELocalDate_st::
 m_l1970DayNumber    912
 m_nLeapsecs         1
 m_TZDTimeZoneID_st.m_unZoneIdx    idx[125] Etc/UTC
 m_TZDTimeZoneID_st.m_unTzDataReleaseYear   52
 m_TZDTimeZoneID_st.m_unTzDataReleaseLetter 0
 m_TZDTimeZoneID_st.m_bCBFLocationExt       FALSE
 m_TZDTimeZoneID_st.m_bCBFAbbrExt           TRUE
 m_TZDTimeZoneID_st.m_bCBFAbbrChangeExt     FALSE
 m_lUTCOffset         0
 m_eLsTodCntMode           TOD_LEAPSECOND_UTC_UTC
 m_eDstMode      DSTCOUNTMODE_NOTAPPLICABLE
 m_bIsLeapSecondDay   FALSE
 m_bIsLeapSecond      FALSE
 m_bIsLeapSecondNegative   FALSE
 m_bUtcShiftExt       FALSE
 m_bDstBiasExt            FALSE
 m_bDstTransDayExt    FALSE
CBFAbbr::
 utc
CBF Total size      27 bytes  216 bits
CBF binary interchange bytes as hexadecimal
04 01 00 00 00 00 00 00 -90 03 00 00 -80 00 00 7d 00 34 10 00 00 04 00 00 -f5 -f4 63
size 27

-- Second preceding 1972 leap-second in New York time zone --
D1972-06-30T23:19:59.999U-04Zamerica/new_yorkAedtV2024aL01+S01cMuX
CBFETime_st::
 m_eRateEnumeration CLOCK_3
 m_bLocalDateExt    TRUE
 m_b24HourPeriodExt  FALSE
```

```
 m_bCounterSign     positive
 m_eCounterSize     COUNTERSIZE_35
 m_ulCounterLow32   84000999
CBFETime_st Counter  84000999
CBFELocalDate_st::
 m_l1970DayNumber    911
 m_nLeapsecs         1
 m_TZDTimeZoneID_st.m_unZoneIdx    idx[230] America/New_York
 m_TZDTimeZoneID_st.m_unTzDataReleaseYear   52
 m_TZDTimeZoneID_st.m_unTzDataReleaseLetter 0
 m_TZDTimeZoneID_st.m_bCBFLocationExt       FALSE
 m_TZDTimeZoneID_st.m_bCBFAbbrExt           TRUE
 m_TZDTimeZoneID_st.m_bCBFAbbrChangeExt     FALSE
 m_lUTCOffset         -18000
 m_eLsTodCntMode           TOD_LEAPSECOND_UTC_UTC
 m_eDstMode      DSTCOUNTMODE_CONVENTIONAL
 m_bIsLeapSecondDay   TRUE
 m_bIsLeapSecond      FALSE
 m_bIsLeapSecondNegative   FALSE
 m_bUtcShiftExt       FALSE
 m_bDstBiasExt            TRUE
 m_bDstTransDayExt    FALSE
CBFAbbr::
 edt
CBFDstBias_st::
 m_nDstBias         3600
CBF Total size      29 bytes  232 bits
CBF binary interchange bytes as hexadecimal
04 01 -e7 -c0 01 05 00 00 -8f 03 00 00 -80 00 00 -e6 00 34 10 -b0 -b9 55 00 02 10 0e -
e5 -e4 74  size 29

-- 1972 leap-second in New York time zone --
D1972-06-30T19:59:60.999U-04Zamerica/new_yorkAedtV2024aL00*S01cMuX
CBFETime_st::
 m_eRateEnumeration CLOCK_3
 m_bLocalDateExt    TRUE
 m_b24HourPeriodExt  FALSE
 m_bCounterSign     positive
 m_eCounterSize     COUNTERSIZE_35
 m_ulCounterLow32   72000999
CBFETime_st Counter  72000999
CBFELocalDate_st::
 m_l1970DayNumber    911
 m_nLeapsecs         0
 m_TZDTimeZoneID_st.m_unZoneIdx    idx[230] America/New_York
 m_TZDTimeZoneID_st.m_unTzDataReleaseYear   52
 m_TZDTimeZoneID_st.m_unTzDataReleaseLetter 0
 m_TZDTimeZoneID_st.m_bCBFLocationExt       FALSE
 m_TZDTimeZoneID_st.m_bCBFAbbrExt           TRUE
 m_TZDTimeZoneID_st.m_bCBFAbbrChangeExt     FALSE
 m_lUTCOffset         -18000
 m_eLsTodCntMode           TOD_LEAPSECOND_UTC_UTC
 m_eDstMode      DSTCOUNTMODE_CONVENTIONAL
 m_bIsLeapSecondDay   TRUE
 m_bIsLeapSecond      TRUE
 m_bIsLeapSecondNegative   FALSE
 m_bUtcShiftExt       FALSE
 m_bDstBiasExt            TRUE
 m_bDstTransDayExt    FALSE
CBFAbbr::
 edt
CBFDstBias_st::
 m_nDstBias         3600
CBF Total size      29 bytes  232 bits
CBF binary interchange bytes as hexadecimal
04 01 -e7 -a5 4a 04 00 00 -8f 03 00 00 00 00 00 -e6 00 34 10 -b0 -b9 55 00 03 10 0e -
e5 -e4 74  size 29
```

```
-- Second following 1972 leap-second in New York time zone --
D1972-07-01T00:00:00.000U-04Zamerica/new_yorkAedtV2024aL01S01cMuX
CBFETime_st::
 m_eRateEnumeration CLOCK_3
 m_bLocalDateExt    TRUE
 m_b24HourPeriodExt  FALSE
 m_bCounterSign     positive
 m_eCounterSize     COUNTERSIZE_35
 m_ulCounterLow32   0
CBFETime_st Counter  0
CBFELocalDate_st::
 m_l1970DayNumber    912
 m_nLeapsecs         1
 m_TZDTimeZoneID_st.m_unZoneIdx    idx[230] America/New_York
 m_TZDTimeZoneID_st.m_unTzDataReleaseYear   52
 m_TZDTimeZoneID_st.m_unTzDataReleaseLetter 0
 m_TZDTimeZoneID_st.m_bCBFLocationExt       FALSE
 m_TZDTimeZoneID_st.m_bCBFAbbrExt           TRUE
 m_TZDTimeZoneID_st.m_bCBFAbbrChangeExt     FALSE
 m_lUTCOffset         -18000
 m_eLsTodCntMode           TOD_LEAPSECOND_UTC_UTC
 m_eDstMode      DSTCOUNTMODE_CONVENTIONAL
 m_bIsLeapSecondDay   FALSE
 m_bIsLeapSecond      FALSE
 m_bIsLeapSecondNegative   FALSE
 m_bUtcShiftExt       FALSE
 m_bDstBiasExt            TRUE
 m_bDstTransDayExt    FALSE
CBFAbbr::
 edt
CBFDstBias_st::
 m_nDstBias         3600
CBF Total size      29 bytes  232 bits
CBF binary interchange bytes as hexadecimal
04 01 00 00 00 00 00 00 -90 03 00 00 -80 00 00 -e6 00 34 10 -b0 -b9 55 00 00 10 0e -e5
-e4 74  size 29

-- Nanosecond preceding 2016 DST Onset in New York time zone --
D2016-03-13T01:59:59.999999999U-05Zamerica/new_yorkAestV2024aL26S00t01a02cMuX
CBFETime_st::
 m_eRateEnumeration CLOCK_9
 m_bLocalDateExt    TRUE
 m_b24HourPeriodExt  FALSE
 m_bCounterSign     positive
 m_eCounterSize     COUNTERSIZE_48
 m_ulCounterLow32   1634811903
CBFECounterHigh16_st::
 m_unCounterHigh16  1676
CBFETime_st Counter  7199999999999
CBFELocalDate_st::
 m_l1970DayNumber    16873
 m_nLeapsecs         26
 m_TZDTimeZoneID_st.m_unZoneIdx    idx[230] America/New_York
 m_TZDTimeZoneID_st.m_unTzDataReleaseYear   52
 m_TZDTimeZoneID_st.m_unTzDataReleaseLetter 0
 m_TZDTimeZoneID_st.m_bCBFLocationExt       FALSE
 m_TZDTimeZoneID_st.m_bCBFAbbrExt           TRUE
 m_TZDTimeZoneID_st.m_bCBFAbbrChangeExt     FALSE
 m_lUTCOffset         -18000
 m_eLsTodCntMode           TOD_LEAPSECOND_UTC_UTC
 m_eDstMode      DSTCOUNTMODE_CONVENTIONAL
 m_bIsLeapSecondDay   FALSE
 m_bIsLeapSecond      FALSE
 m_bIsLeapSecondNegative   FALSE
 m_bUtcShiftExt       FALSE
 m_bDstBiasExt            TRUE
```

```
 m_bDstTransDayExt    TRUE
CBFAbbr::
 est
CBFDstBias_st::
 m_nDstBias         0
CBFDstTransDay_st::
 m_ulDSTTransTime   7200
 m_lDSTBiasChange   3600
CBF Total size      31 bytes  248 bits
CBF binary interchange bytes as hexadecimal
0a 05 -ff 3f 71 61 00 00 -8c 06 -e9 41 00 00 00 0d 00 -e6 00 34 10 -b0 -b9 -d5 00 00
00 00 20 1c 10 0e 00 -e5 -f3 74  size 36

-- Nanosecond at 2016 DST Onset in New York time zone with Location--
D2016-03-13T03:00:00.000000000U-
04Zamerica/new_yorkAedtV2024aL26S01t01a02cMuCzt+4042+51g-07400+23X
CBFETime_st::
 m_eRateEnumeration CLOCK_9
 m_bLocalDateExt    TRUE
 m_b24HourPeriodExt  FALSE
 m_bCounterSign     positive
 m_eCounterSize     COUNTERSIZE_48
 m_ulCounterLow32   2452217856
CBFECounterHigh16_st::
 m_unCounterHigh16  2514
CBFETime_st Counter  10800000000000
CBFELocalDate_st::
 m_l1970DayNumber    16873
 m_nLeapsecs         26
 m_TZDTimeZoneID_st.m_unZoneIdx    idx[230] America/New_York
 m_TZDTimeZoneID_st.m_unTzDataReleaseYear   52
 m_TZDTimeZoneID_st.m_unTzDataReleaseLetter 0
 m_TZDTimeZoneID_st.m_bCBFLocationExt       TRUE
 m_TZDTimeZoneID_st.m_bCBFAbbrExt           TRUE
 m_TZDTimeZoneID_st.m_bCBFAbbrChangeExt     FALSE
 m_lUTCOffset         -18000
 m_eLsTodCntMode           TOD_LEAPSECOND_UTC_UTC
 m_eDstMode      DSTCOUNTMODE_CONVENTIONAL
 m_bIsLeapSecondDay   FALSE
 m_bIsLeapSecond      FALSE
 m_bIsLeapSecondNegative   FALSE
 m_bUtcShiftExt       FALSE
 m_bDstBiasExt            TRUE
 m_bDstTransDayExt    TRUE
CBFAbbr::
 edt
CBFDstBias_st::
 m_nDstBias         3600
CBFDstTransDay_st::
 m_ulDSTTransTime   7200
 m_lDSTBiasChange   3600
CBFLocation_st::
m_i9Lat_Deg       40
m_i7Lat_Min       42
m_i21Lat_uMin     51
m_i9Lgn_Deg       -74
m_i7Lgn_Min       0
m_i21Lng_uMin     23
m_iAlt_cm         16777215
m_bSourceIsExtern 0
m_bIsValidLat     1
m_bIsValidLng     1
m_bIsValidAlt     0
CBF Total size      45 bytes  360 bits
CBF binary interchange bytes as hexadecimal
0a 05 00 -e0 29 -92 00 00 -d2 09 -e9 41 00 00 00 0d 00 -e6 -80 34 10 -b0 -b9 -d5 00 00
10 0e 20 1c 10 0e 00 -e5 -e4 74 33 00 00 05 17 00 -c0 36 -ff -ff -ff 00 2a 03  size 50
```

```
-- Nanosecond following 2016 DST Onset in New York time zone with Location--
D2016-03-13T03:00:00.000000001U-
04Zamerica/new_yorkAedtV2024aL26S01t01a02cMuCzt+4042+51g-07400+23X
CBFETime_st::
 m_eRateEnumeration CLOCK_9
 m_bLocalDateExt    TRUE
 m_b24HourPeriodExt  FALSE
 m_bCounterSign     positive
 m_eCounterSize     COUNTERSIZE_48
 m_ulCounterLow32   2452217857
CBFECounterHigh16_st::
 m_unCounterHigh16  2514
CBFETime_st Counter  10800000000001
CBFELocalDate_st::
 m_l1970DayNumber    16873
 m_nLeapsecs         26
 m_TZDTimeZoneID_st.m_unZoneIdx    idx[230] America/New_York
 m_TZDTimeZoneID_st.m_unTzDataReleaseYear   52
 m_TZDTimeZoneID_st.m_unTzDataReleaseLetter 0
 m_TZDTimeZoneID_st.m_bCBFLocationExt       TRUE
 m_TZDTimeZoneID_st.m_bCBFAbbrExt           TRUE
 m_TZDTimeZoneID_st.m_bCBFAbbrChangeExt     FALSE
 m_lUTCOffset         -18000
 m_eLsTodCntMode           TOD_LEAPSECOND_UTC_UTC
 m_eDstMode      DSTCOUNTMODE_CONVENTIONAL
 m_bIsLeapSecondDay   FALSE
 m_bIsLeapSecond      FALSE
 m_bIsLeapSecondNegative   FALSE
 m_bUtcShiftExt       FALSE
 m_bDstBiasExt            TRUE
 m_bDstTransDayExt    TRUE
CBFAbbr::
 edt
CBFDstBias_st::
 m_nDstBias         3600
CBFDstTransDay_st::
 m_ulDSTTransTime   7200
 m_lDSTBiasChange   3600
CBFLocation_st::
m_i9Lat_Deg       40
m_i7Lat_Min       42
m_i21Lat_uMin     51
m_i9Lgn_Deg       -74
m_i7Lgn_Min       0
m_i21Lng_uMin     23
m_iAlt_cm         16777215
m_bSourceIsExtern 0
m_bIsValidLat     1
m_bIsValidLng     1
m_bIsValidAlt     0
CBF Total size      45 bytes  360 bits
CBF binary interchange bytes as hexadecimal
0a 05 01 -e0 29 -92 00 00 -d2 09 -e9 41 00 00 00 0d 00 -e6 -80 34 10 -b0 -b9 -d5 00 00
10 0e 20 1c 10 0e 00 -e5 -e4 74 33 00 00 05 17 00 -c0 36 -ff -ff -ff 00 2a 03  size 50


Your time is up.
```