Common Calendar YMDhms API

Seconds to YMDhms Conversion - Data and Algorithms

Brooks Harris Version 2 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).

1 Introduction

The main objective of any civil timekeeping system is to produce date and time in the familiar year, month, day, hour, minute, second (YMDhms) form suitable for human understanding. Systems typically operate internally on a seconds and decimal fractions of seconds basis and conversion algorithms between seconds and YMDhms values are fundamental to most implementations.

Many systems adopt the classic operations, or functions, described by Posix and named mktime() (YMDhms-to-seconds) and gmtime() (seconds-to-YMDhms). These execute the Gregorian calendar counting method, including its Leap Years, making calculations based on 86400 second day durations. POSIX has recognized limitations in supporting UTC and leap-seconds, in particular it cannot count to 23:59:60 and has no mechanism to obtain and apply the dynamic TAI-UTC leap-second values.

To overcome this deficiency Common Calendar specifies the YMDhms API. It specifies data types and operations to perform the conversion between seconds and YMDhms with leap-second compensation using the TAI-UTC API for access to the required dynamic TAI-UTC leap-seconds information.

The YMDhms API specifies data types analogous to Posix time_t ("seconds since the Epoch") and struct tm ("broken down time"), called i641970Time_t (signed 64-bit seconds since UTC1970) and YMDhms_tm (YMDhms, leap-seconds and additional metadata). See *5 Annex CCT struct YMDhms_tm Declaration.*

YMDhms API describes functions analogous to Posiz mktime() and gmtime() calling them YMDhmsToSeconds() and SecondsToYMDhms(). CCT act of the same orign as the Posix formulas: 1970-01-01 00:00:10 (TAI) = 1970-01-01T00:00:00 (UTC). (called UTC1970).

YMDhms_tm includes additional information to Posix struct tm, including leap-second values, and directly supports leap-second compensation including count to 23:59:60 for positive leap-seconds and 23:59:58 roll-over for negative leap-seconds.

The CCT c/c++ reference implementation includes the explicit algorithmic operations. See CCT\CCTYMDhmsApiLib.

The Common Calendar specifications are intended to be used as generic APIs suitable for implementation on many platforms. The specifications are written largely in terms of c/c++, adopting the c/c++ language syntax for data types and methods to avoid potential ambiguity. CCT "(or "Common Calendar Timescales") is a c/c++ reference implementation of the Common Calendar specifications. The

specifications reference the CCT code directly to show unambiguous logic and facilitate implementation. Please see accompanying source code file "CCT.zip".

2 Scope

Specifies 64-bit seconds and comprehensive YMDhms data types and algorithms to provide UTC leap-second accurate conversion between seconds-since-UTC1970 and YMDhms representation.

3 Normative References

BIPM The International System of Units (SI) 8th edition 2006 (commonly called the SI Brochure)
http://www.bipm.org/en/publications/si-brochure/

BIPM JCGM 200:2012, International vocabulary of metrology – Basic and general concepts and associated terms (VIM)
http://www.bipm.org/en/publications/guides/vim.html

IERS Conventions (2010), (IERS Technical Note No. 36)
https://www.iers.org/IERS/EN/Publications/TechnicalNotes/tn36.html?nn=94912

Recommendation ITU-R TF.457-2, Use Of The Modified Julian Date By The Standard-Frequency And Time-Signal Services
http://www.itu.int/rec/R-REC-TF.457-2-199710-I

Recommendation ITU-R TF.460-6 (02/02), Standard-Frequency and Time-Signal Emissions
http://www.itu.int/rec/R-REC-TF.460/en

Common Calendar Date and Time Terms and Definitions
https://common-calendar.org/Common_Calendar_Specification/Common_Calendar_Introduction_and_Scope_V11_2022_12_29.pdf

Common Calendar TAI-UTC API
https://common-calendar.org/Common_Calendar_Specification/Common_Calendar_TAI-UTC_API_V3_2024_04_25.pdfl

4 YMDhms API

The TAI timescale is an uninterrupted ascending count of seconds of uniform frequency based on atomic clocks. The UTC timescale is an encoding of TAI as calendar date and time (YMDhms) of the mean solar day used as the basis of civil time. TAI-UTC, the integral second difference between TAI and UTC, is necessary to encode seconds to YMDhms or decode YMDhms to seconds. The functions to accomplish these conversions correctly are vital to accurate timekeeping.

The TAI-UTC API specifies data types, how leap-seconds tables are constructed and populated, operations for disseminating and receiving TAI-UTC data, and methods for leap-second table look-up.

The YMDhms API specifies data types and operations to perform the conversion between seconds and YMDhms representation with leap-second compensation using the TAI-UTC API.

The CCT c/c++ reference implementation class CYMDhmsApi implements the required functionality of the YMDhms API. It includes functions for leap-second compensated seconds-to-YMDhms and YMDhms-to-seconds conversions and related methods. CYMDhmsApi relies on a leap-second table as defined by the TAI-UTC API. Member CYMDhmsApi::m_pCTaiUtcTable is a complete TAI-UTC (leap-second) table as constructed by an instance of class CTaiUtcClient.

The origin of both the TAI-UTC API and the YMDhms API is 1970-01-01T00:00:00 (UTC), coincident with Posix "the Epoch" and its use by IANI Time Zone Database (tzdb).

1970-01-01 00:00:10 (TAI) = 1970-01-01T00:00:00 (UTC).

Class CSeconds1970ToYMDhms implements Seconds1970ToYMDhms() and YMDhmsToSeconds1970(). These methods operate on 86400 days with no leap-second adjustment and are used as core components of CYMDhmsApi functions. These are similar to the classic Posix gmtime() and mktime() functions. The Seconds1970ToYMDhms() method employs an adapted version of Linux/glibc __offtime() function directly. They are implemented separately because the 86400-second-day

Posix-like functions may have utility independent of the leap-second compensated YMDhms API methods.

Class CSeconds1970ToYMDhms is public to class CYMDhmsApi making its functionality available to CYMDhmsApi.

YMDhmsApi.h declares data types i641970Time_t and YMDhms_tm as primary input and output data types for CYMDhmsApi. These are analogous to classic Posix time_t and struct tm but with modifications to support leap-seconds and other important metadata.

Class CYMDhmsApi member variables CYMDhmsApi::m_i641970Time_t and CYMDhmsApi::m_YMDhms_tm provide local variables on which the methods act. Methods such as CYMDhmsApi::SecondsToYMDhms(), CYMDhmsApi::YMDhmsToSeconds(), and CYMDhmsApi::Get1970DayNumberBy1970Seconds() act on these local variables.

4.1 Conversion with Leap-second Compensation

class CYMDhmsApi implements conversion between YMDhms and seconds with leap-second compensation. Its methods operate on a UTC1970 origin.

Access to leap-second metadata as provided by the TAI-UTC API is required and supplied by the reference implementation.

4.1.1 Seconds1970ToYMDhms

Convert UTC1970Seconds to YMDhms_tm with leap-second compensation.

```
YMDhms_tm* CYMDhmsApi::SecondsToYMDhms();
```

4.1.2 YMDhmsToSeconds1970

Convert YMDhms_tm to UTC1970Seconds with leap-second compensation

```
int64_t CYMDhmsApi::YMDhmsToSeconds();
```

4.2 Conversion *without* Leap-second Compensation

class CSeconds1970ToYMDhms implements conversion between YMDhms and seconds *without* leap-second compensation. Its methods operate on a UTC1970 origin.

In some circumstances it is useful to calculate YMDhms and second conversion without leap-seconds compensation, and the methods support that use. These methods also provide the core YMDhms and seconds conversion algorithms utilized by the leap-second compensated class CYMDhmsApi methods.

These methods do not require leap-second metadata.

4.2.1 Seconds1970ToYMDhms

UTC1970 seconds to YMDhms_st *without* leap-second compensation, similar to Posix gmtime().

```
YMDhms_tm* CSeconds1970ToYMDhms::Seconds1970ToYMDhms(int64_t* pi64Secs1970, YMDhms_tm* pYMDhms_tm);
```

4.2.2 YMDhmsToSeconds1900

YMDhms_st to UTC1970 seconds without leap-second compensation. similar to Posix mktime().

```
int64_t CSeconds1970ToYMDhms::YMDhmsToSeconds1970(YMDhms_tm* pYMDhms_tm, int64_t* pi64Secs1970);
```

5 Annex CCT struct YMDhms_tm

Excerpt from YMDhmsApi.h

```
//////////////////////////////////////////////////////
// i641970Time_t uninterrupted incrementing zero-based count of integral seconds
// since UTCT1970 = 1970-01-01T00:00:00 (UTC) = 1970-01-01 00:00:10 (TAI)
// signed 64-bit, // #define _I64_MAX  922337203685477580 7i64
// NOTE this is integral seconds, NOT seconds and some decimal fraction of seconds
// typedef int64_t i641972Time_t; // TaiUtcApi.h
//////////////////////////////////////////////////////

typedef struct YMDhms_tm
{
  int m_iSecond;      // Seconds after the minute
                      // zero based count [0 - 60]
                      // range at least 61 (2^6 = 64 MAX)
                      // supports positive leap-second (23:59:60)
  int m_iMinute;      // Minutes after the hour
                      // zero based count [0 - 59]
                      // range at least 60 (2^6 = 64 MAX)
  int m_iHour;        // Hours since midnight
                      // zero based count [0 - 24]
                      // supports DST 23 and 25 hour day
                      // range at least 25 (2^5 = 32 MAX)
  int m_iDay;         // Day of the month
                      // one based count [1 - 28, 29, 30, 31]
                      // range at least 31 (2^5 = 32 MAX)
  int m_iMonth;       // Months since January
                      // one based count [1 - 12]
                      // range at least 12 (2^4 = 16 MAX)
  int m_iYear;        // Year
                      // zero based count [0000-9999]
                      // range at least 10000 (2^14 = 16384 MAX)
  int m_iDayOfWeek;   // Day of week since Sunday
                      // zero based count [0 - 6], Sunday = 0
                      // range at least 7 (2^3 = 8 MAX)
 int m_iDaysInMonth; // [28 or 29 or 30 or 31]
                      // range at least 31 (2^5 = 32 MAX)
  int m_iDayOfYear;   // Days since January 1 [0 - 365, 366 MAX] 365 = 366th day
                      // zero based count
                      // range at least 367 (2^8 = 256 MAX)
 int m_iDaysSince1970;  // Day number since 1970-01-01T00:00:00 (UTC) [0 - n]
                        // zero based count
                        // range at least (3100 years * 366) = 1134600
                        // (2^21 = 2097152 MAX)
 int m_bIsLeapYear;     // Year is a leap year
                        // [0 false or 1 true]
 int m_iLeapYearsSince1970;  // Leap years since 1970-01-01T00:00:00 (UTC) [0 - n]
                             // zero based count
                             // range at least 367 (2^8 = 256 MAX)
 int m_iLeapsecs;   // Leap-seconds - TAI-UTC minus initial 10s calibration offset
                    // at UTCT1970
                    // range 1023 (2^10 = 1023 MAX)
 int m_bIsLeapSecondDay;  // A leap-second will occur at end of this day
                          // [0 false or 1 true]
 int m_bIsLeapSecondNegative; // if today is a leap-second day the leap-second
                              // is negative
                              // [0 false or 1 true]
 int m_bIsLeapSecond;         // This second is leap-second
                             // [0 false or 1 true]
} YMDhms_tm;
```