Common Calendar Timestamp API

User level interface and manipulation functionality

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).

1 Introduction

Common Calendar as a whole provides specifications, data formats, and manipulation algorithms for deterministic UTC accurate local timekeeping. Common Calendar Timestamp API provides convenient user-level access to the Common Calendar capabilities.

The Common Calendar specifications are intended to be interpreted 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 cite the CCT code directly to show unambiguous logic and facilitate implementation.

2 Scope

Specifies user level functions for populating and reading Common Calendar Binary Format (CBF) and Common Calendar Character Format (CCF) together with general manipulation operations required for timekeeping according to the Common Calendar specifications.

3 Normative References

Common Calendar Date and Time Terms and Definitions

Common Calendar TAI-UTC API

Common Calendar Tz Database API

Common Calendar YMDhms API

Common Calendar Local Timescales

Common Calendar Binary Format

Common Calendar Character Format

4 Common Calendar Timestamp API

Common Calendar Timestamp API supplies user-level functionality to a Common Calendar implementation.

Common Calendar specifies two timestamp formats, Common Calendar Binary Format (CBF) and Common Calendar Character Format (CCF), that are symmetrically convertible to one another.

class CCct contains instances of classes CCbf and CCcf together with methods for populating and manipulating CCbf binary values, and initiating construction and parsing of CCcf character strings.

Many CCct methods are convenience wrappers around the underlying CCbf and CCcf methods, while others are more elaborate, providing commonly used manipulation and user-level input reinforcement and error handling. Several static utility functions, with "UTIL" in the name, are provided.

See Common Calendar Binary Format
See Common Calendar Character Format

4.1 Common Calendar Timestamp API Parameters

CCTParams_st facilitates passing values and user selection parameters to the CCct methods.

Date type struct SecondsFrac_st, declared in CCbf.h, is a fixed point data type of seconds and decimal fractions of seconds, the resolution given by the "rate".

See CCT\CCTLib\CCbt.h, `typedef struct SecondsFrac_st`
See CCT\CCTLib\CCct.h, `typedef struct CCTParams_st`

CCT Timestamp API Parameters

parameter variable	type	description	reference
Seconds	int64	seconds	SecondsFrac_st::m_i64Seconds
ui64Frac	uint64	decimal fraction of seconds	SecondsFrac_st::m_ui64Frac
rate	enum	resolution	SecondsFrac_st::m_CBFRate_et
TZDZoneName	enum	time zone name	TzDatabaseApi.h TZDZoneName_et
TodCountMode	enum	Time-of-day Count Mode	CBF.h typedef enum CBFTodCountMode_et See Common Calendar Local Timescales: 4.2Time-of-day (TOD) Count Mode and Leap-second Introduction
DstCountMode	enum	DST Count Mode	CBF.h typedef enum CBFDstCountMode_et See Common Calendar Local Timescales: 4.1Daylight Savings Time (DST) Count Mode
bUserIncludeDstSchedule	bool	not fully implemented	
bUserIncludeTAIUTCInterval	bool	not fully implemented	
DstRetreatHmsSpan_et	enum	1st or 2nd occurrence of DST Retreat hms	CCct.h typedef enum DstRetreatHmsSpan_et

4.2 Common Calendar Binary Format (CBF) Operations

A CBF and its corresponding CCF format can have one of six meanings:

- time point with UTC accurate local date and time-of-day
- time point with UTC accurate local date with time portion having no relation to the date
- time point less than 24 hours with no relation to date (< 86400 seconds)
- time point 24 hours or greater with no relation to date (>= 86400 seconds)
- time interval less than 24 hours (< 86400 seconds)
- time interval 24 hours or greater (>= 86400 seconds)

The CBF TimeStamp API operations provide means of populating CBF instances for the supported purposes and their manipulating values and metadata. All timekeeping operations act on the binary CBF while the YMDhms form is reflected by a corresponding CCF.

See Common Calendar Binary Format
See Common Calendar Character Format

4.2.1 SetYMDhmsd

Populate a CBF as UTC accurate local date and time-of-day from YMDhmsd user input values and parameters with input value validation.

Input YMDhms values must be valid local time, including DST if applicable. Accepts YMD hh:mm:60.dd Leap-second designations.

Valid local YMDhmsd input values are enforced by detection of the local zone Leap-second and DST rules.

SetYMDhmsd() discovers the Leap-second and DST rules and values for input YMDhmsd values and adjusts the 1970Seconds values accordingly before finally calling CCbf::SetFrom1970SecondsFrac() to set the CCbf counter values which always represent NORMAL time on the local timescale. It also sets local DST metadata in CCbf and this is used to command construction of the Ccf character YMDhms representation later when CCcf::SetCcfFromCCbf() is called. In other words, binary counter values are always normal time of the zone and never represent DST time. DST YMDhms representation is applied to the YMDhms character output only.

For example, in zones using DST, on a DST change date, with a DST bias of 01:00 and a DST onset of 2AM the hmsd, the function will not accept YMD 02:00:00.0 because that value did not exist (it jumped to 03:00:00.0). It has similar protection for Leap-second introduction in local YMDhmsd values.

In zones observing DST, on DST Retreat days the hms values will repeat for the DST bias interval. For example, for a (typical) 1 hour bias, 01:00:00 to 01:59:59 will repeat - once for the DST time (summer time) and once after the DST Retreat shift (winter time). There is no way to automatically detect if the hms input values in this range are intended to be the first or second occurrence of the DST Retreat sequence. For this reason SetYMDhms() has the mandatory DstRetreatHmsSpan_et parameter to allow the user to chose between DSTRETREAT_HMSSPAN_DSTTIME (DST time (summer), Retreat has not occurred) and DSTRETREAT_HMSSPAN_NORMALTIME (Normal time (winter), Retreat has occurred).

From CCct.h
```
typedef enum DstRetreatHmsSpan_et
{
DSTRETREAT_HMSSPAN_DSTTIME, // DST time, summer time, first
                            // occurrence of Retreat hms sequence
DSTRETREAT_HMSSPAN_NORMALTIME // Normal time, winter time, second
                            // occurrence of Retreat hms sequence
} DstRetreatHmsSpan_et;
```

See `CCT\CCTLib\CCct.h`
```
int CCct::SetYMDhmsd(int iYYYY, int iMM, int iDD, int ihh, int imm, int iss,
                               uint64_t ui64dd, CCTParams_st* pCCTParams_st);
```

See `CCTDemoConsole, CCTDemoTests.cpp`
```
    TestSelectedValues(Test_st* pTest_st)
    TestTOD_LEAPSECOND_UTC_XXX(Test_st* pTest_st)
```

4.2.2 SetFrom1970Seconds

Populate a CBF as UTC accurate local date and time-of-day from seconds and decimal fraction values and user input parameters.

Input must be seconds-since-1970 value for local NORMAL time, not DST shifted time.

See `CCT\CCTLib\CCct.h`
```
int CCct::SetFrom1970SecondsFrac(CCTParams_st* pCCTParams_st);
```

See `CCTDemoConsole, CCTDemoTests.cpp`
```
    int TestOneValueAndParseTwice(Test_st* pTest_st)
```

4.2.3 SetIntervalFromSecondsFrac

Populate a CBF as an interval (duration) from SecondsFrac_st user input values and parameters.

The so-called "24 hour period" refers to an 86400 second interval to avoid the use of the term "day" which is reserved for reference to UTC days. See CBF.h - "CBF Construction"

- If SecondsFrac_st < 86400s populates CBF as interval ((I)nterval) < 86400s
  m_CBFTime_st.m_bIsInterval = true; // CBF is an interval
  m_CBFTime_st.m_b24HourPeriodExt = false; // no 24HourPeriodExt present
- If SecondsFrac_st >= 86400s, populates CBF as interval ((P)eriod) >= 86400s
  m_CBFTime_st.m_bIsInterval = true; // CBF is an interval
  m_CBFTime_st.m_b24HourPeriodExt = true; // 24HourPeriodExt present

See `CCT\CCTLib\CCct.h`
```
int CCct::SetIntervalFromSecondsFrac_st(SecondsFrac_st*
                                        pSecondsFrac_st);
```

See `CCTDemoConsole, CCTDemoTests.cpp`
```
    int TestSelectedTimePointsAndIntervals(Test_st* pTest_st)
```

4.2.4 Set24HourTimepointFromSecondsFrac

Populate a CBF as a time-point from SecondsFrac_st user input values and parameters.

Represents a time-point having an abstract zero origin with no relation to calendar date.

The so-called "24 hour period" refers to an 86400 second interval to avoid the use of the term "day" which is reserved for reference to UTC days. See CBF.h - "CBF Construction"

- If SecondsFrac_st < 86400s, as time-point ((T)ime) without date < 86400s
  m_CBFTime_st.m_bIsInterval = false; // CBF is a time-point
  m_CBFTime_st.m_b24HourPeriodExt = false; // no 24HourPeriodExt
- If SecondsFrac_st >= 86400s, populates CBF as time-point ((E)vent) without date >= 86400s
  m_CBFTime_st.m_bIsInterval = false; // CBF is a time-point
  m_CBFTime_st.m_b24HourPeriodExt = true; // 24HourPeriodExt

See `CCT\CCTLib\CCct.h`
```
int CCct::Set24HourTimepointFromSecondsFrac_st(SecondsFrac_st*
                                        pSecondsFrac_st);
```

See `CCTDemoConsole, CCTDemoTests.cpp`
```
    int TestSelectedTimePointsAndIntervals(Test_st* pTest_st)
```

4.2.5 AddUnits (to SecondsFrac_st)

Add units to SecondsFrac_st.

Adding a negative value is subtraction

See CCT\CCTLib\CCct.h
```
int CCct::AddFracUTIL(uint64_t* pui64Addend,
                SecondsFrac_st* pSecondsFrac_st, // in
                SecondsFrac_st* pSecondsFrac_stResult) // out
```

See `CCTDemoConsole, CCTDemoTests.cpp`
```
    int TestAddValue(&Test_stX)
```

4.2.6 AddUnits (to CBF)

Add units to the input CBF values.

The unit resolution is defined by the current rate in effect (CBFTime_st::m_eRateEnumeration).

Adding a negative value performs a subtraction.

See `CCT\CCTLib\CCct.h`
```
static int CCct::AddUnitUTIL(CCbf* pCCbf, uint64_t* pui64Addend);
```

See `CCTDemoConsole, CCTDemoTests.cpp`
```
    int TestAddValue(&Test_stX)
```

4.2.7 Difference

Compute the difference, the duration of the time interval, between two CBF time points (A and B).

- In the case of two CBF time points in the same time zone, this is the absolute duration between the two along that local timescale including any possible Leap-seconds.
- In the case of two CBF time points in different time zones the difference is the absolute duration between the two including the difference between the two local date and times, the UTCT offset difference between the zones, and any possible Leap-seconds.

This difference can be represented as an interval by a CBF or CCF.

See `CCT\CCTLib\CCct.h`
```
static int CCct::DiffUTIL(CCct* pCCctA, CCct* pCCctB, SecondsFrac_st*
                                                    pSecondsFrac_stDiff);
```

See `CCTDemoConsole, CCTDemoTests.cpp`
```
    int TestSelectedDifferences(Test_st* pTest_st)
```

4.2.8 GetCbf1970Seconds

Get total CBF seconds and fractions of seconds. The resolution (rate) is returned.

In the case of UTC accurate local time this is seconds-since-UTC on the local timescale in NORMAl time; DST is never reflected in the CBF counter data.

See `CCT\CCTLib\CCct.h`
```
    int CCct::GetCbf1970Seconds(SecondsFrac_st* pSecondsFrac_st,
                                uint64_t* pui64Exponent = NULL);
```

See `CCTDemoConsole, CCTDemoTests.cpp`
```
    int TestPrintfOneValue(CCct* pCCct)
```

4.2.9 SetLocation

See `CCT\CCTLib\CCct.h`
```
int CCct::SetLocation(char* psLatitude, char* psLongitude, char* psAltitude);
```

See Common Calendar Reference Implementation Guide, Location
See Common Calendar Geostamp

4.3 Common Calendar Character Format (CCF) Operations

Common Calendar Character Format (CCF) represents the binary data and metadata of a corresponding Common Calendar Binary Format (CBF) in a YMDhms form.

A CBF and its corresponding CCF format can have one of six meanings:

- time point with UTC accurate local date and time-of-day
- time point with UTC accurate local date with time portion having no relation to the date
- time point less than 24 hours with no relation to date (< 86400 seconds)
- time point 24 hours or greater with no relation to date (>= 86400 seconds)
- time interval less than 24 hours (< 86400 seconds)
- time interval 24 hours or greater (>= 86400 seconds)

Construction of a CCF from a CBF will reflect any of the CBF supported meanings and parsing a CCF will result in a corresponding CBF.

See Common Calendar Binary Format
See Common Calendar Character Format

4.3.1 SetCcfFromCCbf

Construct a CCF string from CBF binary data and metadata.

A CCF string is constructed within the CCct CCcf instance from CCct CCbf instance binary data and metadata. class CCcf has pointer to corresponding CCbf instance.

The resulting CCF string will reflect the meanings, values, and metadata of the CBF.

Use CCct::GetCcfAscii() to extract the constructed CCF string.

```
See CCT\CCTLib\CCct.h
    int CCct::SetCcfFromCCbf(bool bUserIncludeTime)
```

```
See CCTDemoConsole, CCTDemoTests.cpp
    int TestOneValue(CCct* pCCct, bool bMarkLeapSecond)
```

4.3.2 GetCcfAscii

Get CCF string from class CCcf.

Be sure the CCF has been constructed by CCct::SetCcfFromCCbf();

Copies (strcpy()) CCF to user supplied input string

```
See CCT\CCTLib\CCct.h
    int CCct::GetCcfAscii(char* sCcf);
```

```
See CCTDemoConsole, CCTDemoTests.cpp
    int TestPrintfOneValue(CCct* pCCct)
```

4.3.3 ParseCcfSetCCbf

Parse CCF string and populate a CBF.

```
See CCT\CCTLib\CCct.h
       int CCct::ParseCcfSetCCbf(CCbf* pCCbf, char* sCcf)
```

```
See CCTDemoConsole, CCTDemoTests.cpp
    int TestOneValueAndParse(Test_st* pTest_st)
```

4.4 Common Calendar Calendar Operations

CCT Calendar provides month and year calendar representation.

The core of CCctCalendar is CCctMonth, an array of 42 pointers to class CCbf instances. Each calendar month is thus made up of days of complete date and time-of-day representations made possible by CCbf.

class CCctYear is an array of 12 pointers to instances of CCctMonth thus representing a full year.

class CCctCalendar contains instances of CCctMonth and CCctYear together with method calls to populate months and years and output listings using CCcf character format.

```
See CCTLib, CCctCalendar.h
            CCctCalendar.cpp
              class CCctMonth
              class CCctYear
              class CCctCalendar
```

```
See CCTDemoConsole, CCTDemoTests.cpp
    int TestGetPosixTimeAndCCTCalendar(Test_st* pTest_st)
```

Annex A TestCCctCalendarMonth() Sample Listing shows a sample listing from `CCTDemoTests.cpp`, `TestGetPosixTimeAndCCTCalendar()`

Annex A TestCCctCalendarMonth() Sample Listing

```
=== TestCCctCalendarMonth() ====
Today input value:
D2017-10-14T08:24:58U-05:00Znew_yorkV2016cL37Sc4sMuX        // 1444893925

        OCTOBER 2017
---------------------------
Sun Mon Tue Wed Thu Fri Sat
24  25  26  27  28  29  30
1   2   3   4   5   6   7
8   9   10  11  12  13  14*
15  16  17  18  19  20  21
22  23  24  25  26  27  28
29  30  31  1   2   3   4
---------------------------
D2017-10-14T08:24:58U-05:00Znew_yorkV2016cL37Sc4sMuX        // 1444893925
[SATURDAY 2017 OCTOBER 14 08:24:58:00am America/New_York -05:00]
[DST:01:00 LeapSeconds:37 IsNotLeapYear 1444893925s]

---- CCctCalendar Month array values as CCF ----
D2017-09-24T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1443139227    idx 0
D2017-09-25T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1443225627    idx 1
D2017-09-26T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1443312027    idx 2
D2017-09-27T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1443398427    idx 3
D2017-09-28T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1443484827    idx 4
D2017-09-29T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1443571227    idx 5
D2017-09-30T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1443657627    idx 6
D2017-10-01T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1443744027    idx 7
D2017-10-02T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1443830427    idx 8
D2017-10-03T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1443916827    idx 9
D2017-10-04T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1444003227    idx 10
D2017-10-05T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1444089627    idx 11
D2017-10-06T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1444176027    idx 12
D2017-10-07T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1444262427    idx 13
D2017-10-08T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1444348827    idx 14
D2017-10-09T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1444435227    idx 15
D2017-10-10T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1444521627    idx 16
D2017-10-11T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1444608027    idx 17
D2017-10-12T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1444694427    idx 18
D2017-10-13T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1444780827    idx 19
D2017-10-14T08:24:58U-05:00Znew_yorkV2016cL37Sc4sMuX // 1444893925    idx 20
D2017-10-15T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1444953627    idx 21
D2017-10-16T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1445040027    idx 22
D2017-10-17T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1445126427    idx 23
D2017-10-18T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1445212827    idx 24
D2017-10-19T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1445299227    idx 25
D2017-10-20T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1445385627    idx 26
D2017-10-21T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1445472027    idx 27
D2017-10-22T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1445558427    idx 28
D2017-10-23T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1445644827    idx 29
D2017-10-24T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1445731227    idx 30
D2017-10-25T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1445817627    idx 31
D2017-10-26T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1445904027    idx 32
D2017-10-27T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1445990427    idx 33
D2017-10-28T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1446076827    idx 34
D2017-10-29T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1446163227    idx 35
D2017-10-30T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1446249627    idx 36
D2017-10-31T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1446336027    idx 37
```

```
D2017-11-01T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1446422427    idx 38
D2017-11-02T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1446508827    idx 39
D2017-11-03T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1446595227    idx 40
D2017-11-04T01:00:00U-05:00Znew_yorkV2016cL37Sc4sMuX // 1446681627    idx 41


==== TestCCctCalendarYear() ====
Today input value:
D2017-10-14T08:24:58U-05:00Znew_yorkV2016cL37Sc4sMuX         // 1444893925

        JANUARY 2017
---------------------------
Sun Mon Tue Wed Thu Fri Sat
25  26  27  28  29  30  31
1   2   3   4   5   6   7
8   9   10  11  12  13  14
15  16  17  18  19  20  21
22  23  24  25  26  27  28
29  30  31  1   2   3   4
---------------------------

        FEBRUARY 2017
---------------------------
Sun Mon Tue Wed Thu Fri Sat
29  30  31  1   2   3   4
5   6   7   8   9   10  11
12  13  14  15  16  17  18
19  20  21  22  23  24  25
26  27  28  1   2   3   4
5   6   7   8   9   10  11
---------------------------

        MARCH 2017
---------------------------
Sun Mon Tue Wed Thu Fri Sat
26  27  28  1   2   3   4
5   6   7   8   9   10  11
12  13  14  15  16  17  18
19  20  21  22  23  24  25
26  27  28  29  30  31  1
2   3   4   5   6   7   8
---------------------------

        APRIL 2017
---------------------------
Sun Mon Tue Wed Thu Fri Sat
26  27  28  29  30  31  1
2   3   4   5   6   7   8
9   10  11  12  13  14  15
16  17  18  19  20  21  22
23  24  25  26  27  28  29
30  1   2   3   4   5   6
---------------------------

        MAY 2017
---------------------------
Sun Mon Tue Wed Thu Fri Sat
30  1   2   3   4   5   6
7   8   9   10  11  12  13
14  15  16  17  18  19  20
21  22  23  24  25  26  27
```

```
28  29  30  31  1   2   3
4   5   6   7   8   9   10
--------------------------

        JUNE 2017
--------------------------
Sun Mon Tue Wed Thu Fri Sat
28  29  30  31  1   2   3
4   5   6   7   8   9   10
11  12  13  14  15  16  17
18  19  20  21  22  23  24
25  26  27  28  29  30  1
2   3   4   5   6   7   8
--------------------------

        JULY 2017
--------------------------
Sun Mon Tue Wed Thu Fri Sat
25  26  27  28  29  30  1
2   3   4   5   6   7   8
9   10  11  12  13  14  15
16  17  18  19  20  21  22
23  24  25  26  27  28  29
30  31  1   2   3   4   5
--------------------------

        AUGUST 2017
--------------------------
Sun Mon Tue Wed Thu Fri Sat
30  31  1   2   3   4   5
6   7   8   9   10  11  12
13  14  15  16  17  18  19
20  21  22  23  24  25  26
27  28  29  30  31  1   2
3   4   5   6   7   8   9
--------------------------

        SEPTEMBER 2017
--------------------------
Sun Mon Tue Wed Thu Fri Sat
27  28  29  30  31  1   2
3   4   5   6   7   8   9
10  11  12  13  14  15  16
17  18  19  20  21  22  23
24  25  26  27  28  29  30
1   2   3   4   5   6   7
--------------------------

        OCTOBER 2017
--------------------------
Sun Mon Tue Wed Thu Fri Sat
24  25  26  27  28  29  30
1   2   3   4   5   6   7
8   9   10  11  12  13  14*
15  16  17  18  19  20  21
22  23  24  25  26  27  28
29  30  31  1   2   3   4
--------------------------
D2017-10-14T08:24:58U-05:00Znew_yorkV2016cL37Sc4sMuX         // 1444893925
[SATURDAY 2017 OCTOBER 14 08:24:58:00am America/New_York -05:00]
[DST:01:00 LeapSeconds:37 IsNotLeapYear 1444893925s]
```

NOVEMBER 2017

Sun	Mon	Tue	Wed	Thu	Fri	Sat
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	1	2
3	4	5	6	7	8	9

DECEMBER 2017

Sun	Mon	Tue	Wed	Thu	Fri	Sat
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6