Common Calendar Reference Implementation Guide

CCTMetaGen and CCT Clients

Brooks Harris Version 3 2024-04-30

The Common Calendar specification is available at:
common-calendar.org

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 1970-01-01T00:00:00 (UTC).

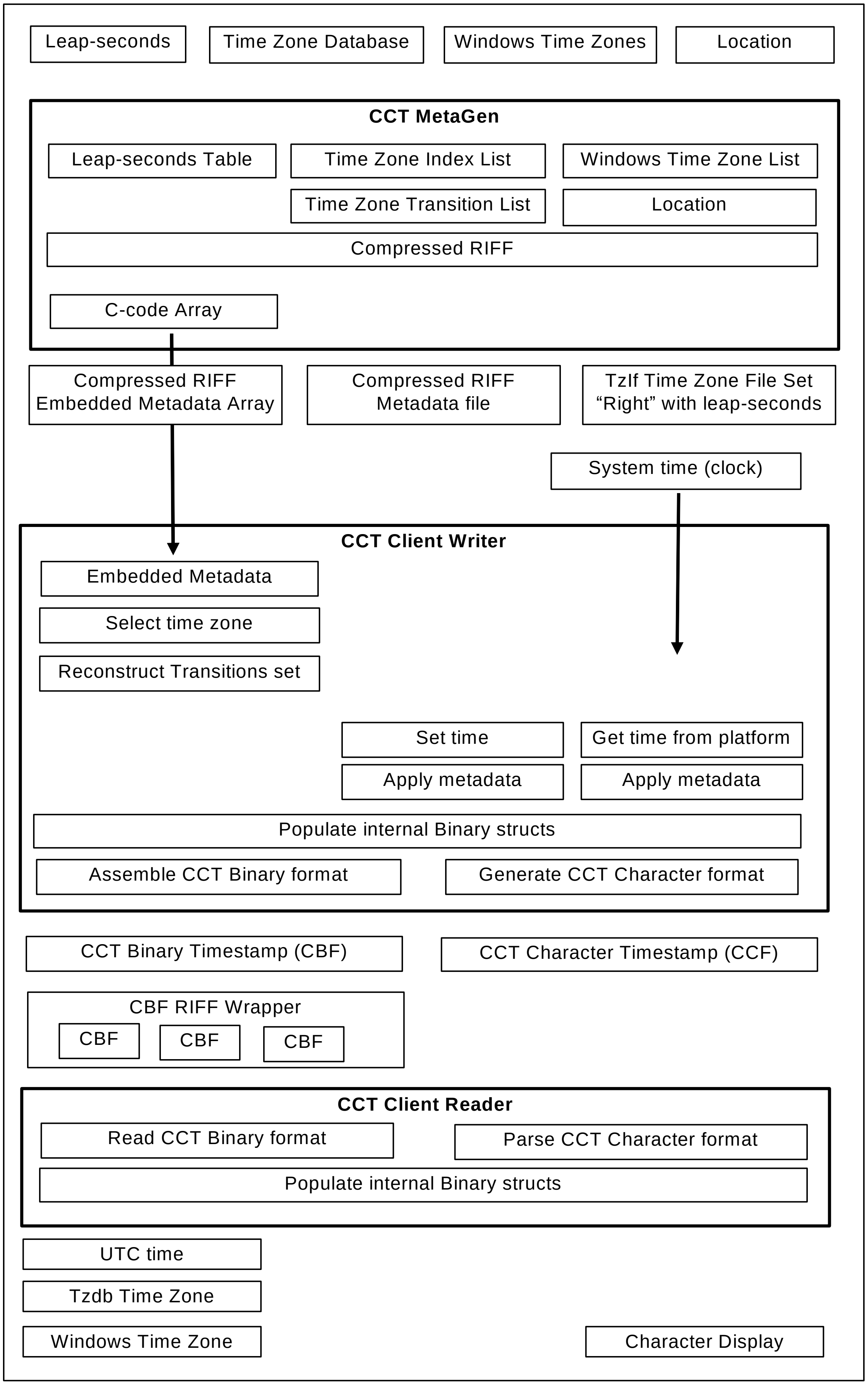
1 Introduction

The set of Common Calendar specifications describe the data types formats and APIs that make up the Common Calendar Timestamp system. This document describes the c/c++ reference implementation.

There are two major parts of the CCT implementation; CCTMetaGen and CCTDemo.

CCTMetaGen obtains external leap-seconds, Tz Database time zones and rules, location and country codes, and Windows time zones and consolidates them into a compact format for distribution to CCT client applications.

CCTDemo is a CCT client demonstrating use of the CCT APIs including incorporation of the metadata created by CCTMetaGen, instantiation of CCT timestamp objects, and exercising the CCT APIs including generating completed timestamps. CCTDemo implements both a CCT Client Writer and a CCT Client Reader.


Leap-seconds
Time Zone Database
Windows Time Zones
Location
CCT MetaGen
Leap-seconds Table
Time Zone Index List
Windows Time Zone List
Time Zone Transition List
Location
Compressed RIFF
C-code Array
Compressed RIFF Embedded Metadata Array
Compressed RIFF Metadata file
TzIf Time Zone File Set "Right" with leap-seconds
System time (clock)
CCT Client Writer
Embedded Metadata
Select time zone
Reconstruct Transitions set
Set time
Get time from platform
Apply metadata
Apply metadata
Populate internal Binary structs
Assemble CCT Binary format
Generate CCT Character format
CCT Binary Timestamp (CBF)
CCT Character Timestamp (CCF)
CBF RIFF Wrapper
CBF
CBF
CBF
CCT Client Reader
Read CCT Binary format
Parse CCT Character format
Populate internal Binary structs
UTC time
Tzdb Time Zone
Windows Time Zone
Character Display

2 Incorporation of Tz Database and glibc

The CCT implementation incorporates adapted versions of Tz Database c-code for functionality provided by zic.c and zdump.c, and relevant portions of glibc c-code. This is done for several reasons.

- To include all the relevant code in a single library to avoid any dependence on operating system services or installed library versions on the platform.
- To provide a timekeeping system that functions completely independent of the operating system.
- To allow the c/c++ optimizers to operate across the libraries and applications.
- To include critical additions to Tz Database and glibc to support the objectives of the CCT system and formats. In particular the addition of an STDOFF member variable to standard version of `struct tm`.

This results in an enhanced implementation of Posix time, glibc and TzDb that is completely independent of the operating system.

2.1 Tz Database

IANA Time Zone Database distributes the time zone data sources and associated c-code.

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

The TzDb zic.c and zdump.c programs are important. They are really the reference implementations of how TzDb should operate and it is challenging to replicate their behavior in all details. CCT has adapted them directly to retain their timekeeping characteristics.

It is difficult to use zic.c and zdump.c in Windows builds, in particular to avoid conflicts with Windows c/c++ header files. To avoid this the source code has been ported to compile directly on Windows c/c++ using MSVC.

The Tz Database code, both zic.c and zdump.c, are employed in both CCTMetaGen to assemble rule sets for distribution, and in clients to reconstruct the full transition set for the selected time zone.

2.2 Posix Time and glibc

The TzDb code relies on the target platform's implementation of Posix time as implemented in relevant portions of glibc.

GNU C Library master sources
https://sourceware.org/git/?p=glibc.git;a=summary

To support the independence of the CCT code from the platform the CCT implementation includes these portions of glibc compiled into the CCT libraries. This avoids conflicts with the platform's Posix time.

It also enables the modification of a critical component of Posix time, the declaration and use of `struct tm`.

2.2.1 Enhanced `struct tm`

The normal version of Posix time's `struct tm` has insufficient information to accomplish the objectives of CCT. In particular it lacks a member variable for STDOFF. This data is required to distinguish GMTOFF from STDOFF and propagate the STDOFF values through the CCTMetaGen processes and to the client.

CCT has renamed `struct tm` to `struct tztm` and added a member, `long int tm_stdoff`.

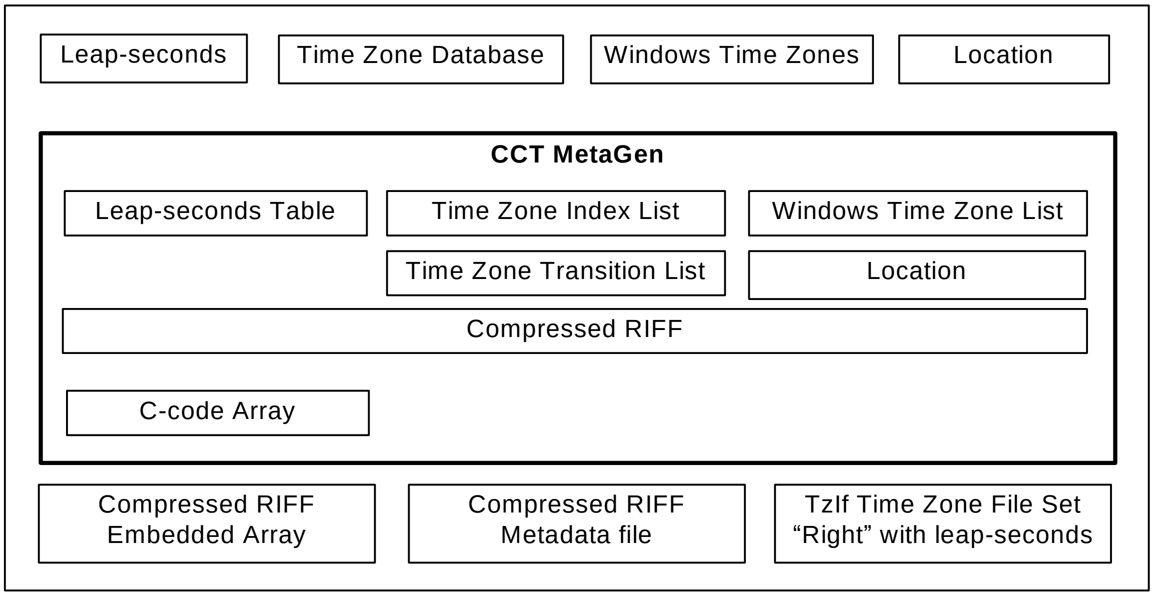
```
struct tztm
{
  int tm_sec;    /* Seconds. [0-60] (1 leap second) */
  int tm_min;    /* Minutes. [0-59] */
  int tm_hour;   /* Hours. [0-23] */
  int tm_mday;   /* Day. [1-31] */
  int tm_mon;    /* Month. [0-11] */
  int tm_year;   /* Year- 1900.  */
  int tm_wday;   /* Day of week. [0-6] */
  int tm_yday;   /* Days in year.[0-365] */
  int tm_isdst;  /* DST. [-1/0/1]*/
  long int tm_gmtoff;  /* Seconds east of UTC.  */
  const char *tm_zone; /* Timezone abbreviation.  */
  long int tm_stdoff;  // CCT addition
};
```

This modification does not alter the behavior of Posix time calls like localtime() but it allows the CCT implementations to properly account for STDOFF shifts together with GMTOFF adjustments.

3 CCTMetaGen Metadata Generator

CCTMetaGen is the central application that generates the metadata for CCT clients. It is analogous to TzDb zic.c; it parses the TzDb source files together with leap-seconds and Windows time zones and generates a binary form of the metadata for use by clients.

CCTMetaGen obtains external information and generates metadata for use by a CCT Client.



CCTMetaGen needs paths to the metadata files and paths to several output file locations.
class CCctPaths::ConstructPaths() provides methods for managing the required directories and file names.

3.1 External Information

CCTMetaGen must first acquire external information from several sources; leap-seconds, Tz Database time zones, default time zone locations, and Windows time zones.

The CCTMetaGen needs paths to external metadata and direct output. Class CCctPaths is provided to construct appropriate paths. See CCctPaths::ConstructPaths().

3.1.1 IERS Leap-seconds

CCTMetagen obtains leap-second information directly from:

IERS Leap_Second_History.dat
https://hpiers.obspm.fr/eoppc/bul/bulc/Leap_Second_History.dat

CCTMetagen parses and processes the Leap_Second_History.dat file through the Common Calendar TAI-UTC API. This populates a double-linked list with struct DateTaiUtc_st data in class CTaiUtcTable.

See Common Calendar TAI-UTC API.

3.1.2 IANA Time Zone Database Time Zones and Rules

CCTMetagen obtains time zone information from:

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

CCTMetagen parses and processes all the TzDb source files through Common Calendar Time Zone Database API, including etcetera, backzone, and zone.tab for default time zone location and country code. TzDb processing is “vanguard” (supporting negative DST) and “Right” (with leap-seconds).

This employs TzDb zic.c and zdump.c code directly to parse the TzDb source files and generate transitions. This populates a set of double-linked lists of transitions CTzTransList.

3.1.2.1 Time Zone Indexing

CCT uses an indexing procedure to provide a compact representation of TzDb time zone identifiers.

The TzDb time zone identifiers, such as "America/New_York", are indexed to provide a compact time zone identifier suited for the CCT binary format and to provide convenient time zone selection methods within CCT clients. When CCTMetaGen has parsed all the TzDb source files it iterates through the time zones in order they appear in the source files and simply assigns an incrementing index integer to each.

This is implemented in CTzDataParse::AssignInitialFixedZoneIndices(); and produces a list, CTZDZoneNameIdxList, of TzDb time zone names and their assigned indexes in data type struct ZoneNameAndIdx_st;

3.1.3 Windows CLDR Time Zones

CCTMetaGen obtains Windows time zone information from:

Unicode CLDR Windows time zones
https://github.com/unicode-rg/cldr/blob/main/common/supplemental/windowsZones.xml

Windows time zones are obtained from the Unicode Windows CLDR, indexed, and mapped to the TzDb time zones. CTzDataParse::MergeWinZonesWithCTZDZones().

3.2 Time Zone Transitions

CCTMeteGen runs zic internally to create the full set of transitions for each time zone. This generates a set of arrays in native zic form internally in memory as output by zic outzone(). A full TzIF file set is also generated by calling zic writezone(); These are “Right” (with leap-seconds).

The TzDb transitions of each time zone are converted to a double-linked list, class CTzTransList, with data type struct TZDTrans_st. This list is filtered to retain only the primary rule set as a double-linked list, with data type struct TZDTransTr_st, This represents the minimal information required to reduce transfer size.

3.3 UTC Offset Shift

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

The presence of a CBFUtcShift_st is flagged by `CBFLocalDate_st::m_bUtcShiftExt:1`

See Common Calendar Binary Format, UTC Offset Shift - `CBFUtcShift_st`

3.4 Location

The Common Calendar Timestamp (CCT) specification has been extended to include geographic coordinates to create a Geostamp.

See Common Calendar Geostamp.

3.5 Compressed RIFF for Distribution and Embedding

CCTMetaGen has assembled lists containing the required metadata for CCT Client operations.

- CTaiUtcTable - Leap-seconds
- CTzTransList - Tz Database transitions
- CTZDZoneNameIdxList - time zone indexes
- CWinZones - Windows time zones
- CBFLocation_st - country codes and default geographic coordinates

This set of lists are reformatted into a RIFF (four cc) container for distribution to CCT clients. The RIFF is zipped (zlib.c), resulting in a compressed binary representation of all needed metadata. This results in a file size about 150kb. (This compares to the TzDb tzdata tarball of about 430kb.)

The byte values of the compressed RIFF are processed through class CMetaEmbedder::GenEmbedderCode() to generate a c-code file, CCTMetaEmbedded.cpp in CCTLib2, that declares and populates an array containing the compressed RIFF. This array may be embedded (compiled and linked) in a CCT client application.

CCTMetaGen will also write the full set of native TzDb TzIF source files. This is useful for testing because it allows native versions if TzDb zdump to generate transitions for comparison to the CCT timestamps, as shown in some test routines in CCTDemo. It also potentially allows other systems to read the CCT generated TzIF files, which include the complete set of time zones in "vanguard" and “right” (with leap-seconds) form.

4 CCT Clients

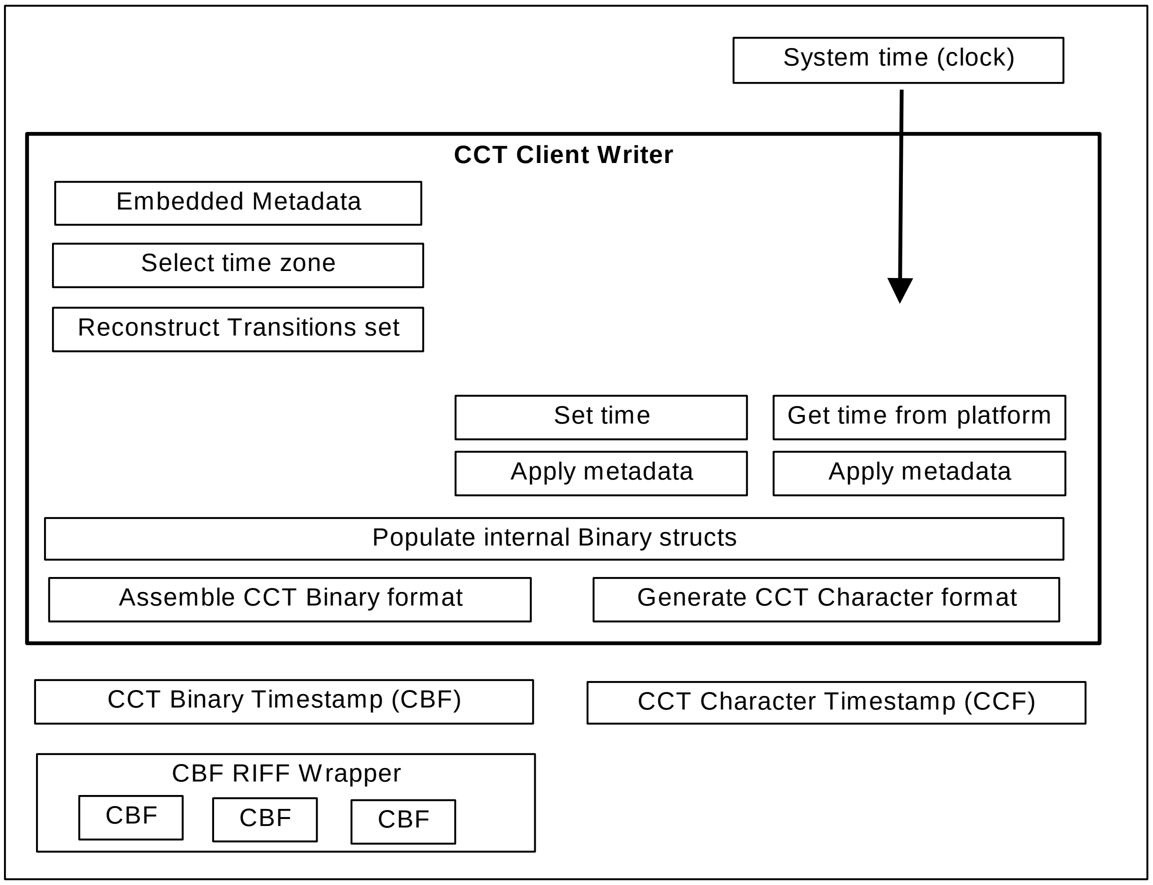
CCT clients are the operational versions of Common Calendar for use by applications, operating systems, anywhere unambiguous timestamps are needed.

The compressed metadata produced by CCTMetaGen can be embedded in client executables on build by including CCTLib2. At runtime the client calls CRiff::MOpen() which calls a static function find_embedded_file() to locate the compressed image and decompresses it to memory. Thus all the metadata is available to the client application as needed.

Construction of, access to, and manipulation of the CCbf and CCcf classes and accompanying functionality are provided by Common Calendar Timestamp API. See Common Calendar Timestamp API. The methods are implemented in class CCct,

4.1 CCT Client Writer

A CCT writer must have access to all applicable metadata to fully populate the internal class CCbf members. The metadata array code generated by CCTMetaGen, CCTMetaEmbedded.cpp, is compiled and linked into the client at compile time.


System time (clock)
CCT Client Writer
Embedded Metadata
Select time zone
Reconstruct Transitions set
Set time
Get time from platform
Apply metadata
Apply metadata
Populate internal Binary structs
Assemble CCT Binary format
Generate CCT Character format
CCT Binary Timestamp (CBF)
CCT Character Timestamp (CCF)
CBF RIFF Wrapper
CBF
CBF
CBF

At runtime, on construction of CCct, this array is read and the recovered RIFF is decompressed to memory. The CTaiUtcTable, CTzTransList, CTZDZoneNameIdxList, CWinZones, and CBFLocation_st metadata are reconstructed.

When the user selects a time zone the CTzTransList is converted back to TzDb native form. Zic is run to reconstruct the native TzDb metadata for that time zone, and zdump is run to generate all the TzDb transitions. The TzDb transitions are converted to a fully populated CTzTransList of the time zone for use by the client.

Windows time zones and CBFLocation_st data are connected to the selected time zone.

The user must supply the desired parameters to control the various options provided by CCT.

See CCct.h - CCTParams_st and CCct::m_CCTParams_st

4.1.1 CCT Binary Format (CBF)

A writer creates the CCT Binary Format (CBF).

class CCbf is populated by calling one of the "CCct::Set" methods.

`CCct::SetSecsFrac()`

If used, `CBFLocation_st` is populated with default location information.

An application can set the CCT location to an actual GPS location using:

`int CCct::SetLocation(char* psLatitude, char* psLongitude, char* psAltitude)`

See Common Calendar Geostamp.

The final operation is to assemble the completed binary CBF from the populated internal CCbf members.

`CCct::AssembleCbf()`

This results in the final result of the Common Calendar - the binary CBF timestamp itself.

This listing shows the CCF character format of a selected example, followed by the values of the internal CCbf struct members, followed by a hex dump of the assembled CBF.

```
D2024-03-10T01:59:59m000U-05Zamerica/new_yorkAestV2021aL27S00t01a02cMuX
// UTC1970 1710054026.000 Day 19792

CCbf contents:
CBFTime_st::
 m_eRateEnumeration CLOCK_3
 m_bLocalDateExt    TRUE
 m_b24HourPeriodExt  FALSE
 m_bCounterSign     positive
 m_eCounterSize     COUNTERSIZE_35
 m_ulCounterLow32   7199000
CBFTime_st Counter  7199000
CBFLocalDate_st::
 m_l1970DayNumber    19792
 m_nLeapsecs         27
 m_TZDTimeZoneID_st.m_unZoneIdx    idx[255] America/New_York
 m_TZDTimeZoneID_st.m_unTzDataReleaseYear   49
 m_TZDTimeZoneID_st.m_unTzDataReleaseLetter 0
 m_TZDTimeZoneID_st.m_bCBFLocationExt       FALSE
 m_TZDTimeZoneID_st.m_bCBFAbbrExt           TRUE
 m_TZDTimeZoneID_st.m_bCBFAbbrChangeExt     FALSE
 m_lUTCOffset         -18000
 m_eTODMode           TOD_LEAPSECOND_UTC_UTC
 m_eDSTCountMode      DSTCOUNTMODE_CONVENTIONAL
 m_bIsLeapSecondDay   FALSE
 m_bIsLeapSecond      FALSE
 m_bIsLeapSecondNegative   FALSE
 m_bUtcShiftExt       FALSE
 m_bDSTExt            TRUE
 m_bDstTransDayExt    TRUE
CBFAbbr::
 est
CBFDst_st::
 m_eDSTBias         0
CBFDstTransDay_st::
 m_ulDSTTransTime   7200
 m_lDSTBiasChange   3600
CBF Total size      29 bytes  232 bits

Assemble binary CBF image from CCbf data
 04  01  18 -d9  6d  00  00  00  50  4d  00  00 -80  0d  00 -ff
 00  31  10 -b0 -b9 -d5  00  00  00  00  20  1c  10  0e  00 -e5
-f3  74
 size 34
```

4.1.2 CBF RIFF Wrapper

The binary CFB is the prime interchange item which may be used unadorned in many circumstances. In some situations it may be convenient to encapsulate the CBF in a known wrapper format to isolate it from some container or transport protocol or to concatenate a list of CBFs. CCT provides the CCT CBF RIFF Wrapper for these purposes.

One or more CBFs may be wrapped in a CBF RIFF Wrapper using:

`CCct::WriteCRiff_Ccbf()`

See Common Calendar Binary Format, CBF RIFF Wrapper.

4.1.3 CCT Character Format (CCF)

The writer can also output the CCT Character Format (CCF). A CCF is constructed from the populated CCbf data.

Construct a CCF string from the CCbf data

`CCct::SetCcfFromCCbf()`

Get the CCF string

`CCct::GetCcfAscii(char* sCcf)`

Example showing the CCF formed from the CCbf member values shown above in the 4.1.1 CCT Binary Format (CBF) section.

`D2020-03-08T01:59:59n000000000U-05Zamerica/new_yorkAestV2021aL27S00t01a02cMuX`

See Common Calendar Character Format.

4.2 CCT Client Reader

The CCT framework is all in service for the Client Reader, allowing applications to read the timestamps and derive the information required for their intended purpose.

- CCTMetaGen has generated metadata used by the Client Writer.
- The Client Writer has produced timestamps.
- The Client Reader can then read CBFs and CCFs and supply information to the application as required.

CCT Readers may be constructed in two forms

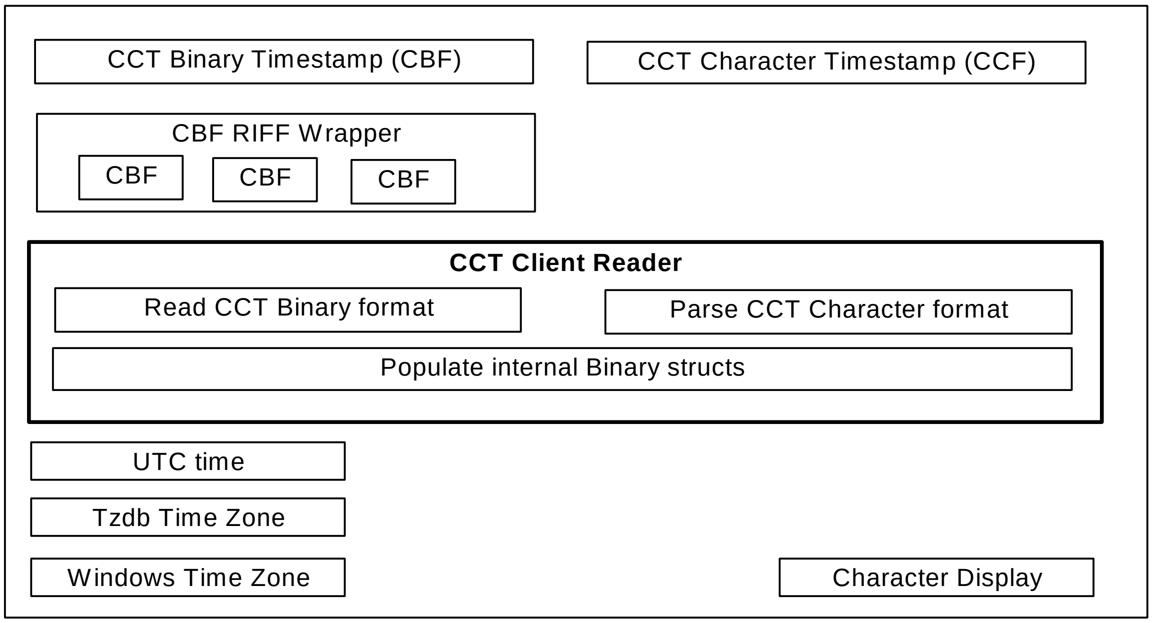
Isolated and memory limited applications

CCT timestamps are designed to provide sufficient metadata to calculate time points and YMDhms representation across the current day to support readers and applications that may have no access to external information and insufficient memory to store all the metadata. A reader of this type can support UTC time and local YMDhms for the current day; it cannot reliably address past and future dates.

Connected and with memory applications

Applications with connectivity and sufficient memory can implement a fully functional system that can calculate past and future dates.

Construction of, access to, and manipulation of the CCbf and CCcf classes and accompanying functionality are provided by Common Calendar Timestamp API. The methods are implemented in class CCct.



4.2.1 Read CBF Binary Format

A CBF can be read to memory, parsed, and populates the set of internal CCbf classes:
`CCct::ReadCbf()`

One or more CBFs may reside in a CBF RIFF Wrapper and can be read and the individual CBFs extracted:
`CCct::ReadCRiff_Ccbf()`

4.2.2 Read CCF Character Format

Read and parse a CCF string and populate the set of internal CCbf classes:
`CCct::ParseCcfSetCCbf(char* sCcf)`

Read the internal CCbf classes and generate the internal CCF string:
`CCct::SetCcfFromCCbf()`

Get the CCF string:
`CCct::GetCcfAscii(char* sCcf)`

Example:
`D1972-06-30T19:59:60U-04Zamerica/new_yorkAedtV2021aL0*Ss+01cMuX`

5 CCTDemoConsole

CCTDemoConsole is a console application that demonstrates CCT use. It includes both a CCT Client Writer and a CCT Client Reader. It also includes access to the isolated TzDb and Posix time functions so that testing can compare the CCT results to expected results from tzbd.

5.1 Construction and Initialization

CCTDemo shows how to construct and use class CCct, the user-level interface to CCT as documented in Common Calendar Timestamp_API.

Instantiation of CCct is shown in CCTDemosConsole's main();

A convenience helper, class Test_st Test_stX, is used to pass objects and parameters to the demo and test routines. See CCTDemoTests.h.

The CCTDemo application needs paths to direct output listings. Class CCctPaths is provided to construct appropriate paths. See CCctPaths::ConstructPaths().

A "test" CCct is instantiated and passed by Test_stX to many of the test and demo routines.

5.1.1 CCct Instantiation

Class CCct is the central object of the CCT client implements. The constructor performs many initialization operations. The required metadata is contained in the embedded compressed RIFF array produced by CCTMetaGen. On construction it is decompressed and parsed to populate the internal metadata lists.

- CTaiUtcTable - Leap-seconds
- CTzTransList - Tz Database transitions
- CTZDZoneNameIdxList - time zone indexes
- CWinZones - Windows time zones
- CBFLocation_st - country codes and default geographic coordinates

5.2 CCTDemoConsole - Tests and Demos

Several demonstrations and tests are implemented in the CCTDemoConsole application.

The listing can be viewed at:
CCTDemoConsole Tests Listing Guide