

---

# Common Calendar Media Timestamp

## Common Calendar Timestamp System

Brooks Harris Version 1 2025-02-02

*The author dedicates this work to the public domain*

---

### Table of Contents

<b>1</b>	<b>Introduction</b> .....	<b>2</b>
<b>2</b>	<b>Scope</b> .....	<b>4</b>
<b>3</b>	<b>Normative References</b> .....	<b>4</b>
<b>4</b>	<b>Common Calendar Media Binary Format (CBFM)</b> .....	<b>5</b>
4.1	CBFM Components .....	5
4.1.1	Time .....	5
4.1.1.1	Rate enumeration.....	6
4.1.1.2	Counter Size enumeration .....	6
4.1.2	Counter Size Extensions .....	7
4.1.3	24 Hour Period.....	8
4.1.4	Local Date.....	9
4.1.4.1	Time Zone Identity .....	10
4.1.4.2	Time-of-Day Count Mode.....	11
4.2	CBFM Construction .....	11
4.2.1	As Time-point:.....	11
4.2.1.1	As UTC accurate local time-point timestamp : .....	12
4.2.1.2	As UTC accurate local date with time portion having no relation to the date: .....	12
4.2.1.3	As time-point timestamp < 86400 seconds:.....	12
4.2.1.4	As time-point timestamp >= 86400 seconds:.....	12
4.2.2	As Time Interval .....	12
4.2.2.1	As time interval < 86400 seconds: .....	13
4.2.2.2	As time interval >= 86400 seconds:.....	13
4.2.3	Assembly Order .....	13
4.3	Geostamp .....	14
4.3.1	Geographic Coordinates.....	15
4.4	CCTE RIFF Wrapper .....	15
<b>5</b>	<b>Common Calendar Media Character Format (CCFM)</b> .....	<b>15</b>
5.1	ISO 8601 Variation .....	16
5.2	Character Set.....	16
5.3	Hard Terminator.....	16
5.4	Data Field Elements .....	16
5.4.1	Time Element.....	17
5.4.1.1	UTC accurate local date and time-of-day .....	18
5.4.1.2	UTC accurate local date and time having no relation to date .....	18
5.4.1.3	Time point less than 86400s .....	18
5.4.1.4	Time point equal or greater than to 86400s.....	18
5.4.1.5	Interval less than 86400s .....	19
5.4.1.6	Interval equal or greater than 86400s .....	19
5.4.2	Event Element .....	19
5.4.3	Period Element .....	20
5.4.4	Date Element .....	20
5.4.5	TOD Count Mode Element .....	22
5.4.6	UTC Offset Element.....	23
5.4.7	Time Zone Element .....	23
5.4.8	IANA Time Zone Database Version Element .....	24
5.5	Assembly and Order .....	24
5.5.1	UTC accurate local date and time-of-day .....	24
5.5.2	UTC accurate local date and time with no relation to the date.....	25
5.5.3	Time point less than 86400s.....	26
5.5.4	Time point equal or greater than 86400s .....	26
5.5.5	Interval less than 86400s.....	26
5.5.6	Interval equal or greater than 86400s.....	26

5.6	Geostamp .....	27
5.6.1	Location Element .....	27
<b>Annex A - CCT Standard Rates .....</b>		<b>29</b>
<b>Annex B - CCFM Character Set .....</b>		<b>31</b>
<b>Annex C - CCFM Example Illustrations.....</b>		<b>34</b>
<b>Annex D - Example Listing from CCT Reference Implementation .....</b>		<b>35</b>

**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**

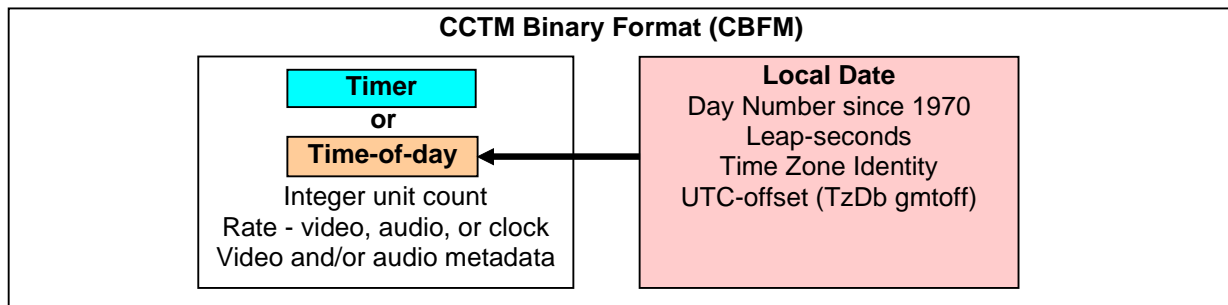
The Common Calendar Media timestamp includes support for video and audio media labeling. It also supports second and decimal-fraction resolutions (clock).

Common Calendar Media Binary Format (CBFM) provides a compact binary data format to facilitate fast transfer, efficient interchange, and economical storage. The CBFM 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.

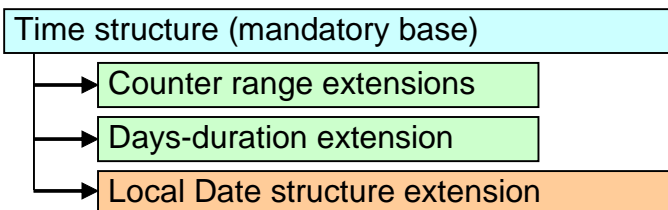
CBFM acts in concert with Common Calendar Media Character Format (CCFM) to provide comprehensive description of local date and time with symmetrical conversion between the binary and character based YMDhms representations.

**Common Calendar Media Binary Format (CBFM)**

The CBFM binary timestamp is a variable length compound counter with associated metadata. It is made up of the mandatory Time structure base that provides the anchor for any configuration of CBFM, 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:

$$\text{seconds-since-UTC1970} = (\text{DayNumber} * 86400) + (\text{leap-seconds}) + (\text{seconds-since-midnight})$$

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 Media 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 Media Binary Format (CBFM) 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 CBFM to make clear the connections between the CBFM data and metadata. The corresponding CCFE character representations, and the details of conversion algorithms required.

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

### CCTE Character Format (CCFE)

```
D2024-03-10T03:00:00U-06Zamerica/denverAmdtV2024aL27S01t01a02cMuX
```

The CCT reference implementation implements CCFE and CBFM in the CCTELib library. CBFM is implemented as class CCbfE and the CBFM data types are defined in CBFM.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/CBFM.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 Media Binary Format (CBFM)

Common Calendar Media Binary Format (CBFM) 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 Media Character Format (CCFE).

A CBFM 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 CBFM 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 CBFM 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 CBFM 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 CBFM Components

A CBFM is constructed from the mandatory Time data structure CBFMTime\_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 CBFM Construction and Assembly.

#### 4.1.1 Time

The CBFMTime\_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 CBFMTime\_st 35-bit counter may be extended to 48, 64, or 80 bits by addition of the optional CBFMCounterHigh16\_st and/or CBFMCounterHigh32\_st extensions to accommodate 24 hours at rates higher than milliseconds.

The presence of one or both of the range extensions is indicated by CBFMTime\_st::m\_eCounterSize holding a CBFMCounterSize\_et enumeration. See CBFMCounterSize\_et. The extensions are treated as the high-words and the CBFMTime\_st::m\_ulCounterLow32 counter as the low-word.

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

```
typedef struct CBFMTime_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;
} CBFMTime_st;

```

#### 4.1.1.1 Rate enumeration

CBFM 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 CBFMTime\_st 36-bit counter may be extended to 48, 64, or 80 bits by addition of the optional CBFMCounterHigh16\_st and/or CBFMCounterHigh32\_st extensions to accommodate 24 hours at rates higher than milliseconds.

The presence of one or both of the range extensions is indicated by CBFMTime\_st::m\_eCounterSize holding a CBFMCounterSize\_et enumeration. See CBFMCounterSize\_et. The extensions are treated as the high-words and the CBFMTime\_st 32-bit counter as the low-word.

```

typedef enum CBFMCounterSize_et // m_eCounterSize:2
{
COUNTERSIZE_32 = 0, // 32-bit counter contained in base object
COUNTERSIZE_48, // 48-bit counter
// 16-bit counter extension present,
// CBFMTime_st::m_lCounterLow32,
COUNTERSIZE_64, // 64-bit counter
// 32-bit counter extension present,

```

```

// TBOCounter32_st::m_ulCounterHigh32 and
// CBFMTIME_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
// CBFMTIME_st::m_lCounterLow32,
} CBFMCounterSize_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
CBFMRate_et			CBFMCounterSize_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  $\leq$  CLOCK\_3 (milliseconds) the CBFMTIME\_st::m\_ulCounterLow32 forms the entire counter and no counter size extension is required..

If rate is CLOCK\_6 (microseconds) or CLOCK\_9 (nanoseconds) CBFMCounter16\_st shall be added and CBFMCounter16\_st::m\_CounterHigh16 is treated as high word and CBFMTIME\_st::m\_ulCounterLow32 as the low word of the compound 48-bit counter.

If rate is CLOCK\_12 (picoseconds) CBFMCounter16\_st and CBFMCounter32\_st shall be added. The CBFMCounter32\_st::m\_CounterHigh32 is treated as high word, CBFMCounter16\_st::m\_CounterHigh16 as a "mid word", and CBFMTIME\_st::m\_ulCounterLow32 as the low word of the compound 80-bit counter.

See Cbf::SetTimeCounter(..) for bit-shift examples from each CBFM counter size to uint64\_t (unsigned 64-bit integer type).

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

#### 4.1.2 Counter Size Extensions

The CBFMTIME\_st::m\_ulCounterLow32 counter can be extended to 48-bits by adding CBFMCounter16\_st::m\_CounterHigh16, or 64-bits by adding CBFMCounter32\_st::m\_CounterHigh32, or 80-bits by adding both. The presence of these extensions are indicated by CBFMTIME\_st::m\_eCounterSize enumeration. See CBFMCounterSize\_et.

If rate is  $\leq$  CLOCK\_3 (milliseconds) the CBFMTIME\_st::m\_ulCounterLow32 forms the entire counter.

If rate is CLOCK\_6 (microseconds) or CLOCK\_9 (nanoseconds) CBFMCounter16\_st is added and CBFMCounter16\_st::m\_CounterHigh16 is treated as high word and CBFMTime\_st::m\_ulCounterLow32 as the low word of the compound 48-bit counter.

If rate is CLOCK\_12 (picoseconds) CBFMCounter16\_st and CBFMCounter32\_st are added and CBFMCounter32\_st::m\_CounterHigh32 is treated as high word, CBFMCounter16\_st::m\_CounterHigh16 as a "mid word", and CBFMTime\_st::m\_ulCounterLow32 as the low word of the compound 80-bit counter.

See CCBf::SetTimeCounter(..) for bit-shift examples from each CBFM counter size to uint64\_t (unsigned 64-bit integer type).

See CCBf::GetTimeSeconds(..) for bit-shift examples to set CBFM 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 CBFMCounterHigh16_st // 16 bit counter extension
{
    unsigned short m_unCounterHigh16;
} CBFMCounterHigh16_st;

typedef struct CBFMCounterHigh32_st // 32 bit counter extension
{
    unsigned long m_ulCounterHigh32;
} CBFMCounterHigh32_st;
```

### 4.1.3 24 Hour Period

CBFM24HourPeriod\_st extension is used to increase the range of the CBFMTime\_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 CBFM24HourPeriod\_st extension is present the CBFMTime\_st counter value shall be interpreted as a zero-base 86400 second period within the CBFM24HourPeriod\_st count.

The CBFM24HourPeriod\_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.

CBFM24HourPeriod\_st can be used to extend the range of a time-point or interval.

If CBFMTime\_st::m\_blsInterval == false the CBFM represents a time-point and the CBFM24HourPeriod\_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 CBFMTime\_st::m\_blsInterval == true the CBFM represents an interval and the CBFM24HourPeriod\_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 CBFMLocalDate\_st is prohibited if CBFMTime\_st::m\_blsInterval == 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 CBFM date-time local representations and calculate the duration between those two time points. A CBFM can be used to represent that difference.

```
                                     // 32 bit 24 hour (86400
typedef struct CBFM24HourPeriod_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;
} CBFM24HourPeriod_st;

```

#### 4.1.4 Local Date

The optional CBFMLocalDate\_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 CBFMDstBias\_st extension adds Daylight Saving bias if applicable. The CBFMDstTransDay\_st signals a DST change during the day if applicable. The CBFMUtcShift\_st extension adds support of UTC-offset shifts if applicable. The presence of CBFMLocalDate\_st is indicated by CBFMTime\_st::m\_bLocalDateExt.

If the CBFMLocalDate\_st extension is present the CBFMTime\_st counter value represents time-of-day (24 hours plus one leap-second if applicable) on the calendar date indicated by the values of CBFMLocalDate\_st. DST information shall be provided by the CBFMDstBias\_st and CBFMDstTransDay\_st extensions if applicable.

The values and metadata of CBFMTime\_st, CBFMLocalDate\_st and CBFMDstBias\_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 CBFM 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 IANA 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, CBFMLocalDate\_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 CBFMDstBias\_st structure shall be appended to provide required DST metadata if applicable. See CBFMLocalDate\_st::m\_bDSTExt and CBFMDstBias\_st.

The optional CBFMDstTransDay\_st structure shall be appended to provide required DST changes if applicable. See CBFMLocalDate\_st::m\_bDstTransDayExt and CBFMDstTransDay.

The optional CBFMUtcShift\_st structure shall be appended to provide required UTC-Offset metadata if applicable. See CBFMLocalDate\_st::m\_bUtcShiftExt and CBFMUtcShift.

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

The presence of a CBFMLocalDate\_st is indicated by

```
CBFMTime_st::m_bLocalDateExt:1
```

```
typedef struct CBFMLocalDate_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 database 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;
} CBFMLocalDate_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 database 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 Time-of-Day Count Mode

CCTM provides means to construct the YMDhms counting sequence through Common Calendar Character Format (CCFM) 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).

CBFTodCntMode\_et instructs how the CBFM binary representation is to be converted to the CCFM YMDhms counting sequence representation.

See Common Calendar Local Timescales.

```
typedef enum CBFTodCntMode_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)
} CBFTodCntMode_et;
```

## 4.2 CBFM Construction

CBFMTime\_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 CBFMTime\_st::m\_blsInterval flag indicates if the CBFM represents a time-point (m\_blsInterval == false) or a time interval (m\_blsInterval == true).

### 4.2.1 As Time-point:

If m\_blsInterval == false a CBFMTime\_st represents a time-point.

Used without the optional CBFMLocalDate\_st extension, the values of a CBFMTime\_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 CBFMLocalDate\_st extension the CBFM represents a leap-second accurate local date and time. The CBFMTime\_st counter value represents time-of-day (24 hours plus one leap-second if applicable) on the calendar date indicated by the CBFMLocalDate\_st member values. The presence of CBFMLocalDate\_st is indicated by the CBFMTime\_st::m\_bLocalDateExt flag.

If DST is observed in the time zone, the CBFMDst\_st extension shall be present, providing the necessary DST information. The presence of CBFMDst\_st is indicated by the CBFMLocalDate\_st::m\_bDSTExt flag.

#### 4.2.1.1 As UTC accurate local time-point timestamp :

CBFMTime\_st – mandatory

CBFMTime\_st::m\_blsInterval == false

CBFMCounterHigh16\_st and/or CBFMCounterHigh32\_st optional

CBFMLocalDate\_st – local date extension

CBFMLocalDate\_st::m\_eTODMode == one of CBFMTodCountMode\_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)

CBFMDst\_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:

CBFMTime\_st – mandatory

CBFMTime\_st::m\_blsInterval == false

CBFMCounterHigh16\_st and/or CBFMCounterHigh32\_st optional

CBFMLocalDate\_st – local date extension

CBFMDstBia\_st – Daylight Saving Time metadata extension (if applicable)

CBFMDstTransDay\_st - Daylight Saving Change Day

CBFMLocalDate\_st::m\_eTODMode == CBFMTodCountMode\_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:

CBFMTime\_st – mandatory

CBFMTime\_st::m\_blsInterval == false

CBFMCounterHigh16\_st and/or CBFMCounterHigh32\_st optional

#### 4.2.1.4 As time-point timestamp >= 86400 seconds:

CBFMTime\_st – mandatory

CBFMTime\_st::m\_blsInterval == false

CBFMCounterHigh16\_st and/or CBFMCounterHigh32\_st optional

CBFM24HourPeriod\_st – 24 hour period counter extension

### 4.2.2 As Time Interval

If CBFMTime\_st::m\_blsInterval == true the CBFM represents an interval, rather than a time-point.

Used without the optional CBFM24HourPeriod\_st extension the CBFM represents an interval less than 24 hours (< 86400 seconds).

If optional CBFM24HourPeriod\_st extension is present the CBFM represents an interval equal to or greater than 24 hours (>= 86400 seconds). The CBFM24HourPeriod\_st::m\_ul24HourPeriods value represents a count of 24 hour periods (1 == 86400 seconds) and the CBFMTime\_st counter value represents a count of 86400 seconds within that 24 hour period. The CBFM24HourPeriod\_st shall be present if the state of CBFMTime\_st::m\_blsInterval == true and the total counter value exceeds 86400.

Note intervals  $\geq 86400$  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:

CBFMTime\_st – mandatory  
CBFMTime\_st::m\_bIsInterval == true  
CBFMCounterHigh16\_st and/or CBFMCounterHigh32\_st optional

#### 4.2.2.2 As time interval $\geq 86400$ seconds:

CBFMTime\_st – mandatory  
CBFMTime\_st::m\_bIsInterval == true  
CBFMCounterHigh16\_st and/or CBFMCounterHigh32\_st optional  
CBFM24HourPeriod\_st – 24 hour period counter extension

### 4.2.3 Assembly Order

A CBFM shall be formed with the mandatory CBFMTime\_st structure with optional components concatenated in the following order:

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

CBFMTime\_st - mandatory and CBFMTime\_st::m\_bIsInterval == false  
CBFMCounterHigh16\_st and/or CBFMCounterHigh32\_st optional  
extensions to form 48, 64, or 80 bit counter

- To form a time-point timestamp  $\geq 86400$  seconds:

CBFMTime\_st - mandatory and CBFMTime\_st::m\_bIsInterval == false  
CBFMCounterHigh16\_st and/or CBFMCounterHigh32\_st optional  
extensions to form 48, 64, or 80 bit counter  
CBFM24HourPeriod\_st - optional 24 hour period counter

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

CBFMTime\_st - mandatory and CBFMTime\_st::m\_bIsInterval == false  
CBFMCounterHigh16\_st and/or CBFMCounterHigh32\_st optional  
extensions to form 48, 64, or 80 bit counter  
CBFMLocalDate\_st - optional local date extension  
CBFMDst\_st - optional Daylight Savings Time metadata extension if applicable  
CBFMUtcShift\_st - optional UTC-offset shift metadata extension if applicable  
Variable length Posix TZ environment string of CBFMChar\_st  
in array CCBf::m\_aCBFMChar\_st16Abbr[]

#### Interval and LocalDate are exclusive

- To form an interval  $< 86400$  seconds:

CBFMTime\_st - mandatory and CBFMTime\_st::m\_bIsInterval == true  
CBFMCounterHigh16\_st and/or CBFMCounterHigh32\_st optional  
extensions to form 48, 64, or 80 bit counter

- To form an interval  $\geq 86400$  seconds:

CBFMTime\_st - mandatory and CBFMTime\_st::m\_bIsInterval == true  
CBFMCounterHigh16\_st and/or CBFMCounterHigh32\_st optional  
extensions to form 48, 64, or 80 bit counter  
CBFM24HourPeriod\_st - optional day duration counter

CBFM assembly order in pseudocode -

```
CBFMTime_st - mandatory primary component  
if(m_CBFMTime_st.m_eCounterSize == COUNTERSIZE_48)  
    append CBFMCounterHigh16_st  
if(m_CBFMTime_st.m_eCounterSize == COUNTERSIZE_64)  
    append CBFMCounterHigh32_st  
if(m_CBFMTime_st.m_eCounterSize == COUNTERSIZE_80)  
    append CBFMCounterHigh16_st  
    append CBFMCounterHigh32_st
```

```

// Interval and LocalDate are exclusive
if(m_CBFMTime_st.m_bIsInterval)
{
    if(m_CBFMTime_st.m_b24HourPeriodExt)
        append CBFM24HourPeriod_st
    }
else
{
    if(m_CBFMTime_st.m_bLocalDateExt)
        append CBFMLocalDate_st
    if(m_CBFMLocalDate_st.m_bUtcShiftExt)
        append CBFMUtcShift_st
    if(m_CBFMLocalDate_st.m_bDSTExt)
        append CBFMDst_st
    if(m_CBFMLocalDate_st.m_TZDTimeZoneID_st.m_bCBFMAbbrExt)
        append number of CBFMChar_st in array m_aCBFMChar_st16Abbr[]
    if(m_CBFMLocalDate_st.m_TZDTimeZoneID_st.m_bCBFMLocationExt)
        append CBFMLocation_st
    }
}

```

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

Assembled CBFM size guide:

- Minimum size 8 bytes (mandatory CBFMTime\_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 GPGBA form and `m_CCTCParams_st.m_CBFLocation_st.m_bSourceIsExtern == true`.

See `CCctC::SetLocation()`

See `CTzDataParse::LatLgnAltToCBFLocation_stUTIL()`

### 4.3.1 Geographic Coordinates

The `CBFLocation_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`, `CBFLocation_st`

See `CCctM.h`, `CCctM.cpp` class `CCctM`

See `CCbfM.h`, `CCbfM.cpp` class `CCbcM`

## 4.4 CTE RIFF Wrapper

One or more CTEs may be contained in a CTE RIFF Wrapper.

The four-cc of the CBFM RIFF shall be " CTE"

The four-cc of each CBFM chunk shall be " ccte"

See `CCTRiffCCct.h`

```
#define FOURCC_CCTE "CTE" // RIFF CTE header
```

```
#define FOURCC_ccte "ccte" // RIFF CTE chunk
```

See `CCTRiffCCct.h` and `CCTRiffCCct.cpp`

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

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

The CBFM RIFF Wrapper can be opened, populating one or more CCBf classes, and closed by

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

## 5 Common Calendar Media Character Format (CCFM)

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 CBFM binary format providing complete symmetrical conversion between the two.

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

A CCF, like its corresponding CBFM 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 CBFM 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 CBFM support only complete and deterministic representation of local date and time including the required local time metadata. Other representations are outside the scope of CBFM 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: <ul style="list-style-type: none"> <li>singly - time point &lt; 86400s</li> <li>with Date and TOD_LEAPSECOND_MIDNIGHT or TOD_LEAPSECOND_UTC_UTC or TOD_LEAPSECOND_UTC_NTP or TOD_LEAPSECOND_UTC_POSIX <ul style="list-style-type: none"> <li>- time-of-day of UTC accurate local date</li> </ul> </li> <li>with Date and TOD_NONE <ul style="list-style-type: none"> <li>- accurate local date and time having no relation to the date</li> </ul> </li> <li>with Event - time point &gt;= 86400s</li> </ul>	T23:59:59n999999999
E	Event with Time - time point >= 86400s	E123T23:59:59n999999999
I	Interval: <ul style="list-style-type: none"> <li>singly - time Interval &lt; 86400s</li> <li>with Period - time interval &gt;= 86400s</li> </ul>	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
V	TzDb release version	V2021a
M	Time-of-day (TOD) Count Mode:	Mu
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 CBFM binary data structure is CBFMTime\_st. The CBFM member variable: CBFMTime\_st::m\_blsInterval flags if the data represents a time point (m\_blsInterval == false) or interval (m\_blsInterval == 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 CBFM

CBFMTime\_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 CBFMRate\_et enumeration in the CBFMTime\_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	CBFM rate CBFMRate_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	1/100000000000000000 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 CBFM binary data structures are CBFMTime\_st., CBFMLocalDate\_st, and CBFMDst\_st.

Delimiters	Description	CBFM member variable state
D and T plus applicable metadata delimiters	UTC accurate local date and time-of-day time point	CBFMTime_st::m_bIsInterval == false CBFMLocalDate_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 CBFM binary data structures are CBFMTime\_st., CBFMLocalDate\_st, and CBFMDst\_st.

Delimiters	Description	CBFM member variable state
D and T plus applicable metadata delimiters	Time-of-day unrelated to UTC date	CBFMTime_st::m_bIsInterval == false CBFMLocalDate_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	CBFM member variable state
T	Time point ((T)ime) less than 24 hours (< 86400 seconds) with no relation to date	CBFMTime_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	CBFM member variable state
E and T	Time point ((E)vent) 24 hours or greater ( $\geq$ 86400 seconds) with no relation to date	CBFMTime_st::m_bIsInterval == false

Example:

E123T23:59:59X - time-point (Event)  $\geq$  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	CBFM member variable state
I	Interval ((I)nterval) less than 24 hours ( $<$ 86400 seconds)	CBFMTime_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	CBFM member variable state
P and I	Interval ((P)eriod) 24 hours or greater ( $\geq$ 86400 seconds)	CBFMTime_st::m_bIsInterval == true.

Example:

P123I23:59:59u999999X - interval (Period)  $\geq$  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  
I23:59:59u9999999X - interval (Interval)  $<$  24 hours in microseconds  
E1T23:59:59X - time-point (Event)  $\geq$  24 hours in seconds  
E12T23:59:59m999X - time-point (Event)  $\geq$  24 hours in milliseconds  
E123T23:59:59n999999999X - time-point (Event)  $\geq$  24 hours in nanoseconds  
P1I23:59:59X - interval (Period)  $\geq$  24 hours in seconds  
P2I23:59:59u9999999X - interval (Period)  $\geq$  24 hours in microseconds  
D1972-06-30T19:59:60U-05:00Znew\_yorkV2016cL10\*Sc4sMuX - Local Date and time

See CBFM.h,

CBFMTime\_st

CBFMRATE\_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 CBFM binary data structure is CBFM24HourPeriod\_st.

*This is a range of approximately 27378 years, far greater than the approximate 3000 year range supported by Common Calendar. The corresponding CBFM 21-bit data member CBFM24HourPeriod\_st::m\_ul24HourPeriods has a range of  $2^{21}$ -bits = 2097152 MAX.*

Delimiters	Description	CBFM member variable state
E and T	Time point ((E)vent) 24 hours or greater ( $\geq 86400$ seconds) with no relation to date	CBFMTime_st::m_blsInterval == false

Examples:

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

See CBFM.h, CBFM24HourPeriod\_st  
 CBFM24HourPeriod\_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 CBFM binary data structure is CBFM24HourPeriod\_st.

Delimiters	Description	CBFM member variable state
P and I	Interval ((P)eriod) 24 hours or greater ( $\geq 86400$ seconds)	CBFMTime_st::m_blsInterval == true.

Examples:

```
P1T23:59:59X          - interval (Period)  $\geq$  24 hours in seconds
P2T23:59:59u9999999X - interval (Period)  $\geq$  24 hours in microseconds
```

See CBFM.h, CBFM24HourPeriod\_st  
 CBFM24HourPeriod\_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:  
D1972-01-01

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 `CBFM CBFMTime_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()
```

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\_IUTCOffset (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.

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\_IUTCOffset

See CBF.h, CBFLocalDate\_st::m\_IUTCOffset  
See CCcf.cpp, CCcf::SetUTCOffsetAndZoneFromCCbf()  
CCcf::ParseUTCOffsetSetCCbf()

#### 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 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-04Zamerica/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()

#### 5.5 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

DTUZAVLSMX required  
EIP 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

DTUZVLMX required  
SEIP 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_CBFLocationExt == 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)ordinates) 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 , // 1/1000000000000000000000 zeptosecond
    // CLOCK_24 , // 1/1000000000000000000000000 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 - CCFM Character Set

CCFM 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 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-04Zamerica/new_yorkAedtV2021aL25*S01cMuX
-----
```

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

-- Interval ((I)nterval) < 86400s --

I301:46:38m999X

CBFMTIME\_st::

```
m_eRateEnumeration CLOCK_3
m_bLocalDateExt     FALSE
m_b24HourPeriodExt  FALSE
m_bCounterSign      positive
m_eCounterSize      COUNTERSIZE_35
m_ulCounterLow32    1086398999
CBFMTIME_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 --

PI100:00:00m000X

CBFMTIME\_st::

```
m_eRateEnumeration CLOCK_3
m_bLocalDateExt     FALSE
m_b24HourPeriodExt  TRUE
m_bCounterSign      positive
m_eCounterSize      COUNTERSIZE_35
m_ulCounterLow32    0
CBFMTIME_st Counter 0
CBFM24HourPeriod_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

CBFMTIME\_st::

```
m_eRateEnumeration CLOCK_3
m_bLocalDateExt     FALSE
m_b24HourPeriodExt  FALSE
m_bCounterSign      positive
```

```
m_eCounterSize      COUNTERSIZE_35
m_ulCounterLow32    1086398999
CBFMTime_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

CBFMTime\_st::

```
m_eRateEnumeration  CLOCK_3
m_bLocalDateExt     FALSE
m_b24HourPeriodExt  TRUE
m_bCounterSign      positive
m_eCounterSize      COUNTERSIZE_35
m_ulCounterLow32    0
CBFMTime_st Counter 0
CBFM24HourPeriod_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

CBFMTime\_st::

```
m_eRateEnumeration  CLOCK_3
m_bLocalDateExt     TRUE
m_b24HourPeriodExt  FALSE
m_bCounterSign      positive
m_eCounterSize      COUNTERSIZE_35
m_ulCounterLow32    86399999
CBFMTime_st Counter 86399999
CBFMLocalDate_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

CBFMTime\_st::

```
m_eRateEnumeration  CLOCK_3
m_bLocalDateExt     TRUE
m_b24HourPeriodExt  FALSE
m_bCounterSign      positive
m_eCounterSize      COUNTERSIZE_35
m_ulCounterLow32    86400999
```

```

CBFMTIME_st Counter 86400999
CBFMTIME_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

```

```

CBFMTIME_st::
  m_eRateEnumeration CLOCK_3
  m_bLocalDateExt TRUE
  m_b24HourPeriodExt FALSE
  m_bCounterSign positive
  m_eCounterSize COUNTERSIZE_35
  m_ulCounterLow32 0
CBFMTIME_st Counter 0
CBFMTIME_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

```

```

CBFMTIME_st::
  m_eRateEnumeration CLOCK_3
  m_bLocalDateExt TRUE
  m_b24HourPeriodExt FALSE

```

```

m_bCounterSign    positive
m_eCounterSize    COUNTERSIZE_35
m_ulCounterLow32  84000999
CBFMTime_st Counter 84000999
CBFMLocalDate_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

```

CBFMTime_st::
  m_eRateEnumeration CLOCK_3
  m_bLocalDateExt     TRUE
  m_b24HourPeriodExt  FALSE
  m_bCounterSign      positive
  m_eCounterSize      COUNTERSIZE_35
  m_ulCounterLow32    72000999
CBFMTime_st Counter 72000999
CBFMLocalDate_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
CBFMTime_st::
  m_eRateEnumeration CLOCK_3
  m_bLocalDateExt     TRUE
  m_b24HourPeriodExt  FALSE
  m_bCounterSign      positive
  m_eCounterSize      COUNTERSIZE_35
  m_ulCounterLow32    0
CBFMTime_st Counter 0
CBFMLocalDate_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
CBFMTime_st::
  m_eRateEnumeration CLOCK_9
  m_bLocalDateExt     TRUE
  m_b24HourPeriodExt  FALSE
  m_bCounterSign      positive
  m_eCounterSize      COUNTERSIZE_48
  m_ulCounterLow32    1634811903
CBFMCounterHigh16_st::
  m_unCounterHigh16   1676
CBFMTime_st Counter 7199999999999
CBFMLocalDate_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

```

```

CBFMTIME_st::
m_eRateEnumeration  CLOCK_9
m_bLocalDateExt     TRUE
m_b24HourPeriodExt  FALSE
m_bCounterSign      positive
m_eCounterSize      COUNTERSIZE_48
m_ulCounterLow32    2452217856
CBFMCounterHigh16_st::
m_unCounterHigh16   2514
CBFMTIME_st Counter  10800000000000
CBFMLocalDate_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_i21Lgn_uMin       23
m_iAlt_cm           16777215
m_bSourceIsExtern   0
m_bIsValidLat       1
m_bIsValidLgn       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
CBFMTime_st::
  m_eRateEnumeration CLOCK_9
  m_bLocalDateExt     TRUE
  m_b24HourPeriodExt  FALSE
  m_bCounterSign      positive
  m_eCounterSize      COUNTERSIZE_48
  m_ulCounterLow32    2452217857
CBFMCounterHigh16_st::
  m_unCounterHigh16   2514
CBFMTime_st Counter  108000000000001
CBFMLocalDate_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
  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_i21Lgn_uMin        23
  m_iAlt_cm             16777215
  m_bSourceIsExtern    0
  m_bIsValidLat        1
  m_bIsValidLgn        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.