Common Calendar Time Zone API

Time Zone Interface

Brooks Harris Version 3 2024-04-25

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).

"Midday" refers to any time-of-day between midnights

1 Introduction

"Civil time", sometimes called "Local time", refers to time scales established by law or custom in some jurisdiction at some geographic location for legal, commercial, and social purposes[1]. Not all systems fully support accurate and deterministic representation of local time and over coming these deficiencies is the main objective of Common Calendar. The Common Calendar Time Zone API provides representation of the IANA Time Zone Database (TzDb) local time parameters in a form suitable for implementation and consistent with the Common Calendar Local Timescales.

Local time is usually based on UTC, typically defined by its offset from UTC and any Daylight Saving Time (DST) rules in use. UTC is necessary but not sufficient; the local time parameters are also required. Those parameters include:

- Time zone name identifier
- UTC-offset
- UTC-offset Shifts
- If Daylight Saving Time (DTS) is observed
- If a DST change is an Onset (DST on or spring forward) or Retreat (DST off or fall back)
- Onset bias (the DST hh:mm:ss shift applied to the YMDhms encoding)
- Onset calendar date and time-of-day
- Retreat bias (the DST hh:mm:ss shift applied to the YMDhms encoding)

[1] There are generally two types of time zone in use: civil (land) and navigation (nautical and aeronautic). Civil time zones are usually designated as a time offset from the UTC applicable to some territory on land. Nautical time zones are specified by longitude for purposes of navigation at sea. Common Calendar is concerned only with civil time and does not address nautical time zones. (see *Date and Time Terms and Definitions, 11 Civil Time (Local Time*)

- Retreat calendar date and time-of-day
- DST state (Standard or DST in effect)

All of these variables are dynamic because any of them may change at any time due to local political or cultural decisions. Thus they are *all* required to accurately represent a local time. The details of how these rules describe local time are typically guided by several standards, de facto standards, and common practice.

Unfortunately, there is no universally applicable set of formulas that strictly define local time. The rules cannot be 100% correct in all places in all circumstances because some jurisdictions may define local time is some unusual way or make sudden changes. Nonetheless, common practice has led to a high degree of uniformity, accuracy, and confidence in most time zones where responsible decisions are made. It is especially important that notice of any upcoming changes are given well in advance and this has typically been the case in responsibly managed jurisdictions[2].

IANA maintains the Time Zone Database (TzDb) (https://www.iana.org/time-zones). It is the most comprehensive source of time zone and DST information. Originally known as the "Olson Database" or "Tz Database", it has been the de facto standard for time zones and DST rules for many timekeeping systems for decades. It has recently gained further stature since IANA has taken on responsibility for its oversight and management.

The zones and rules as known to the IANA Time Zone Database system are made available in the "Time Zone Data" files tzdata.tar.gz and are available as described in the "Distribution" section of the web site. The zones and rules for Daylight Saving Time are maintained as described by IETF Rfc 6557 Procedures for Maintaining the Time Zone Database (http://tools.ietf.org/html/rfc6557).

More detail about TzDb is offered in:
Theory and pragmatics of the tz code and data
https://www.ietf.org/timezones/code/theory.html

Common Calendar Time Zone Database API (Tz Database API) is an interface to the IANA Time Zone Database data to manage this critical set of dynamic metadata. It adapts data types for zones and rules in a form and manner consistent with Common Calendar Local Timescales.

Common Calendar Time Zone API conforms to the IANA Time Zone Database rules governing the zone and DST data and interprets their values using the formulas of Common Calendar Local Timescales and YMDhms API.

Subsequent processing applies the YMDhms API formulae with its native support of leap-seconds. The Common Calendar c++ Reference Implementation details these operations.

Common Calendar and Tz Database API enhance the IANA Time Zone Database system by adopting time zone and DST data update policies to allow deterministic prediction for at least four months. Common Calendar Binary Format and Character Format record the IANA Time Zone Database tzdata release version in their metadata to provide an audit trail for forensic determination of local time if required.

Supplementary Materials, Annex D c++ Listings ,D.2 Listing - Tz Database API, excerpts from TzDatabaseApi.h shows the related data structures.

2 Scope

Specifies a set of data types, enumerations, logical rules, and construction of lists that unambiguously describe local date and time and Daylight Saving parameters consistent with Common Calendar Local Timescales, IANA Time Zone Database, and CLDR Windows time zones.

3 Normative References

Common Calendar TAI-UTC API

Common Calendar YMDhms API

Common Calendar Local Timescales

IANA Time Zone Database

[2] For example, the USA Energy Policy Act of 2005 was signed into law August 8, 2005 specifying a new rule for DST Onset as the first Sunday on or after the 8th of March. This took effect Sunday 2007-03-11, providing 22 months of advance notice.

https://www.iana.org/time-zones

Theory and pragmatics of the tz code and data
https://www.ietf.org/timezones/code/theory.html

IETF Rfc 6557 Procedures for Maintaining the Time Zone Database
http://tools.ietf.org/html/rfc6557

IETF Rfc 8536 The Time Zone Information Format (TZif)
https://www.rfc-editor.org/rfc/rfc8536

Unicode Common Locale Data Repository (CLDR) Project
https://cldr.unicode.org/

Windows CLDR
https://raw.githubusercontent.com/unicode-org/cldr/master/common/supplemental/windowsZones.xml

4 Common Calendar Time Zone API

Common Calendar Time Zone API is an interface to the IANA Time Zone Database (TzDb) source files to manage time zone and DST metadata. It adapts data types for zones and rules in a form and manner consistent with Common Calendar Local Timescales.

Time Zone Database API conforms to the IANA Time Zone Database rules governing the zone and DST data and interprets their values using the specifications of Common Calendar Local Timescales and formulas of YMDhms API.

Common Calendar Binary Format (CBF) and Character Format (CCF) record the IANA Time Zone Database tzdata release file version in their metadata to provide an audit trail for forensic determination of local time if required.

Common Calendar Time Zone API date types, enumerations, and structures.

Data types and enumerations for time zone and DST parameters and rules are declared in TzDatabaseApi.h.

Structures assembling time zone and DST parameters into related member variables are declared in TzDatabaseApi.h.

4.1 Tz Database Data Release Version

Release version of IANA Time Zone Database tzdata source files.

Incorporating the release version in the CCT timestamps supports tracking any changes of zone and DST rules that may have occurred after a timestamp was generated.

```
typedef struct TZDDataRelease_st // 3 bytes, 24 bits
{
unsigned short m_unTzDataReleaseYear:12; // UTC1972 zero based year number
                                         // 1972 + 3465 = year 5437
                                         // 2^12 = 4096 MAX
unsigned short m_ucReserved:4;
unsigned char m_ucTzDataReleaseLetter:5; // 26 release letters a-z
                                         // 2^5 = 32 MAX
unsigned char m_ucReserved2:3;
} TZDDataRelease_st;
```

See TzDatabaseApi.h TZDDataRelease_st

4.2 Time Zone Identifiers

Time Zone Identifiers are provided for Tz Datebase and Windows time zones.

4.2.1 Tz Database Time Zone Identifiers

The IANA Time Zone Database time zone identifiers are text strings providing the primary unique identification of a particular time zone. They are coded as "continent" and "city", "town", or "territory";

For examples:

```
America/New_York
```

```
Europe/Berlin
```

TzDb refers to these as "tags". Currently there are roughly 490 time zones.

CCT uses an indexing scheme to provide compact representation of the time zone names to reduce the size of the CBF binary format and provide a convenient method of addressing time zones within the CCT implementation. These index values are established when the TzDb source files are parsed. See CTzDataParse::AssignInitialFixedZoneIndices(). See TzDatabaseApi.h:

```
typedef struct ZoneNameAndIdx_st
{
char m_sZoneName[40];
short m_nZoneIdx;
} ZoneNameAndIdx_st;
```

TzDb assigns strings for the Posix TZ environment variable. These are critical components of how Posix time operates. They are typically called "abbreviated names". CCT Time Zone API stores these as a special variable length string of type CBFChar_st. See CBF.h

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

Some time zones have changed the Posix abbreviated name irrespective of other transition factors. This rare occurrence is handled with a CBF extension to record such changes using struct CBFAbbrChange_st; See CBF.h

```
typedef struct CBFAbbrChange_st   // 5 bytes, 40-bit
{
unsigned long m_ulAbbrChangeTime:17; // Abbr change time-of-day
                                     // 86400 = 24 hours
                                     // 17 bits unsigned max
                                     // 131071 / 3600 = 36.40861111 hrs
CBFChar_st m_aCBFChar_st16Abbr_Before[16];
CBFChar_st m_aCBFChar_st16Abbr_After[16];
unsigned char m_lReserved:6;
} CBFAbbrChange_st;
```

All the time zone identification factors are collected in the `TZDTimeZoneID_st struct. See TzDatabaseApi.h:`

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

See TzDatabaseApi.h
See CBF.h
See CTzZones.h class CTzZones

4.2.2 Windows Time Zone Identifiers

CCT supports Windows time zones. The Unicode Common Locale Data Repository (CLDR) Project includes windowsZones.xml which maps TzDb time zone identifiers to the Windows time zone identifiers. For example,
TzDb "`America/New_York`" maps to Windows "`(UTC-05:00) Eastern Time (US & Canada)`".

CCT reads the windowsZones.xml and connects the Windows time zones to the TzDb time zones.

Similar to the TzDb indexing scheme, the Windows time zones are also indexed and linked to the TzDb time zone indices.

The Windows time zone data is captured in `WinZoneNameAndIdx_st.`

```
typedef struct WinZoneNameAndIdx_st
{
char m_sWinZoneName[80];
short m_nWinZoneIdx;
} WinZoneNameAndIdx_st;
```

See CWinZones.h WinZoneNameAndIdx_st
See CWinZones.h class CWinZonesXml
See CWinZones.h class CWinZones.

4.3 UTC-Offset

The offset from UTC is the first main factor distinguishing time zones. TzDb calls this value STDOFF. It is recorded in CBFLocalDate_st::m_lUTCOffset in seconds.

See CBF.h CBFLocalDate_st::m_lUTCOffset.

4.3.1 UTC-offset Shifts

Many time zones have changed their UTC-Offset independent of DST shifts. These relatively rare transitions are supported by CCT using the CBFUtcShift_st extension to CBFLocalDate_st.

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

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

See CBF.h CBFLocalDate_st::m_bUtcShiftExt
See CBF.h CBFUtcShift_st

4.4 Daylight Saving Time

Many time zones observe Daylight Saving Time (DST). TzDb supplies the applicable DST rule sets to each time zone. CCT Time Zone API collects the DST rules with the following factors.

4.4.1 Daylight Saving - CBFDstBias_st

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

Presence is signaled by CBFLocalDate_st:: m_bDstBiasExt:1

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

4.4.2 Daylight Saving - CBFDstTransDay_st

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

Presence is signaled by CBFLocalDate_st:: m_bDstTransDayExt:1

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

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

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

4.4.3 DST Count Mode

Enumeration of DST application to the YMDhms count sequence in CCF character format.

```
typedef enum CBFDstCountMode_et
{
DSTCOUNTMODE_NOTSETORNOTAPPLICABLE = 0,
DSTCOUNTMODE_CONVENTIONAL,
DSTCOUNTMODE_UNINTERRUPTED
} CBFDstCountMode_et;
```

See TzDatabaseApi.h `CBFDstCountMode_et`
See CBF.h `CBFDst_st::m_eDSTCountMode`

Two DST “count modes” are defined for YMDhms representations: “conventional” and “uninterrupted”.

- “DST Conventional Count Mode” produces the familiar and expected discontinuity in the YMDhms sequence with DST onset (DST on, spring forward) or retreat (DST off, fall back).
- “DST Uninterrupted Count Mode” is provided for systems or applications that cannot tolerate, or choose to avoid, these discontinuities. This count mode delays the DST shift in the YMDhms sequence until the end of the day yielding an uninterrupted incrementing YMDhms value until midnight.

This capability is supported by rules and data of the Tz Database API and defined by the YMDhms API. See Common Calendar Binary Format, excerpts from CBF.h, CBFDstCountMode_et. See Common Calendar Character Format Annex C - Example Listing from CCT Reference Implementation,

4.4.3.1 DST Count Mode - Conventional

The Conventional DST Count Mode produces the familiar and expected discontinuity in the YMDhms sequence at the DST change. Example America/New_York.

Common Calendar Character Format (CCF)	Seconds since UTC1970
DST Onset	
D2016-03-13T01:59:58U-05America/new_yorkAestV2021aL26Swo02+01cMmX	1457852424
D2016-03-13T01:59:59U-05America/new_yorkAestV2021aL26Swo02+01uMmX	1457852425
D2016-03-13T03:00:00U-04America/new_yorkAedtV2021aL26Sso02+01uMmX	1457852426
D2016-03-13T03:00:01U-04America/new_yorkAedtV2021aL26Sso02+01uMmX	1457852427
DST Retreat	
D2016-11-06T01:59:58U-05America/new_yorkAedtV2021aL26Ssr02+01uMmX	1478415624
D2016-11-06T01:59:59U-05America/new_yorkAedtV2021aL26Ssr02+01uMmX	1478415625
D2016-11-06T01:00:00U-05America/new_yorkAestV2021aL26Swr02+01uMmX	1478415626
D2016-11-06T01:00:01U-05America/new_yorkAestV2021aL26Swr02+01uMmX	1478415627

4.4.3.2 DST Count Mode - Uninterrupted

The Uninterrupted DST Count Mode produces a 23 hour day on DST Onset days and a 25 hour day for DST Retreat days for the typical 1 hour DST bias. This requires the seconds-to-YMDhms algorithm support counting to 24:59:59. Example America/New_York.

Common Calendar Character Format (CCF)	Seconds since UTC1970
DST Onset delayed until midnight (no midday discontinuity)	
D2016-03-13T01:59:58U-05America/new_yorkAestV2021aL26Swo23+01cMuX	1457852424
D2016-03-13T01:59:58U-05America/new_yorkAestV2021aL26Swo23+01cMuX	1457852425
D2016-03-13T02:00:00U-05America/new_yorkAestV2021aL26Swo23+01cMuX	1457852426
D2016-03-13T02:00:01U-05America/new_yorkAestV2021aL26Swo23+01cMuX	1457852427
DST Onset midnight rollover at 23 hours	
D2016-03-13T22:59:58U-04America/new_yorkAedtV2021aL26Swo23+01cMuX	1457924424
D2016-03-13T22:59:59U-04America/new_yorkAedtV2021aL26Swo23+01cMuX	1457924425
D2016-03-13T00:00:00U-04America/new_yorkAedtV2021aL26Ss+01cMuX	1457924426
D2016-03-13T00:00:01U-04America/new_yorkAedtV2021aL26Ss+01cMuX	1457924427
DST Retreat delayed until midnight (no midday discontinuity)	
D2016-11-06T01:59:58U-05America/new_yorkAestV2021aL26Ssr25+01cMmX	1478415624
D2016-11-06T01:59:59U-05America/new_yorkAestV2021aL26Ssr25+01cMmX	1478415625
D2016-11-06T02:00:00U-05America/new_yorkAestV2021aL26Ssr25+01cMmX	1478415626
D2016-11-06T02:00:01U-05America/new_yorkAestV2021aL26Ssr25+01cMmX	1478415627
DST Retreat midnight rollover at 25 hours	
D2016-11-06T24:59:58U-05America/new_yorkAestV2021aL26Ssr25+01cMmX	1478494824
D2016-11-06T24:59:59U-05America/new_yorkAestV2021aL26Ssr25+01cMmX	1478494825
D2016-11-06T00:00:00U-05America/new_yorkAestV2021aL26SwcMmX	1478494826
D2016-11-06T00:00:01U-05America/new_yorkAestV2021aL26SwcMmX	1478494827

4.4.4 DST Rules

The DST Rules, struct TZDDstRule_st, are internal to the CCT implementation. They are used to collect DST rules from the TzDb source files and processing to collate the DST rules for producing the CBF binary format results.

Sections below show the enumeration types used.

```
typedef struct TZDDstRule_st
{
int m_iDstYearIn;     // [0-4095] (zero based count)
                      // 2^12 = 4096 MAX
MONTH_et m_MONTH_et;  // [1-12] (one based count)
                      // 2^4 = 16 MAX
                      // See TzDatabaseApi.h MONTH_et
int m_iDstYearOut;    // [0-4095] (zero based count)
                      // 2^12 = 4096 MAX
DAYOFWEEK_et m_DAYOFWEEK_et;  // [0-6] (Sunday = 0)
                              // 2^3 = 8 MAX
                              // See TzDatabaseApi.h DAYOFWEEK_et
TZDDataRelease_st m_TZDDataRelease_st; // Tz Database version
                                       // see TZDDataRelease_st
TZDDstIsOnset_et m_TZDDstIsOnset_et;   // 1 = DST_ONSET, 0 = DST_RETREAT
                                       // See TZDDstIsOnset_et
```

```
int m_eDstDayOfMonth;              // [0-30] (days enumerated values 1-31)
                                   // 2^5 = 32 MAX
                                   // DAYOFMONTH_NOTAPPLICABLE0 == (MAX 32)
                                   // See TzDatabaseApi.h DAYOFMONTH_et
int64_t m_i64DstBias;              // Dst Bias in seconds
int64_t m_i64DstTimeOfDay;         // Dst transition time-of-day in seconds
TZDDstDayOfMonthRule_et m_eDstDayOfMonthRule; // See enum
                                              // TZDDstDayOfMonthRule_et
                                              // 2^3 = 8 MAX
DSTTIME_et m_eDSTTOD_et;   // CTzDataParse::PopulateRule() // Rule case AT
char m_sTzRuleName[64];    // Tz Database # Rule NAME like "US","Toronto"
short m_nTzLineNumRule;    // TzDb rule source file line number
int64_t m_i641970TzTransAtUT;
int64_t m_i641970TzTransAtLocal;
int64_t m_i641970TzTransAtLocalLs;
int m_todisstd; // is r_tod standard time?
int m_todisut;  // is r_tod UT?
                // else is WALL
char m_sr_abbrvar[5];
char m_sr_abbr[16];
} TZDDstRule_st;
```

See TzDatabaseApi.h TZDDstRule_st

4.4.4.1 DST Onset or Retreat

Enumeration of DST Onset or Retreat Rule. Indicates this DST rule is an Onset or Retreat rule

```
typedef enum TZDDstIsOnset_et
{
DST_RETREAT = 0,
DST_ONSET = 1
} TZDDstIsOnset_et;
```

See TzDatabaseApi.h `TZDDstIsOnset_et`
See TzDatabaseApi.h TZDDstRule_st::m_TZDDstIsOnset_et

4.4.4.2 DST Day of Month Rule

Enumerated form of IANA Time Zone Database DST change day-of-month rules.

```
typedef enum TZDDstDayOfMonthRule_et // 2^3 == 8 MAX
{
DSTRULE_NOT_KNOWN = 0, // not set or error
DSTRULE_NOT_OBSERVED = 1,
DSTRULE_DAYOFWEEK_LESSTHANEQUAL_DAYOFMONTH,
DSTRULE_DAYOFWEEK_GREATERTHANEQUAL_DAYOFMONTH,
DSTRULE_DAYOFWEEK_LESSTHAN_DAYOFMONTH,
DSTRULE_DAYOFWEEK_GREATERTHAN_DAYOFMONTH,
DSTRULE_LAST_DAYOFWEEKOFMONTH,
DSTRULE_DAYOFMONTH
} TZDDstDayOfMonthRule_et;
```

See TzDatabaseApi.h TZDDstDayOfMonthRule_et
See TzDatabaseApi.h TZDDstRule_st::TZDDstDayOfMonthRule_et