Common Calendar Conventional Timestamp

Common Calendar Timestamp System

Brooks Harris Version 1 2025-02-02

Table of Contents

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

1 Introduction

The Common Calendar Conventional Timestamp (CCTC) presents a standardized form of traditional timestamps.

Most timekeeping systems adhere to the sets of rules specified in several standards:

- The *IEEE Portable Operating System Interface* (POSIX) specifies rules that define how time will be handled within operating systems. It based on the ISO/IEC 9899, Programming languages, C. These rules underpin most timekeeping interoperability.
- POSIX specifies that date and time be recoded as a zero-based integer count of seconds-since-UTC1970 (time_t) and how this count is to be encoded in YMDhms form (struct tm, "broken down time"). This is the basic design of conventional timekeeping.
- *IANA Time Zone Database* (TzDb) supplements POSIX with time zones and local time rule sets. It maintains source data, algorithms and reference implementations (zic.c and zdump.c) to express local time consistent with POSIX time.
- *ISO 8601 Date and time, Representations for information interchange* (8601) defines methods for character representation of date and time. It provides various optional representations and is typically used by systems to represent date and time in character form both as interoperable text strings and for presentation to human users.
- *ITU-R Recommendation TF.460* defines how leap-seconds are to be incorporated in the calendar YMDhms representation.

Systems combine these rule sets to implement timekeeping on various platforms. There are technical subtleties to these specifications and their implementations. The historical development on many platforms over the years has resulted in a wide variety of slightly different practices depending on the era of development, the capability of the machines and the choices made to satisfy user requirements within the constraints of the available technology. Some of these inconsistencies persist in modern systems and there is no single standardized unambiguous timestamp.

CCTC has been developed to provide uniform and comprehensive representation of local time. CCTC can be used in most cases where conventional timestamps are used such as operating systems, file systems, network time, databases and desktops in binary and character formats.

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

Common Calendar Conventional Character Format (CCFC) provides a character-based format that acts in concert with CBFC with symmetrical conversion between the two.

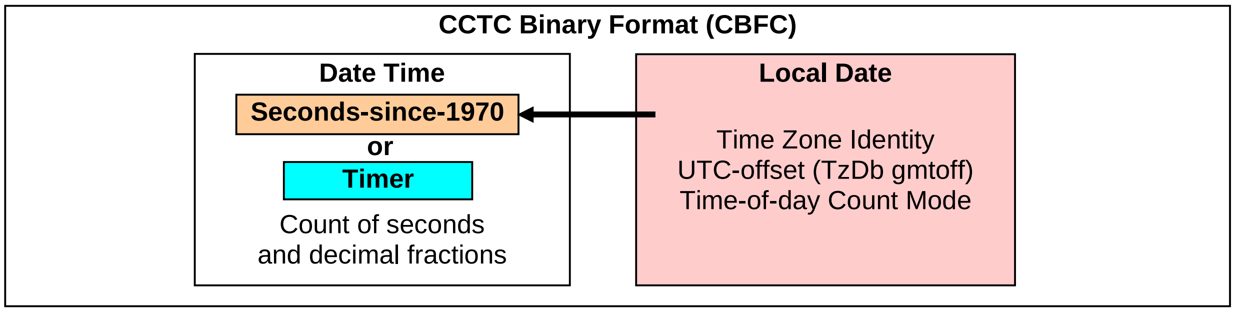
Optional extensions can support Society of Moving Pictures and Television Engineers (SMPTE) timecode labeling of video and audio media.

Common Character Conventional API provides high-level methods for constructing, manipulating, writing, reading of the timestamp formats.

Common Calendar Conventional Binary Format (CBFC)

Common Calendar Conventional Binary Format (CBFC) specifies a binary data format. Like POSIX time, it is made up of seconds-since-UTC1970 and decimal-fractions-of-seconds combined with TzDb time zone metadata. The design is intended for binary systems, such as operating systems, embedded systems, protocols such as time dissemination, and expression in languages such as c/c++, Java and XML.

The CBFC is made up of the mandatory Date Time structure that provides the anchor for any configuration of CBFC, and optional extensions, including the important Local Date structure.



Date Time structure is made up of a 6-byte 48-bit zero-based count of seconds-since-UTC1970 and decimal-fractions-of-seconds. Used by itself it is a simple 24-hour timer. The Local Date structure extension may be added to provide full local date and time-of-day representation. Its time zone metadata uses elements derived from the Common Calendar Time Zone API.

A Time-of-day Count Mode is included to indicate the YMDhms labeling sequence.

Common Calendar Conventional Character Format (CCFC)

Common Calendar Character Format (CCFC) 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.

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

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

CCTC Character Format (CCFC)

```
D2024-11-03T01:00:00.999U-07Zamerica/denverV2024aMuX
```

The CCT reference implementation implements CCFC and CBFC in the CCTCLib library. CBFC is implemented as class CCbfC and the CBFC data types are defined in CBFC.h header. CCFC is implemented as class CCcfC, and delimiter characters are explicitly defined in the CCFC.h header. See CCTCLib/CCFC.h, CCcfC.h, and CCcfC.cpp. See CCTCLib/CBFC.h, CCbfC.h, and CCctC.cpp.

Geostamp

The Common Calendar Timestamp (CCT) specification can be 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 CCT timestamp and geographic coordinates.

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

In addition to CCT Conventional (CCTC), the subject of this document it, Common Calendar also presents two other types, or classes, of timestamps:
CCT Media (CCTM) - Optimized for labeling video and audio media.
CCT Enhanced (CCTE) – Similar to CCTM while also maintaining aspects of the TzDb source data which are typically not included in timestamps used in common practice.

Common Character Conventional API

Common Character Conventional API provides high-level methods for constructing, manipulating, writing, reading of the timestamp formats.

2 Scope

This interoperability standard specifies a standardized form of timestamps in binary and character formats consistent with POSIX-time, TzDb time zone rules, and ISO 8601 text formatting. It includes 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

IEEE Portable Operating System Interface, IEEE Std 1003.1™-2024 Edition
https://pubs.opengroup.org/onlinepubs/9799919799/

ISO/IEC 9899 - Programming languages - C
https://www.open-std.org/JTC1/SC22/WG14/www/standards

IANA Time Zone Database
https://www.iana.org/time-zones

RFC 6557, Procedures for Maintaining the Time Zone Database
https://www.rfc-editor.org/rfc/rfc6557.html

RFC 9636, The Time Zone Information Format (TZif)
https://www.rfc-editor.org/rfc/rfc9636.html#RFC7808

RFC 4833, Timezone Options for DHCP
https://www.rfc-editor.org/rfc/rfc4833

RFC 5545, Internet Calendaring and Scheduling Core Object Specification (iCalendar)
https://www.rfc-editor.org/rfc/rfc5545

ITU-R Recommendation ITU-R TF.460-6, Standard-frequency and time-signal emissions
https://www.itu.int/rec/R-REC-TF.460/en

ISO 8601-1:2019, Date and time - Representations for information interchange
https://www.iso.org/iso-8601-date-and-time-format.html

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 Conventional Binary Format (CBFC)

Common Calendar Conventional Binary Format (CBFC) specifies a compact variable length binary data structure containing

1) Mandatory CCTDateTime_st structure holding seconds-since-UTC1970 or seconds (as timer or interval).
2) Optional CCTCDecimalFrac32_st structure holding decimal fractions of seconds at a specified rate (resolution).
3) Optional CCTCLocalDate_st holding TzDb time zone identity, TzDb release (version), and current local UTC-offset.

A CBFC and its corresponding CCFC format may 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 CBFC and its corresponding CCFC shall contain seconds-since-UTC1970, and time zone metadata as known to the emitting system when generated in accordance with the rules and guidelines set out in Common Calendar Local Timescales.

A CBFC 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 CBFC 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 CBFC Components

A CBFC is constructed from the mandatory Date Time data structure CCTDateTime_st with optional extensions for decimal fractions of seconds and time zone. 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 CBFC Construction and Assembly.

4.1.1 Date Time

The CCTDateTime_st structure is mandatory and provides the fixed-size (64-bit) anchor of the variable length format. It contains a count of seconds and decimal fractions since UTC1970 and extension flags indicating presence of decimal fractions and time zone extensions. The counter is a 48-bit 6-byte data type, giving a range of approximately -2229912 to 2229912 years.

```
typedef struct CCTDateTime_st      // 8 bytes, 64-bits
{
unsigned char m_eDecFracRate:4;    // Enumerated rates of decimal fractions
                                   // See CRateTable.h CBFRate_et
unsigned char m_bLocalDateExt:1;   // CCTCLocalDate_st extension
                                   // is present
                                   // See CBFLocalDate_st
unsigned char m_bIsInterval:1;     // CCTC is an interval
unsigned char m_bSecsIsNegative:1; // 1 = seconds counter value is negative
unsigned char m_Reserved1:1;
unsigned char m_Reserved8:8;
                                // Seconds since 1970 zero-based counter
unsigned short m_unSecsHigh16;  // 16 bit unsigned high word
unsigned long m_ulSecsLow32;    // 32 bit unsigned low word
                                // high/low 48 bits 6 bytes
                                // -140737488355328.00 min
                                // 140737488355327.00 max
                                // range 1628906115.22 days
```

```
                                          // approx 4459823.99 years;
                                          // approx -2229912 to 2229912 years
} CCTDateTime_st;
```

m_eDecFracRate
Enumerated value of rate, or resolution, of decimal fraction. The rates are enumerated in CRateTable.h CBFRate_et. If m_eDecFracRate > CLOCK_0 (seconds) the CCTCDecimalFrac32_st shall be present. The CCTCDecimalFrac32_st struct holds the value of the decimal fraction of seconds at the indicated rate.

m_bLocalDateExt
Flag indicating the presence of the CCTCLocalDate_st stuct. If CCTCLocalDate_st is present the CCTC timestamp is a date and time-of-day. See section Local Date.

m_**bIsInterval**
Flag indicating the values of CCCT are an interval.

m_**bSecsIsNegative**
Flag indication counter value is negative

m_**unSecsHigh16**
unsigned short high word of counter

m_**ulSecsLow32**
unsigned long low word of counter

4.1.2 Decimal Fraction Rate

CBFC supports several rates, or resolutions, of decimal fraction of seconds. These are indicated by enumerations. CCT "standard rates" are enumerated in CCTRateLib, CRateTable.h.
See Annex A - CCT Standard Rates

```
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
-- abridged --
} CBFRate_et;
```

4.1.3 Decimal Fractions of Seconds

Optional if decimal fractions of second are included. Presence indicated by CCTDateTime_st::m_eDecFracRate == one of the `CLOCK_x` CTCRate_et enumerations greater than `CLOCK_0`.

```
typedef struct CCTCDecimalFrac32_st  // 4 bytes, 32 bit
{
signed long m_lDecimalFrac32; // 2147483647 max -2147478648 min decfrac
                              // 1000000000 max nanoseconds
} CCTCDecimalFrac32_st;
```

4.1.4 Local Date

The optional CCTCLocalDate_st structure represents the time zone identity together with UTC-offset. The presence of CBFCLocalDate_st is indicated by CCTDateTime_st:: m_bLocalDateExt..

The local time information is obtained from the IANI Time Zone Database (TzDb) through the Time Zone API, including the time zone identity and UTC-offset. See TzDatabaseApi.h.

The `m_lCurUTCOffset` member is the primary UTC-offset plus any secondary (DST) offset that may be in effect, consistent with the TzDb gmtoff value.

```
typedef struct CCTCLocalDate_st  // 8 bytes, 64-bit
{
CCTCZoneID_st m_CCTCZoneID_st;    // TzDb zone index and release
signed long m_lCurUTCOffset:21;   // Current UTC Offset in seconds
                                  // as per 8601 and TzDb gmtoff
signed long m_eTodCntMode:3;      // Enumerated Leap-second Count Mode
                                  // See CBFLSTODMODE_et
signed long m_lReserved:8;
} CCTCLocalDate_st;
```

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

4.1.4.1 Time Zone Identity

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

The version of the TzDb release source files is recorded to make accurate forensic analysis possible in cases where the TzDb data may have changed since a timestamp was written.

```
typedef struct CCTCZoneID_st         // 4 bytes, 32 bits
{
unsigned short m_unZoneIdx:10;       // TzDb zone index
                                     // 2^10 = 1024 MAX
unsigned short m_unTzDbRelsLetter:5; // 26 release letters a-z
                                     // 2^5 = 32 MAX
unsigned short m_unReserved1:1;
unsigned short m_unTzDbRelsYear:12;  // UTC1970 zero based year number
                                     // 1970 + 3465 = year 5435
                                     // 2^12 = 4096 MAX
unsigned short m_unReserved4:4;
} CCTCZoneID_st;
```

See Common Calendar Time Zone API

4.1.4.2 Current UTC Offset

m_lCurUTCOffset

4.1.4.3 Time-of-Day Count Mode

CBFTodCntMode_et instructs how the CBF binary representation is to be converted to the CCFC YMDhms counting sequence representation. These are declared in CCTRateLib, CRateTable.h. Options include:.

- 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. “Rolling leap-seconds”. CCF TOD Count Mode char indicator "m" (midnight)
- TOD_LEAPSECOND_24HOUR_DAY - 86400-second-days without leap-seconds, POSIX time_t., CCF TOD Count Mode char indicator "g" (Gregorian)

- TOD_24HOUR_DAY_NOT_TOD - UTC accurate local date with time portion having no relation to the date, a timer. 86400-second-days without leap-seconds, POSIX time_t., CCF TOD Count Mode char indicator "t" (timer)

```
typedef enum CBFTodCntMode_et
{
TOD_LEAPSECOND_NOT_SET,   // not set or logic error (default)
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_MIDNIGHT,  // Leap-seconds introduced at
                          // midnight on local timescales
                          // (Rolling leap-seconds)
                          // CCF char indicator "m" (midnight)
TOD_24HOUR_DAY,           // 86400-second-days of calendar
                          // leap-seconds unknown. unavailable
                          // or not applicable
                          // This is POSIX time_t
                          // seconds-since-UTC1970
                          // without leap-seconds
                          // CCF char indicator "g" (gregorian)
TOD_24HOUR_DAY_NOT_TOD    // 86400-second-days of calendar date
                          // without leap-seconds
                          // accurate date
                          // inaccurate time-of-day
                          // time-of-day decoupled from date
                          // CCF char indicator "t" (timer
} CBFTodCntMode_et;
```

See Common Calendar Local Timescales.

4.2 CBFC Construction

CCTDateTime_st structure is mandatory and the anchor of the binary image.

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

4.2.1 Time-point

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

Used without the optional CBFCLocalDate_st extension, the values of a CCTDateTime_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 CBFCLocalDate_st extension the CBFC represents a leap-second accurate local date and time. The CCTDateTime_st counter value represents time-of-day (24 hours plus one leapsecond if applicable) on the calendar date indicated by the CBFCLocalDate_st member values. The presence of CBFCLocalDate_st is indicated by the CCTDateTime_st:: m_bLocalDateExt flag.

4.2.1.1 UTC accurate local date and time-of-day

CCTDateTime_st – mandatory
CCTDateTime_st::m_bIsInterval = false
CCTDateTime_st::m_eDecFracRate = CBFRate_et enumeration
CCTCDecimalFrac32_st - decimal fraction extension if CCTDateTime_st::m_eDecFracRate > CLOCK_0
CCTDateTime_st::m_unSecsHigh16, m_ulSecsLow32 – seconds-since-UTC1970
CCTDateTime_st:: m_bLocalDateExt = true
CBFCLocalDate_st – local date extension
CBFCLocalDate_st m_lDecimalFrac32 –faction of second at CCTDateTime_st::m_eDecFracRate
CCTCLocalDate_st ::CCTCZoneID_st::m_unZoneIdx = selected TzDb time zone index
CBFCLocalDate_st:: m_lCurUTCOffset = current UTC-offset (TzDb dstoff)
CBFCLocalDate_st::m_eTodCntMode = one of CBFTodCntMode_et enumerations:

- 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)

The CCTC character representation has several delimited fields. For example, In New York, the second before the November DST ‘fall back’ at CLOCK_7, 100-nanosecond, with TOD_LEAPSECOND_UTC_UTC count mode:

CCctC call to set CCbfC binary:
```
CCctC->SetDateTime("America/New_York", TOD_LEAPSECOND_UTC_UTC,
                     1730613626, 123, CLOCK_7);
```

As CCcfC character format:
```
D2024-11-03T01:59:59.0000123U-04Zamerica/new_yorkV2024aMuX
// UTC  1730613626
```

4.2.1.2 UTC accurate local date with arbitrary time-of-day

CCTDateTime_st – mandatory
CCTDateTime_st::m_bIsInterval = false
CCTDateTime_st:: m_bLocalDateExt = true
CBFCLocalDate_st – local date extension
CCTCLocalDate_st ::CCTCZoneID_st::m_unZoneIdx = selected TzDb time zone index
CBFCLocalDate_st:: m_lCurUTCOffset = current UTC-offset (TzDb dstoff)
CCTCLocalDate_st ::m_eLsTodCntMode = TOD_24HOUR_DAY_NOT_TOD

Time point with UTC accurate local date with time portion having no relation to the date. This supports uses such timers during a date, such as game clocks or rocket launches (“T-10 and counting..”), where the hms portion of the timestamp is a timer, not time-of-day.

4.2.1.3 Time-point less than 24 hours

CCTDateTime_st – mandatory
CCTDateTime_st::m_bIsInterval == false

If the CCTDateTime_st seconds-since-UTC1970 is less than 24 hours (< 86400 seconds) the CCCT timestamp represents an interval less than 86400 seconds.

The CCTC character representation is delimited with “I” (for Interval) followed by hmsd. For example,an intervalof 86399.999 seconds at CLOCK_3, milliseconds:

CCctC call to set CCbfC binary:
```
CCctC->SetEvent(86399, 999, CLOCK_3);
```

As CCcfC character format:
```
T23:59:59.999X // 0000086399.999 CLOCK_3
```

4.2.1.4 Time-point 24 hours or greater (Event)

CCTDateTime_st – mandatory
CCTDateTime_st::m_bIsInterval == false

If the CCTDateTime_st seconds-since-UTC1970 is less than 24 hours (< 86400 seconds) the CCCT timestamp represents a time-point less than 86400 seconds.

The CCTC character representation is delimited with “E” (for 24-hout Event) followed a 24-hour value, followed by an “T” delimiter, followed by hmsd. For example, 2 24-hour periods and 86399.999 seconds at CLOCK_3, milliseconds:

CCctC call to set CCbfC binary:
```
CCctC->SetEvent((86400 * 2) + 86399, 999, CLOCK_3);
```

As CCcfC character format:
```
E2T23:59:59.999X // 0000259199.999 CLOCK_3
```

4.2.2 Interval

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

4.2.2.1 Interval less than 24 hours

CCTDateTime_st – mandatory
CCTDateTime_st::m_bIsInterval == true

If the CCTDateTime_st seconds-since-UTC1970 is less than 24 hours (< 86400 seconds) the CCCT timestamp represents an interval less than 86400 seconds.

The CCTC character representation is delimited with “I” (for Interval) followed by hmsd, For example, 86399.999 seconds at CLOCK_3, milliseconds:

CCctC call to set CCbfC binary:
```
CCctC->SetInterval(86399, 999, CLOCK_3);
```

As CCcfC character format:
```
I23:59:59.999X //0000086399.999 CLOCK_3
```

4.2.2.2 Interval 24 hours or greater (Period)

CCTDateTime_st – mandatory
CCTDateTime_st::m_bIsInterval == true

If the CCTDateTime_st seconds-since-UTC1970 is equal or greater than 24 hours (>= 86400 seconds) the CCCT timestamp represents an interval equal or greater than 86400 seconds.

The CCTC character representation is delimited with “P” (for 24-hout Period) followed a 24-hour period value, followed by an “I” delimiter, followed by hmsd. For example, 2 24-hour periods and 86399.999 seconds at CLOCK_3, milliseconds:

CCctC call to set CCbfC binary:
```
CCctC->SetInterval((86400 * 2) + 86399, 999, CLOCK_3);
```

As CCcfC character format:
```
P2I23:59:59.999X //0000259199.999 CLOCK_3
```

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

4.3 Geostamp

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

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

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

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

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

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

CCTC supports two forms of location:

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

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

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

4.3.1 Geographic Coordinates

The CBFcLocation_st struct carries geographic coordinates.

```
typedef struct CBFLocation_st  // 14 bytes
{
signed long m_i21Lat_uMin:21;  // micro-minutes -100000 to 100000 range
                               // [((2^21)/2)-1  =  1048575 MAX]
signed long m_i9Lat_Deg:9;     // degrees -90 to 90 range, negative is South
                               // [((2^9) / 2) - 1 = 255 MAX]
signed long m_Pad1:2;
signed long m_i21Lng_uMin:21;  // micro-minutes -100000 to 100000 range
                               // [((2^21)/2)-1  =  1048575 MAX]
signed long m_i9Lng_Deg:9;     // degrees -180 to 180 range, negative is West
                               // [((2^9) / 2) - 1 = 255 MAX]
signed long m_Pad2:2;
signed long m_i25Alt_cm:25;    // signed centimeters range [((2^25)/2)*-1 MIN
                               // -16777216, ((2^25)/2)-1 MAX 16777215
signed long m_i7Lng_Min:7;     // minutes 0 to 60 range
                               // [((2^7) / 2) - 1 = 63 MAX]

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

See TzDatabaseApi.h, CBFCLocation_st
See CCctC.h, CCctC.cpp class CCctC
See CCbfC.h, CCbfC.cpp class CCbcC

4.4 Assembly Order

A CBFC shall be formed with the mandatory CCTDateTime_st structure with optional components concatenated in the following order:

- To form a time-point timestamp < 86400 seconds:
`CCTDateTime_st` - mandatory and `CCTDateTime_st::m_bIsInterval = false`

- To form a time-point timestamp >= 86400 seconds:
`CCTDateTime_st` - mandatory and `CCTDateTime_st::m_bIsInterval = false`

- To form a UTC accurate local time-point timestamp:
`CCTDateTime_st` - mandatory and `CCTDateTime_st::m_bIsInterval = false`
`CBFCLocalDate_st` - local date extension

Interval and LocalDate are exclusive

- To form a interval < 86400 seconds:
`CCTDateTime` - mandatory and `CCTDateTime_st::m_bIsInterval = true`

- To form an interval >= 86400 seconds:
`CCTDateTime` - mandatory and `CCTDateTime_st::m_bIsInterval = true`

See `CCcfC::SetCcfCAssembleStrings()`

4.5 RIFF Wrapper

CCctC::AssembleCbfC() generates a raw CCbfC binary image. These are intended for cases where the timestamps are known within the context of use. Some applications may need or prefer to identify those raw images. The CCTC RIFF wrapper (class CRiff_CCct) provides means to wrap the raw timestamp in a RIFF four-cc format.

The four-cc of the CBFC RIFF shall be " CCTC"
The four-cc of each CBFC chunk shall be " cctc"
See CCTRiffCCct.h

```
#define FOURCC_CCTC "CCTC" // RIFF CCTC header
#define FOURCC_cctc "cctc" // RIFF CCTC chunk
```

See CCTRiffCCct.h and CCTRiffCCct.cpp

One or more CCTCs may be contained in a CCTC RIFF Wrapper.

First, set the output/input path for the RIFF wrapper file:

```
CCctC::SetCRiff_CCctPath(sRiffCbfPath);
```

A single CBFC can be assembled and written to a RIFF wrapper by:

```
CCctC::AssembleAndWriteSingleItemCRiff_CCct_CCct();
```

A series of CBFCs can be written to a RIFF wrapper by:

1) Create RIFF:
`CCctC::CreateCRiff_CCctAppendLIST()`

2) For each CBFC
`CCctC::AssembleAndAppendToCRiff_CCct()`

3) Complete and close
`CCctC::WriteListAndCompleteCRiff_CCct()`

The CBFC RIFF Wrapper can be opened and read, populating one or more CBFC timestamps by:

`CCctC::ReadCRiff_CCct()`

5 Common Calendar Conventional Character Format (CCFC)

Common Calendar Conventional Character Format (CCFC) 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 CBFC binary format providing complete symmetrical conversion between the two.

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

A CCFC, like its corresponding CBFC 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 CCFC and its corresponding CBFC 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.cpp, CCTDemoTestsCCctC.cpp, TestSelectedCCctCConfigurationsAndShowCbfCValues()`.

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.
- TzDb time zone identity is added
- TzDb release (version) is added
- The character "Z" is used to delimit the time zone identification data field, replacing the ISO 8601 use of “Z” for "Zulu"

CCFC 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, CCFC and CBFC support only complete and deterministic representation of local date and time including the required local time metadata. Other representations are outside the scope of CBFC and CCFC.

5.2 Character Set

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

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

5.3 Hard Terminator

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

5.4 Data Field Elements

A CCFC 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.

CCFC 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.

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

Delimiting Character	Element	Example fragment(s)
`T`	Time: • singly - time point < 86400s • with Date and TOD_LEAPSECOND_MIDNIGHT or TOD_LEAPSECOND_UTC_UTC or TOD_LEAPSECOND_UTC_NTP or TOD_LEAPSECOND_UTC_POSIX - time-of-day of UTC accurate local date • with Date and TOD_24HOUR_DAY_NOT_TOD Accurate local date with time portion having no relation to the date time-of-day	`T23:59:59.999999999`
`E`	Event with Time - time point >= 86400s	`E123T23:59:59.999999999`
`I`	Interval: • singly - time Interval < 86400s • with Period - time interval >= 86400s	`I23:59:59.999999999`
`P`	Period with Interval - time interval >= 86400s	`P123I23:59:59.999999999`
`D`	Date	`D2016-02-22`
`U`	UTC Offset	`U-5, U+5, U+02:30:17, U+3w01+02`
`Z`	(Z)one – TzDb time zone identity	`Zamerica/new_york`
`V`	TzDb release version	`V2021a`
`M`	Time-of-day Count Mode:	`Mu`
`X`	Hard terminator	`X`

5.4.1 Time Element

Encodes a timer, an interval or, in combination with a Date Element, time-of-day.

The corresponding CBFC component is CCTDateTime_st.

The time element represents a time-point or interval in the familiar hours:minutes:seconds form, that is; 60 seconds = 1 minute, 60 minutes = 1 hour, and 24 hours = one 24 hour (86400 second) period.

If the time element appears alone, it shall represent one of:

If delimited by T –
A time-point ((T)ime) less than 24 hours (< 86400 seconds) with no relation to date. The corresponding CBFC CTDateTime_st::m_bIsInterval member shall be set to false.

If delimited by I –
An interval ((I)nterval) less than 24 hours (< 86400 seconds)
The corresponding CBFC CTDateTime_st::m_bIsInterval member shall be set to true.

If the time element appears in combination with a 24-hour Period Element it shall represent one of:

If delimited by E and T -

A time-point ((E)vent) 24 hours or greater (>= 86400 seconds)with no relation to date. The corresponding CBFC CTDateTime_st::m_bIsInterval member shall be set to false.

If delimited by P and I -
An interval ((P)eriod) 24 hours or greater (>= 86400 seconds). The corresponding CBFC CTDateTime_st::m_bIsInterval member shall be set to true.

If the time element appears in combination with the date element ("DYYYY-MM-DD") together with its required metadata elements) it shall represent the time-of-day portion of a complete UTC accurate local date and time. See Date Element.

The first eight characters of the time element shall be fixed length containing two-digit values of hours, minutes, and seconds, separated by colons, for example "00:00:00".

If a period character (".") follows the hh:mm:ss field some number of numeric characters (digits) shall follow up to the next element delimiter. The number of digits indicates the decimal fraction of seconds resolution.

The corresponding value of a binary CBFC m_CCTDateTime_st::m_eDecFracRate member shall be set as indicated in the following table:

Digits	CBFC rate CBFCRate_et	
`None`	CLOCK_0	1/1 second
`1`	CLOCK_1	1/10 tenths of second
`2`	CLOCK_2	1/100 hundredths of second
`3`	CLOCK_3	1/1000 millisecond
`4`	CLOCK_4	1/10000 10ths of millisecond, 100 microsecond
`5`	CLOCK_5	1/100000 100ths of millisecond, 1000 microsecond
`6`	CLOCK_6	1/1000000 microsecond
`7`	CLOCK_7	1/10000000 10ths of microsecond, 100 nanosecond
`8`	CLOCK_8	1/100000000 100ths of microsecond, 1000 nanosecond
`9`	CLOCK_9	1/1000000000 nanosecond

For example, 3 digits indicates milliseconds; "12:34:56.999". The corresponding CBFC m_CCTDateTime_st::m_eDecFracRate member is CLOCK_3.

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 one of TOD_LEAPSECOND_MIDNIGHT, TOD_LEAPSECOND_UTC_UTC, 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 CBFC binary data structures are CCTDateTime_st and CBFCLocalDate_st.

Delimiters	Description	CBFC member variable state
`D` and `T` plus applicable metadata delimiters	UTC accurate local date and time-of-day time point	`CCTDateTime_st::m_bIsInterval == false CBFCLocalDate_st::m_eTodCntMode == TOD_LEAPSECOND_UTC_UTC or TOD_LEAPSECOND_UTC_NTP or TOD_LEAPSECOND_UTC_POSIX Or TOD_LEAPSECOND_MIDNIGHT`

Example:
`D2015-06-30T19:59:60U-04Zamerica/new_yorkV2024aMuX`

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_24HOUR_DAY_NOT_TOD, character Mt, it shall represent a time having no relation to the UTC accurate local date. See Date Element.

The corresponding CBFC binary data structures are CCTDateTime_st and CBFCLocalDate_st.

Delimiters	Description	CBFC member variable state
`D` and `T` plus applicable metadata delimiters	Time-of-day unrelated to UTC date	`CCTDateTime_st::m_bIsInterval == false` `CBFCLocalDate_st::m_eTodCntMode == TOD_24HOUR_DAY_NOT_TOD`

Example:
`D2015-06-30T12:00:00U-04Zamerica/new_yorkV2024aMuX`

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

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

5.4.1.4 Time point equal or greater than 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	CBFC member variable state
E and T	Time point ((E)vent) 24 hours or greater (>= 86400 seconds) with no relation to date	`CCTDateTime_st::m_bIsInterval == false`

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

5.4.1.5 Interval less than 86400s

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

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

Example:
`I23:59:59.999999X` - 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	CBFC member variable state
P and `I`	Interval ((P)eriod) 24 hours or greater (>= 86400 seconds)	`CCTDateTime_st::m_bIsInterval == true.`

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

Examples summary:

`T23:59:59X`	- time-point (Time) in seconds
`T23:59:59.999X`	- time-point (Time) in milliseconds
`T23:59:59.999999999X`	- time-point (Time) in nanoseconds
`I23:59:59X`	- interval (Interval) < 24 hours in seconds
`I23:59:59.999999X`	- interval (Interval) < 24 hours in microseconds
`E1T23:59:59X`	- time-point (Event) >= 24 hours in seconds

```
E12T23:59:59.999X          -  time-point (Event) >= 24 hours in  milliseconds
E123T23:59:59.999999999X -  time-point (Event) >= 24 hours in nanoseconds
P1I23:59:59X               -  interval (Period) >= 24 hours in seconds
P2I23:59:59.999999X        -  interval (Period) >= 24 hours in microseconds
D2015-06-30T19:59:60U-04Zamerica/new_yorkV2024aMuX -  Local Date and time
```

See CCTC.h, CCTDateTime_st
CBFCRATE_et
See CCcfC.cpp, CCcfC::SetTimeFromCCbfC()
CCcfC::ParseTimeSetCCbfC()

5.5 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.

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

Examples:
```
E1T23:59:59X             - time-point (Event) >= 24 hours in seconds
E12T23:59:59.999X        - time-point (Event) >= 24 hours in  milliseconds
E123T23:59:59.999999999X - time-point (Event) >= 24 hours in nanoseconds
```

See CBFC.h, CBFC24HourPeriod_st
CBFC24HourPeriod_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()

5.6 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.

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

Examples:
```
P1T23:59:59X             - interval (Period) >= 24 hours in seconds
P2T23:59:59.999999X      - interval (Period) >= 24 hours in microseconds
```

See CBFC.h, CBFC24HourPeriod_st
CBFC24HourPeriod_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()

5.7 Date Element

Encodes local date.

The corresponding CBFC component is CBFCLocalDate_st.

A CBFC with date shall include the CBFCLocalDate_st structure, indicated by CCTDateTime_st::m_bLocalDateExt == true.

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 element is given it shall be accompanied by:

T	Time Element (time-of-day)
U	UTC Offset Element
Z	Zone Element
V	TzDb Version Element
M	TOD Count Mode Element

A date element together with its required metadata elements describes a time-point, indicated by CBFC CCTDateTime_st::m_bIsInterval == false.

The meaning of the timestamp and the YMDhms sequence shall be governed by the TOD Count Mode CBFTodCntMode_et enumerated values of CBFCLocalDate_st:: m_eTodCntMode. See TOD Count Mode Element.

Construction of CCFC YMDhms from CBFC binary data shall employ the algorithms described by the YMDhms API The YMDhms values will depend on CBFCLocalDate_st::m_eTodCntMode.

In the case of UTC accurate data and time, where CBFCLocalDate_st::m_eTodCntMode is one of TOD_LEAPSECOND_UTC_UTC, TOD_LEAPSECOND_UTC_NTP, TOD_LEAPSECOND_UTC_POSIX or TOD_LEAPSECOND_MIDNIGHT, positive leap-second days are 86401 seconds, the last hour of the day (23) is 3601 seconds and its last minute 61 seconds.

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

The details of the CBFC to CCFC conversion are implemented in:
CCcfC.cpp, CCcfC::SetCcfCFromCCbfC ()
CCcfC::SetCcfCFromCCbfC_TOD_LEAPSECOND()
CCcfC::SetCcfCFromCCbfC_TOD_24HOUR_DAY()

Parsing of a CCFC string and populating a CBF binary is shown in:
CCcfC.cpp, CCcfC::ParseDateSetCbfC()

Examples:

`D1972-06-30T19:59:60U-04Zamerica/new_yorkV2024aMuX`
- 1972-06-30T19:59:60 date and time-of-day
- U-04 - UTC-offset
- Zamerica/new_york - time zone

- V2024a - Tz Database version
- Mu - TOD Count Mode TOD_LEAPSECOND_UTC_UTC
- X - terminator

`D1972-07-01T00:59:60U+01Zeurope/berlinV2024aMuX`

- 1972-07-01T00:59:60 date and time-of-day
- U+01 - UTC offset
- Zeurope/berlin - time zone
- V2024a - Tz Database version
- Mu - TOD Count Mode TOD_LEAPSECOND_UTC_UTC
- X - terminator

5.8 UTC Offset Element

Encodes the current local time zone offset from UTC in seconds resolution as a variable length +-hh:mm:ss representation.

The corresponding CBFC member is CCTCLocalDate_st::m_lCurUTCOffset.
(This is the same value given by TzDb and POSIX as tm_gmtoff.)

The UTC Offset element shall be delimited with uppercase “U”, for (U)TC Offset.
Shall be followed with the “+” or “-“ UTC offset sign
Shall be followed by variable length hh, mm, or ss:
If even hour, 2 chars, 2 digits of hours
If even minute, 5 chars, 2 digits of hours, colon, 2 digits of minutes.
If seconds, 7 chars, 2 digits of hours, colon, 2 digits of minutes, colon, 2 digits of seconds

A zero UTC offset shall be preceded by the “+” sign. “-00” is reserved as an error indicator.

Example fragments:
U+00
U+02
U+05:45
U+00:12:12
U-06
U-03:30
U-04:56:02

Example:
`D2015-06-30T19:59:60U-04Zamerica/new_yorkV2024aMuX`

See CCTC.h, CCTCLocalDate_st::m_lCurUTCOffset
See CCcf.cpp, CCcfC::SetUTCOffsetAndZoneFromCCbfC()
CCcfC::ParseUTCOffsetSetCCbfC()

5.9 Time Zone Element

Encodes the TzDb time zone identifier.

The corresponding CBF member is CCTCLocalDate_st::m_CCTCZoneID_st::m_unZoneIdx..

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

Delimiter	Time Zone name
Z	variable length string as lower-case of TzDb time zone identifier

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

Example:

`D2015-06-30T19:59:60U-04Zamerica/new_yorkV2024aMuX`

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

See CCTC.h, CBFLocalDate_st::m_TZDTimeZone_st
See CCcf.cpp,
    CCcfC::SetUTCOffsetAndZoneFromCCbf()
    CCcfC::ParseZoneSetCCbf()
See Common Calendar Time Zone API

5.10 IANA Time Zone Database Version Element

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

The corresponding CBFC members are
CCTCLocalDate_st::m_CCTCZoneID_st::m_unTzDataReleaseYear
CCTCLocalDate_st::m_CCTCZoneID_st::m_unTzDataReleaseLetter

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:
`D2015-06-30T19:59:60U-04Zamerica/new_yorkV2024aMuX`

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

See CCTC.h, CBFLocalDate_st::m_TZDTimeZone_st
See CCcfC.cpp, CCcfC::SetUTCOffsetAndZoneFromCCbf()
    CCcfC::ParseTzVersionSetCCbf()

5.11TOD Count Mode Element

Encodes the TOD Count Mode value.

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

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

The corresponding CBFC member is CBFCLocalDate_st::m_eTodCntMode.

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
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")

m	TOD_LEAPSECOND_MIDNIGHT	Leap-seconds introduced at midnight on local timescales (Rolling leap-second)
g	TOD_24HOUR_DAY_DATE	86400-second-days of calendar (leap-seconds unknown or unavailable)
t	TOD_24HOUR_DAY_NOT_TOD	Time has no relation to date or time-of-day

Example fragments:
`Mu` (TOD_LEAPSECOND_UTC_UTC)
`Mn` (TOD_LEAPSECOND_UTC_NTP)
`Mp` (TOD_LEAPSECOND_UTC_POSIX)
`Mm` (TOD_LEAPSECOND_MIDNIGHT)
`Mg` (TOD_24HOUR_DAY_DATE)
Mt (TOD_24HOUR_DAY_NOT_TOD)

Example:
```
D2015-06-30T19:59:60U-04Zamerica/new_yorkV2024aMuX
```

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

See CRateTable.h, CBFTodCntMode_et
See CCTC.h, CBFCLocalDate_st::m_eTodCntMode
See CCct.cpp, CCcfC::SetCcfCFromCCbfC()
CCcfC::ParseTODModeSetCCbf()

5.12 Assembly and Order

A CCFC 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.

5.12.1 UTC accurate local date and time-of-day

To represent a UTC accurate local date and time time point a CCFC 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 TzDb release	required
6	M	TOD Count Mode shall be u TOD_LEAPSECOND_UTC_UTC or n TOD_LEAPSECOND_UTC_NTP or p TOD_LEAPSECOND_UTC_POSIX or g TOD_LEAPSECOND_MIDNIGHT	required
7	X	terminator	required
	E	Event (24 hour period)	excluded
	I	Interval (24 hour period)	excluded
	P	Period (24 hour period)	excluded

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

Corresponding CBF variable states
```
   CCTDateTime_st::m_bIsInterval = false
   CCTDateTime_st::m_bLocalDateExt == true
   CBFCLocalDate_st::m_eTodCntMode ==
      TOD_LEAPSECOND_UTC_UTC or
      TOD_LEAPSECOND_UTC_NTP or
      TOD_LEAPSECOND_UTC_POSIX or
      TOD_LEAPSECOND_MIDNIGHT
```

Example:
D2015-06-30T19:59:60U-04Zamerica/new_yorkV2024aMuX

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

5.12.2 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 CCFC 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 TzDb release	required
6	M	TOD Count Mode shall be t TOD_24HOUR_DAY_NOT_TOD	required
7	X	terminator	required
	E	Event (24 hour period)	excluded
	I	Interval (24 hour period)	excluded
	P	Period (24 hour period)	excluded

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

Corresponding CBF variable states
```
   CCTDateTime_st::m_bIsInterval == false
   CCTDateTime_st::m_bLocalDateExt == true
   CBFCLocalDate_st::m_eTodCntMode == TOD_24HOUR_DAY_NOT_TOD
```

Example:
D2015-06-30T19:59:60U-04Zamerica/new_yorkV2024aMtX

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

5.13Time point less than 86400s

To represent a time point less than 24 hours (< 86400s) a CCFC 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 M	excluded

Indicated time shall be less than 86400 seconds.

Corresponding CBF variable states
```
   CCTDateTime_st::m_bIsInterval == false
   CCTDateTime_st::m_bLocalDateExt == false
```

Example:
`T23:59:59X`

5.14Time point equal or greater than 86400s

To represent a time point 24 hours or greater (>= 86400s) a CCFC 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 M	excluded

Corresponding CBF variable states
```
   CCTDateTime_st::m_bIsInterval == false
   CCTDateTime_st::m_bLocalDateExt == false
```

Example:
```
E1T23:59:59X
```

5.15 Interval less than 86400s

To represent an interval less than 24 hours (< 86400s) a CCFC 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 M	excluded

Indicated interval shall be less than 86400 seconds.

Corresponding CBF variable states
```
   CCTDateTime_st::m_bIsInterval == true
   CCTDateTime_st::m_bLocalDateExt == false
```

Example:
```
I23:59:59X
```

5.15.1 Interval equal or greater than 86400s

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

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

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

Corresponding CBF variable states
```
   CCTDateTime_st::m_bIsInterval == true
   CCTDateTime_st::m_bLocalDateExt == false
```

Example:
```
P1I23:59:59X
```

5.16 Geostamp

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

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

CCTC supports two forms of location:

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

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

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

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

Altitude is not given by TzDb zone.tab.

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

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

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

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

5.16.1 Location Element

Encodes geographic coordinates.

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

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

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

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

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

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

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

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

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

Example fragments:
As TzDb zone.tab:
```
Czt+404251g-0740023
```

As NMEA 0183 GPGGA:
```
Cet+4042.85000g-00740.38333a123.45
```

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

6 Common Calendar Conventional API

Common Calendar Conventional API defines a set of commands to set, get, and manipulate CCTC binary and character formats. These are declared and implemented in class CCctC. See CCctC.h and CCctC.cpp.

A brief sample of methods are illustrated below.

CCctC::SetDateTime(SecsFrac_st* pSecsFrac_st);

CCctC::SetYMDhmsd(int iYYYY, int iMM, int iDD, int ihh, int imm, int iss, uint64_t ui64dd);

CCctC::GetSecsFrac(SecsFrac_st* pSecsFrac_st);

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 - CCFC Character Set

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

Disallowed Characters

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

Active Characters

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

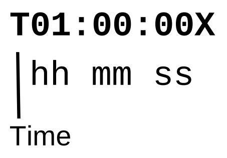
Character	Dec	Hex	Name
!	33	(0x21)	Exclamation
"	34	(0x22h)	Quote
#	35	(0x23)	Number
$	36	(0x24)	Dollar
%	37	(0x25)	Percent
&	38	(0x26)	Ampersand
'	39	(0x27)	Apostrophe
(	40	(0x28)	Open Parenthesis
)	41	(0x29)	Close Parenthesis
*	42	(0x2A)	Asterisk
+	43	(0x2B)	Plus
,	44	(0x2C)	Comma
-	45	(0x2D)	Hyphen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

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

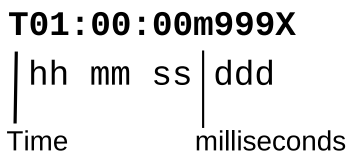
p	112	(0x70)	
q	113	(0x71)	
r	114	(0x72)	
s	115	(0x73)	
t	116	(0x74)	
u	117	(0x75)	
v	118	(0x76)	
w	119	(0x77)	
x	120	(0x78)	
y	121	(0x79)	
z	122	(0x7A)	
{	123	(0x7B)	Open Brace
|	124	(0x7C)	Vertical Bar
}	125	(0x7D)	Close Brace
~	126	(0x7E)	Tilde

Annex C - CCFC Example Illustrations

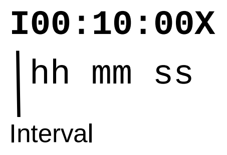
Time point < 86400s in seconds



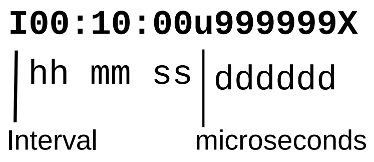
Time point < 86400s in milliseconds



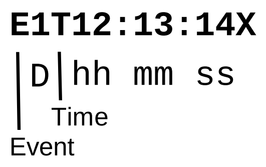
Interval < 86400s in seconds



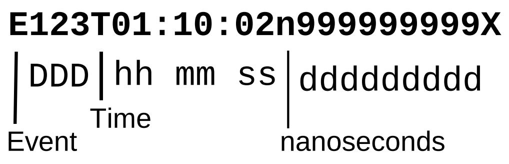
Interval < 86400s in microseconds



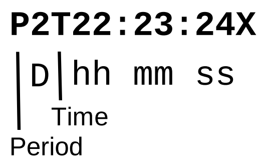
Time point >= 86400s in seconds



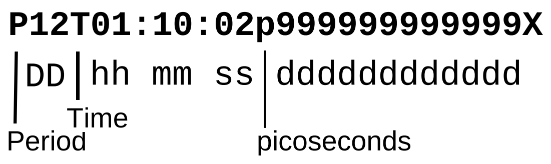
Time point >= 86400s in nanoseconds



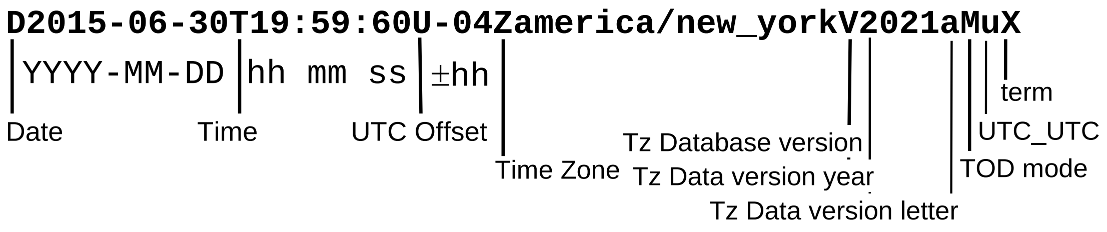
Interval >= 86400s in seconds



Interval >= 86400s in picoseconds



2015 Leap-second in New York with TOD_LEAPSECOND_UTC_UTC Count Mode



Annex D - Example Listing from CCT Reference Implementation

```
CCT Version 3.0.0.0 2025-01-20 00:00:00

Time flies when you're having fun!


File out: ..\OutputFiles\CCTOut_BB_TESTSELECTEDCCTCCONFIGURATIONSANDSHOWCBFCVALUES.txt


======= TestSelectedCCctCConfigurationsAndShowCbfCValues() =======


-------- Common Calendar Conventional Character Format (CCFC) --------
Date       Time     UTC Offset
|          |        |   Zone (TzDb time zone identity)
|          |        |   |                Version (Tz Database release)
|          |        |   |                |     Mode (TOD count mode)
|          |        |   |                |     | X terminator
D2015-06-30T19:59:60U-04Zamerica/new_yorkV2024aMuX
----------------------------------------------------------------------


--------------------------------------
CCFC, CCbfC member values and CBFC binary
--------------------------------------

-- Interval ((I)nterval) < 86400s --
I23:59:59.999X
m_CCTDateTime_st:
  m_eDecFracRate:4          CLOCK_3
  m_bLocalDateExt:1         false
  m_bIsInterval:1           true
  m_bSecsIsNegative:1       false
  m_unSecsHigh16            0000000000
  m_ulSecsLow32             0000086399
  Seconds                         86399
m_CCTCDecimalFrac32_st::
  m_lDecimalFrac32          00000999
CBF binary interchange bytes as hexadecimal
24 00 00 00 7f 51 01 00 -e7 03 00 00  size 12

-- Interval ((P)eriod) >= 86400s --
P1I00:00:00.000X
m_CCTDateTime_st:
  m_eDecFracRate:4          CLOCK_3
  m_bLocalDateExt:1         false
  m_bIsInterval:1           true
  m_bSecsIsNegative:1       false
  m_unSecsHigh16            0000000000
  m_ulSecsLow32             0000086400
  Seconds                         86400
m_CCTCDecimalFrac32_st::
  m_lDecimalFrac32          00000000
CBF binary interchange bytes as hexadecimal
24 00 00 00 -80 51 01 00 00 00 00 00  size 12

-- Time point ((T)ime) < 86400s --
T23:59:59.999999999X
m_CCTDateTime_st:
  m_eDecFracRate:4          CLOCK_3
  m_bLocalDateExt:1         false
  m_bIsInterval:1           false
  m_bSecsIsNegative:1       false
  m_unSecsHigh16            0000000000
  m_ulSecsLow32             0000086399
  Seconds                         86399
```

```
m_CCTCDecimalFrac32_st::
  m_lDecimalFrac32          999999999
CBF binary interchange bytes as hexadecimal
04 00 00 00 7f 51 01 00 -ff -c9 -9a 3b  size 12

-- Time point ((E)vent) >= 86400s --
E1T00:00:00.000X
m_CCTDateTime_st:
  m_eDecFracRate:4          CLOCK_3
  m_bLocalDateExt:1         false
  m_bIsInterval:1           false
  m_bSecsIsNegative:1       false
  m_unSecsHigh16            0000000000
  m_ulSecsLow32             0000086400
  Seconds                         86400
m_CCTCDecimalFrac32_st::
  m_lDecimalFrac32          00000000
CBF binary interchange bytes as hexadecimal
04 00 00 00 -80 51 01 00 00 00 00 00  size 12

-- Second preceding 1972 leap-second in UTC time zone --
D1972-06-30T23:59:59U+00Zetc/utcV2024aMgX
m_CCTDateTime_st:
  m_eDecFracRate:4          CLOCK_0
  m_bLocalDateExt:1         true
  m_bIsInterval:1           false
  m_bSecsIsNegative:1       false
  m_unSecsHigh16            0000000000
  m_ulSecsLow32             0078796799
  Seconds                      78796799
m_CCTCLocalDate_st::
  CCTCZoneID_st::
  m_unZoneIdx:10            [125] Etc/UTC
  m_unTzDbRelsYear:12       52
  m_unTzDbRelsLetter:5      0
  m_lCurUTCOffset          000000
  m_eTodCntMode:3        TOD_24HOUR_DAY
CBF binary interchange bytes as hexadecimal
11 00 00 00 -ff 57 -b2 04 7d 00 34 00 00 00 00 00 05  size 17

-- 1972 leap-second in UTC time zone --
D1972-07-01T00:00:00U+00Zetc/utcV2024aMgX
m_CCTDateTime_st:
  m_eDecFracRate:4          CLOCK_0
  m_bLocalDateExt:1         true
  m_bIsInterval:1           false
  m_bSecsIsNegative:1       false
  m_unSecsHigh16            0000000000
  m_ulSecsLow32             0078796801
  Seconds                      78796801
m_CCTCLocalDate_st::
  CCTCZoneID_st::
  m_unZoneIdx:10            [125] Etc/UTC
  m_unTzDbRelsYear:12       52
  m_unTzDbRelsLetter:5      0
  m_lCurUTCOffset          000000
  m_eTodCntMode:3        TOD_24HOUR_DAY
CBF binary interchange bytes as hexadecimal
11 00 00 00 01 58 -b2 04 7d 00 34 00 00 00 00 00 05  size 17

-- Second following 1972 leap-second in UTC time zone --
D1972-07-01T00:00:00U+00Zetc/utcV2024aMgX
m_CCTDateTime_st:
  m_eDecFracRate:4          CLOCK_0
  m_bLocalDateExt:1         true
  m_bIsInterval:1           false
  m_bSecsIsNegative:1       false
  m_unSecsHigh16            0000000000
  m_ulSecsLow32             0078796801
  Seconds                      78796801
m_CCTCLocalDate_st::
  CCTCZoneID_st::
  m_unZoneIdx:10            [125] Etc/UTC
  m_unTzDbRelsYear:12       52
  m_unTzDbRelsLetter:5      0
  m_lCurUTCOffset          000000
  m_eTodCntMode:3        TOD_24HOUR_DAY
CBF binary interchange bytes as hexadecimal
11 00 00 00 01 58 -b2 04 7d 00 34 00 00 00 00 00 05  size 17

-- Second preceding 1972 leap-second in New York time zone --
D1972-06-30T23:19:59U-04Zamerica/new_yorkV2024aMgX
m_CCTDateTime_st:
  m_eDecFracRate:4          CLOCK_0
  m_bLocalDateExt:1         true
  m_bIsInterval:1           false
  m_bSecsIsNegative:1       false
  m_unSecsHigh16            0000000000
  m_ulSecsLow32             0078808800
  Seconds                      78808800
m_CCTCLocalDate_st::
  CCTCZoneID_st::
  m_unZoneIdx:10            [230] America/New_York
  m_unTzDbRelsYear:12       52
  m_unTzDbRelsLetter:5      0
  m_lCurUTCOffset          -14400
  m_eTodCntMode:3        TOD_24HOUR_DAY
CBF binary interchange bytes as hexadecimal
11 00 00 00 -e0 -86 -b2 04 -e6 00 34 00 -c0 -c7 1f 00 05  size 17

-- 1972 leap-second in New York time zone --
D1972-06-30T19:59:59U-04Zamerica/new_yorkV2024aMgX
m_CCTDateTime_st:
  m_eDecFracRate:4          CLOCK_0
  m_bLocalDateExt:1         true
  m_bIsInterval:1           false
  m_bSecsIsNegative:1       false
  m_unSecsHigh16            0000000000
  m_ulSecsLow32             0078796800
  Seconds                      78796800
m_CCTCLocalDate_st::
  CCTCZoneID_st::
  m_unZoneIdx:10            [230] America/New_York
  m_unTzDbRelsYear:12       52
  m_unTzDbRelsLetter:5      0
  m_lCurUTCOffset          -14400
  m_eTodCntMode:3        TOD_24HOUR_DAY
CBF binary interchange bytes as hexadecimal
11 00 00 00 00 58 -b2 04 -e6 00 34 00 -c0 -c7 1f 00 05  size 17

-- Second following 1972 leap-second in New York time zone --
D1972-07-01T00:00:00U-04Zamerica/new_yorkV2024aMgX
m_CCTDateTime_st:
  m_eDecFracRate:4          CLOCK_0
  m_bLocalDateExt:1         true
  m_bIsInterval:1           false
  m_bSecsIsNegative:1       false
  m_unSecsHigh16            0000000000
  m_ulSecsLow32             0078811201
  Seconds                      78811201
m_CCTCLocalDate_st::
  CCTCZoneID_st::
  m_unZoneIdx:10            [230] America/New_York
  m_unTzDbRelsYear:12       52
  m_unTzDbRelsLetter:5      0
  m_lCurUTCOffset          -14400
  m_eTodCntMode:3        TOD_24HOUR_DAY
CBF binary interchange bytes as hexadecimal
11 00 00 00 41 -90 -b2 04 -e6 00 34 00 -c0 -c7 1f 00 05  size 17

-- Nanosecond preceding 2016 DST Onset in New York time zone --
D2016-03-13T01:59:59.999999999U-05Zamerica/new_yorkV2024aMgX
m_CCTDateTime_st:
  m_eDecFracRate:4          CLOCK_9
  m_bLocalDateExt:1         true
  m_bIsInterval:1           false
  m_bSecsIsNegative:1       false
  m_unSecsHigh16            0000000000
  m_ulSecsLow32             1457852425
  Seconds                    1457852425
m_CCTCDecimalFrac32_st::
  m_lDecimalFrac32          999999999
m_CCTCLocalDate_st::
  CCTCZoneID_st::
  m_unZoneIdx:10            [230] America/New_York
  m_unTzDbRelsYear:12       52
  m_unTzDbRelsLetter:5      0
  m_lCurUTCOffset          -18000
  m_eTodCntMode:3        TOD_24HOUR_DAY
CBF binary interchange bytes as hexadecimal
1a 00 00 00 09 10 -e5 56 -ff -c9 -9a 3b -e6 00 34 00 -b0 -b9 1f 00 05  size 21

-- Nanosecond at 2016 DST Onset in New York time zone --
D2016-03-13T03:00:00.000000000U-04Zamerica/new_yorkV2024aMgX
m_CCTDateTime_st:
  m_eDecFracRate:4          CLOCK_9
  m_bLocalDateExt:1         true
  m_bIsInterval:1           false
  m_bSecsIsNegative:1       false
  m_unSecsHigh16            0000000000
  m_ulSecsLow32             1457852426
  Seconds                    1457852426
m_CCTCDecimalFrac32_st::
  m_lDecimalFrac32          00000000
m_CCTCLocalDate_st::
  CCTCZoneID_st::
  m_unZoneIdx:10            [230] America/New_York
  m_unTzDbRelsYear:12       52
  m_unTzDbRelsLetter:5      0
  m_lCurUTCOffset          -14400
  m_eTodCntMode:3        TOD_24HOUR_DAY
CBF binary interchange bytes as hexadecimal
1a 00 00 00 0a 10 -e5 56 00 00 00 00 -e6 00 34 00 -c0 -c7 1f 00 05  size 21

-- Nanosecond following 2016 DST Onset in New York time zone --
D2016-03-13T03:00:00.000000001U-04Zamerica/new_yorkV2024aMgX
m_CCTDateTime_st:
  m_eDecFracRate:4          CLOCK_9
  m_bLocalDateExt:1         true
  m_bIsInterval:1           false
  m_bSecsIsNegative:1       false
  m_unSecsHigh16            0000000000
  m_ulSecsLow32             1457852426
  Seconds                    1457852426
m_CCTCDecimalFrac32_st::
  m_lDecimalFrac32          00000001
m_CCTCLocalDate_st::
  CCTCZoneID_st::
  m_unZoneIdx:10            [230] America/New_York
  m_unTzDbRelsYear:12       52
  m_unTzDbRelsLetter:5      0
  m_lCurUTCOffset          -14400
  m_eTodCntMode:3        TOD_24HOUR_DAY
CBF binary interchange bytes as hexadecimal
1a 00 00 00 0a 10 -e5 56 01 00 00 00 -e6 00 34 00 -c0 -c7 1f 00 05  size 21


Your time is up.
```