Formal Vindications Time Manager: Technical Specification

Formal Vindications S. L.
Prometheuss Group

`Version - Patch`

Contents

1 Introduction **1**
1.1 Formal Verification 2
1.2 About time and calendars 2
1.3 Local time and time zones 6

2 Formal Vindications Time Manager Behavior **9**
2.1 The calendar in The Coq Time Library 10
2.2 General remarks on the extraction 11
2.3 Datatypes 13
2.4 Functions 15
2.5 Error messages 29

3 Coq References **31**

A Tables **33**

B tz Tables **35**

1 Introduction

This document contains the *Technical Specifications* for the **Formal Vindications Time Manager** (**FVTM**); a formally verified library that provides various time-related functions. The main functionalities of **FVTM** are verified using the Coq proof assistant and subsequently extracted to OCaml via the program extraction mechanism.

> This document is an approximation to the formal specification in Coq using a semi-formal notation. In case there is any divergence of this document from the formal specification, the latter prevails over the former.

One of the main features of **FVTM** is that this library contains a proper implementation of the **Coordinated Universal Time** (UTC) time standard. This means that **FVTM** supports leap seconds and therefore, timestamps are obtained with UTC-precision. Furthermore, **FVTM** also provides functions to manage different UTC local time zones. These extra functions are directly implemented in OCaml and make use of the `tz` database.

To the best of our knowledge, **FVTM** is the first formally verified time library implementing UTC-time standard.

1.1 Formal Verification

The technology of *verified software* is founded in the following idea: the software is given by a triple (Σ, Π, Δ) of parts of code written in a formal language – in our case, specifically in the computer language of the Coq proof assistant.

- $\boldsymbol{\Sigma}$ is the *formal specification*, that tells with mathematical precision and rigor what the software should do. It is completely unambiguous in the strict mathematical sense;

- $\boldsymbol{\Pi}$ consists of the code implementing algorithms oriented to computational efficiency, the resulting code should behave according to Σ;

- $\boldsymbol{\Delta}$ is a mathematical *proof* that the software (Π) does exactly what the formal specification (Σ) says it should do.

Some functions have an inefficient, more mathematical version in Σ and an efficient, more algorithmic version in Π. Then the part of Δ regarding this function is just a proof that both versions coincide (give the same results to the same arguments). Sometimes, depending on the nature of the function itself, there is just one implementation in Π, and in Σ the statement of one or several theorems expressing the correctness of the function; then Δ is the proof of these theorems.

Coq and OCaml

Coq is a formal proof management system. It provides a formal language to write formal specifications of algorithms as well as their implementation.

Coq is the result of about 30 years of research. It started in 1984 from an implementation of the Calculus of Constructions and later was extended to the Calculus of Inductive Constructions; CIC for short. The CIC logical framework is a constructive theory based on dependent types. Constructive means that in order to prove that something is true, we need to *construct* evidence for that. For instance if we want to prove that there is a number satisfying some property, we need to find such number and show that it indeed satisfies the property. One of the main benefits of the constructive approach is computability. In more detail, this means that the proofs built within Coq are algorithms that can be executed on a computer. This allows us to design and implement algorithms that are guaranteed to behave as expected, in other words, they will never crash, hang or behave differently than their specification.

Now, since Coq has such a strong mathematical expressiveness, it is not as any other programming language, and in particular running Coq code is higly inefficient. Thus, we use a tool given by Coq which is called *extraction*. The extraction process is the translation of Coq code into another functional programming language which is meant to be run, in our case, OCaml. Extraction is just an automatic syntactic translation from one language to another one.

OCaml is an industry-strength functional programming language which is renowned for its robustness and performance (Coq itself is implemented in OCaml). The main benefits of OCaml reside in its safety. OCaml is a strongly typed language where the types of all values are checked during compilation to ensure that they are well defined. Thus, any typing error will be picked up at compile-time by the instead of at run-time.

An important remark is that the extraction process, which is also performed by a software, is not verified, i.e., there is no proof to the extent that the program which performs the translation is correct. Since the translation is quite simple and the extraction is widely used in the Coq community, we can have good trust that it will not introduce any bugs into our extracted code, but the possibility will be there until the extraction process itself is verified.

In conclusion, at the end of this process what we obtain is an efficient OCaml program which, up to the unverified extraction, is guaranteed not to misbehave.

1.2 About time and calendars

A time standard is a specification for measuring time: 1. the rate at which time passes; 2. points in time; 3. both. In modern times, several time specifications have been officially recognized as standards, where formerly they were matters of custom and practice. Standardized time measurements are made using a clock to count periods of some period changes, which may be either the changes of a natural phenomenon or of an artificial machine.

Historically, time standards were often based on the Earth's rotational period. From the late 18th century to the 19th century it was assumed that the Earth's daily rotational rate was constant. Astronomical observations of several kinds, including eclipse records, studied in the 19th century, raised suspicions that the rate at which Earth rotates is gradually slowing and also shows small-scale irregularities, and this was confirmed in the early 20th century. The invention in 1955 of the caesium atomic clock has led to the replacement of older and purely astronomical time standards, for most practical purposes, by newer time standards based wholly or partly on atomic time.

There are essentially two types of time standards.

- **Solar:** they are based on Earth's rotation period, which defines a day. Seconds in these systems are defined as the $\frac{1}{86400}$ of a day. Since the rotation period turns out to slightly vary, seconds are not constant.
- **Atomic:** they are based on the SI definition of second, which is designed to be a constant amount of time. The definition is "the duration of 9 192 631 770 periods of the radiation corresponding to the transition between two hyperfine levels of the ground state of the caesium 133 atom". In these systems, a day is defined as a fixed number of SI seconds (usually 86400, but as we shall see this varies among systems).

Since the Earth's rotation period varies, and in particular it is generally slowing, a time standard with days defined as 86400 SI seconds will gradually differ from solar time. This may be irrelevant for some purposes, but when civil time is somehow involved this can present a problem: without adjustments, noon in atomic time could even happen during a night (in the sense of period with no solar light).

UTC, UT1 and TAI

Mean solar time was originally apparent solar time corrected by the equation of time. Mean solar time was sometimes derived, especially at sea for navigational purposes, by observing apparent solar time and then adding to it a calculated correction, the equation of time, which compensated for two known irregularities, caused by the ellipticity of the Earth's orbit and the obliquity of the Earth's equator and polar axis to the ecliptic (which is the plane of the Earth's orbit around the sun).

UT1 is one of the systems based on this of mean solar time and Earth's rotation. It is a modern continuation of GMT. While conceptually it is mean solar time at 0° longitude, precise measurements of the Sun are difficult. Hence, it is computed from observations of distant quasars using long baseline interferometry, laser ranging of the Moon and artificial satellites, as well as the determination of GPS satellite orbits. UT1 is the same everywhere on Earth, and is proportional to the rotation angle of the Earth with respect to distant quasars, specifically, the International Celestial Reference Frame (ICRF), neglecting some small adjustments. The format is generally the Gregorian calendar date, together with the time expressed in hours, minutes and seconds.

On the other hand, atomic times are measured using devices called atomic clocks, first built during the 1950s. Thus, in practice all these time standards are measured as the number of seconds elapsed since a designated time, called epoch. However, following civil customs and standards, these dates are most usually expressed in Gregorian calendar format.

International Atomic Time (TAI, from the French name *temps atomique international*) is a high-precision atomic coordinate time standard based on the notional passage of proper time on Earth's geoid. It is also the basis for Coordinated Universal Time (UTC), which is used for civil timekeeping all over the Earth's surface.

TAI may be reported using traditional means of specifying days, carried over from non-uniform time standards based on the rotation of the Earth. Specifically, both Julian Dates and the Gregorian calendar are used.

For the Gregorian calendar format, in TAI, seconds are SI seconds, minutes always have 60 seconds, hours always have 60 minutes, and days always have 24 hours. TAI in this form was synchronised with UT1 at the beginning of 1958, and the two have drifted apart ever since, due to the changing motion of the Earth. TAI as of December 2018 reads approximately 37 seconds ahead of UT1.

Coordinated Universal Time (abbreviated to UTC) is the primary time standard by which the world regulates clocks and time. It is within about 0.9 seconds of UT1, and is not adjusted for daylight saving time.

Since UTC uses SI seconds (and in particular it is based on TAI), some adjustments are needed to keep it close to UT1. The solution adopted is as follows: days always have 24 hours, which always have 60 minutes, but minutes have occasionally 59 or 61 seconds. The possibility of 59 seconds is theoretical, in practice this has never happened: since SI seconds are slightly shorter than solar seconds in average, it is extremely unlikely that a minute of 59 seconds will occur. When a minute of 61 seconds occurs, the extra second is called leap second.

Hence, UTC is based on TAI with leap seconds added at irregular intervals to compensate for the slowing of the Earth's rotation. Leap seconds are inserted as necessary to keep UTC within 0.9 seconds of UT1, and are thus unpredictable in the long-term.

Leap seconds When it occurs, a positive leap second is inserted between second 23:59:59 of a chosen UTC calendar date and second 00:00:00 of the following date. The definition of UTC states that the last day of December and June are preferred, with the last day of March or September as second preference, and the last day of any other month as third preference. All leap seconds (as of 2017) have been scheduled for either June 30 or December 31. The extra second is displayed on UTC clocks as 23:59:60. On clocks that display local time tied to UTC, the leap second may be inserted at the end of some other hour (or half-hour or quarter-hour), depending on the local time zone. A negative leap second would suppress second 23:59:59 of the last day of a chosen month, so that second 23:59:58 of that date would be followed immediately by second 00:00:00 of the following date. Since the introduction of leap seconds, the mean solar day has outpaced UTC only for very brief periods, and has not triggered a negative leap second.

Because the Earth's rotation speed varies in response to climatic and geological events, UTC leap seconds are irregularly spaced and unpredictable. Insertion of each UTC leap second is usually decided about six months in advance by the International Earth Rotation and Reference Systems Service (IERS), when needed to ensure that the difference between the UTC and UT1 readings will never exceed 0.9 seconds. See Table 7 in Appendix A

The modern version of UTC, implemented in 1972, established leap seconds and synchronised UTC with TAI, with an initial difference of 10 seconds, which was the approximate difference at the time between TAI and UT1. More precisely, UTC was set so that 1 January 1972 00:00:00 UTC was exactly 1 January 1972 00:00:10 TAI. As of December 2018, there have been 27 leap seconds added to UTC, so that currently TAI is exactly 37 seconds ahead of UTC.

The second is the basic unit in **FVTM** and since it contains a proper implementation of UTC, a second in this library will be understood as *atomic second*.

Calendars in FVTM

A calendar is a system of organizing days for social, religious, commercial or administrative purposes. This is done by giving names to periods of time, typically days, weeks, months and years. A date is the designation of a single, specific day within such a system.

There is a *de facto* standard civil calendar, known as the Gregorian calendar. It is a solar calendar, which means that it assigns a date to each solar day.

The Julian calendar proposed by Julius Caesar in 46 BC, took effect on 1 January 45 BC. It was the predominant calendar in the Roman world, most of Europe, and in European settlements in the Americas and elsewhere.

The Julian calendar has two types of year: "normal" years of 365 days and "leap" years of 366 days. There is a simple cycle of three "normal" years followed by a leap year and this pattern repeats forever without exception. The Julian year is, therefore, on average 365.25 days long. Consequently, the Julian year drifts over time with respect to the tropical year.

The Gregorian calendar has the same months and month lengths as the Julian calendar, but, in the Gregorian calendar, year numbers divisible by 100 are not leap years, except that those divisible by 400 remain leap years. To state it clearer, the rule says:

Every year divisible by 400 is a leap year.
Every year divisible by 4 but not by 100 is also a leap year.
No other year is a leap year.

The Gregorian calendar is the most widely used as a civil calendar. Furthermore, the Gregorian calendar can be extended backwards to dates preceding its official introduction in 1582. This is known as the proleptic Gregorian calendar, which is explicitly recommended for all dates before 1582 by ISO 8601:2004.

The calendar format of **FVTM** is the *Gregorian UTC Calendar*. By that, we mean exactly UTC time since its modern definition in 1970 with the format of the Gregorian calendar: days fit into years using the commonly known 12 months of 28 to 31 days each, for a total of 365 or 366 days in a year, and leap (366-day) years add a day after February 28 called February 29. Moreover, the *Gregorian UTC Calendar* takes into account leap seconds.

The OCaml part of **FVTM** takes 1970-1-1 00:00:00 as epoch, and valid times range since 1970-1-1 00:00:00 until 9999-12-31 23:59:59. Internally, the Coq implementation of the calendar includes times since 1-1-1 00:00:00 until 9999-12-31 23:59:59, since that definition allows us to give a clean formal specification of the calendar and an arithmetical expression for determining leap years. This should be seen as a mathematical extension of the Gregorian UTC calendar which has no physical support (since atomic clocks did not exist). We of course assume there are no leap seconds prior to 1970. We call the theoretical calendar resulting from those assumptions *proleptic Gregorian UTC calendar*.

Observe that in this calendar, the duration of the second is constant, but the other components are not. The duration of the minutes measured in seconds is not constant because of the possibility of leap seconds, and so, this irregularity propagates to hours and days. The duration of the months is not constant, adding to the leap seconds problem the fact of the leap day in February in leap years and the 31-30 alternation. And then of course, also the duration of the years is not constant due to leap seconds and the leap day in February.

Because of all this, the duration of the interval between two points in the Gregorian UTC calendar made in seconds is mathematically consistent and clear, but the duration can not be grouped uniformly in higher regular units corresponding with calendar positions unless we introduce hard extra definitions. Also nowadays some functions about adding or subtracting a duration to a date which has not specified these complications are being used and the result gives paradoxes. This kind of paradoxes are due to the fact that these methods are not adding constant intervals, but just changing the date components.

FVTM also provides a solution to this question by defining the **Formal Calendar**. It is aimed to group the duration in higher units than seconds, keeping the meaning of the intuitive minute, hour, day, month and year defined in Gregorian Calendar. Note that actually is not a calendar in the proper sense of representing a point in the time line, but we keep that name because of the names of the units. The formal second is equivalent to the atomic second and thus constant. With the formal second, the remaining units are easily definable:

- the formal minute duration will be 60 seconds;
- the formal hour duration will be 60 formal minutes;
- the formal day duration will be 24 formal hours;
- the formal month duration will be 30 formal days;
- the formal year duration will be 365 formal days, that is 12 formal month duration plus 5 formal days duration, which is the same as 31.536.000 seconds.

The way of counting time

In this section e discuss the duration of an interval as cardinal of a set instead of as the arithmetical difference between the end points of the interval.

Measuring time is counting the number of movements made by the stick of the clock from one to another position. More generally, for a clock or any machine with periodic repetitive movements, measuring time is counting the number of (periodic repetitive) movements made by one central element (electron inside of an atom, pendulum, clock stick...). We name the positions between movements by $1, 2, 3, \ldots, n+1, \ldots$.

In the case of a round clock, with 60 small different marks and 60 spaces among them, the positions are fixed and the fact of being round makes the first and the last position the same $1, 2, 3, \ldots, 60 = 1, 2, \ldots$. A cleaner mathematical solution to this is to use the number 0 for the first position, where no movement has been made, thus identifying the name of a position with the number of movements needed to reach it from the origin. By the fact of being round, the position 60 is the same that the 0 starting again the cycle.

The definition of interval between positions n_1 and n_2 is the set of all the consecutive numbers of positions between n_1 and n_2 including both, and is represented by $[n_1, n_2] = \{n_1, n_1 + 1, \ldots, n_2\}$.

For defining the duration of the interval $[n_1, n_2]$ we can use the "common sense" or "comon idea": the number of movements the central elements did to reach the position n_2 starting from position n_1. Hence, since the number of movements until n_2 is the number of movements until n_1 plus the movements from n_1 to n_2, the number of movements from n_1 to n_2 can be computed as $n_2 - n_1$. Note that in any partition of this kind we also have one space less than sticks, and this coincides with the cardinal of the set minus one of the elements. Hence:

$$\text{Duration}\,[n_1, n_2] = n_2 - n_1 = \text{Cardinal}\,[n_1, n_2)\,.$$

For example, the duration of the calendar-style interval [1970/1/1/00:00:00, 1970/1/1/00:01:00] would be:

$$\text{Duration}\,[0, 60] = 60 - 0 = \text{Cardinal}\,[0, 60) = 60.$$

And similarly, the duration of [1970/1/1/00:01:00, 1970/1/1/00:02:00] would be:

$$\text{Duration}\,[60, 120] = 120 - 60 = \text{Cardinal}\,[60, 120) = 60.$$

In **FVTM** differences between dates are proven equal to the cardinal definition. For a more detailed examination see Table 9 in Appendix 7.

1.3 Local time and time zones

Local-UTC conversion

FVTM is conceived to work in UTC. However this library also provides additional OCaml (non-verified) functions to manage different local time zones. With these extra functionalities we can translate local times to UTC and *vice versa*. Once local times are converted into UTC times, we can use them as arguments to any of the remaining functions.

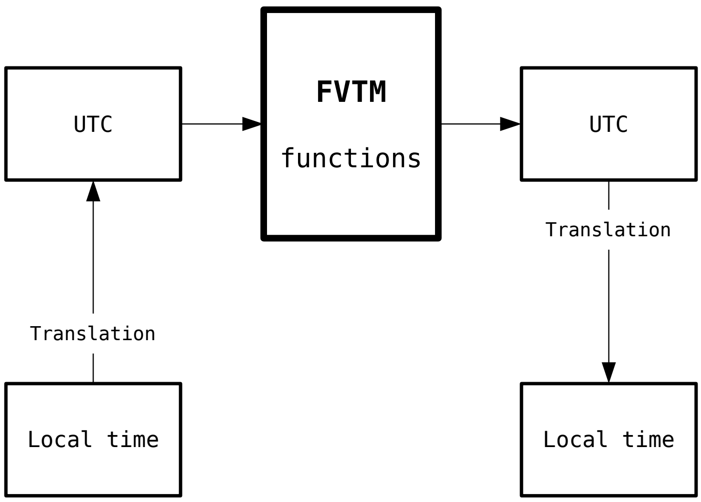

Figure 1: Behavior of the **FVTM**

Translating local times into UTC is a good practice in order to avoid inaccuracies. Here we have a real example of the industry to illustrate it. In the road transport industry the tacograph is used to ensure trucks follow the legislation. One of the main concepts in this law is the *Continuous Driving Time*. We can define the driving time of an interval as the duration of the driving activity until the next non-driving activity. Consider the ordered list of times and activities in Figure 2. According to the definition of driving time, the interval is [00:55, 01:05] and then the duration of the interval yields 10 minutes of driving.

If we translate the list of activities to local time in Portugal, having in mind that on 2018/10/28/2:0:0 there was a DST change from UTC+1 to UTC+0 we have:

The second row would be 02:05 in UTC+1, but just at 02:00 we change from UTC+1 to UTC+0, the situation is similar with the fourth row. If we calculate again the driving time, the interval is now [01:55, 22:00] and then the duration of the interval yields 20 hours and 5 minutes of driving.

There is not a perfect and definitive solution for this problem, since the time change occurs at one point or another. This same driving time duration change can also happen when a driver moves from a country or region to another in a different time zone.

activity	time	kind
Break/Rest	2018/10/27/23:30:00	UTC
Driving	2018/10/28/00:55:00	UTC
Break/Rest	2018/10/28/01:05:00	UTC
Work	2018/10/28/22:00:00	UTC

Figure 2: Tachograph example of times and activities

activity	time	kind
Break/Rest	2018/10/27/00:30:00	Portugal Local time: UTC+1
Break/Rest	2018/10/28/01:05:00	Portugal Local time: UTC+0
Driving	2018/10/28/01:55:00	Portugal Local time: UTC+1
Work	2018/10/28/22:00:00	Portugal Local time: UTC+0

Hence **FVTM** will make all the computations with the uniform UTC time and later if this is the user preference the results can be translated to a local time.

In this section we introduce a new setting to consider local times.

While UTC time can be considered like a uniform time for the whole planet, we need to include the local times for different regions of the world. We have the concept of local time of a region, which is the translation of UTC corresponding to the longitude of the region, the so-called offset and also the possibility of DST, which is Daylight Saving Time, the hour added in some places facing the summer to take more profit of the solar light.

We need to have in mind that in several places the offset does not correspond exactly to the longitude of the region. For instance we have the case of Spain which is physically in the Greenwich Meridian and thus it can have a +0 offset like Portugal but instead of it we have an offset of +1.

So we are going to present functions to go from UTC to local time and the converse, from local time to UTC:

The first one which take as input a `time` and a code of the region of the world, outputs data of a new datatype representing the local time, this depending on the region and the date will be UTC+n, where n is the offset or UTC+n+1 if the region consider DST and the date is in that period.

The function from local time to UTC, conversely, takes as input datatype for local time and the region code and will outputs data of type `time` representing the UTC time. Again this depends on the region and the date.

Also we can consider a function that given the code of a region and an interval of years outputs the time structure of the region in that period, that is, the offset, whether the region consider or not DST and in an affirmative answer the dates for that changes in the given period.

The Formal Vindications Time Manager includes an international homologated database called `tz` database (`tz` stands for time zones)[1]. This database contains the offset of the regions of the whole planet and also the date and time of the changes for the regions which consider daylight saving time (DST) changes. It is updated whenever is necessary (and also we will update the **FVTM**). This data can also be confirmed in the site `timeanddate.com` which has a more comfortable way of visualizing the information.

The last decision regarding the DST changes lies in each country –or even in the international institutions to which the country belongs–. Sometimes the fact whether a region consider DST or not changes by decree in little time, as was the case of Brazil last year, which makes some problems to the industry. We hope that the inclusion of the `tz` database will be helpful for that kind of problems.

Together with the rest of the functionalities, we give an explanation of the functions that **FVTM** provides to deal with local times, and at the end of the document we attach the whole list of regions considered. The proper name of the regions for the purpose of local time is *time zone*. As codes for the regions, we will use the international names of the time zones as they appear in the `tz` database. Hence the list is helpful to know the codes one needs to use the functions.

[1] See: `https://es.wikipedia.org/wiki/TZ_Database` and also `https://en.wikipedia.org/wiki/List_of_tz_database_time_zones`

`tz` Database

The Time Zone Database (often called `tz` or zoneinfo) contains a collaborative compilation of code and data that presents the history of local time for many representative locations around the globe. The database records historical time zones and all civil changes since 1970. It includes transitions such as daylight saving time, and also records leap seconds. `tz` is used by several implementations, including the GNU C Library (used in GNU/Linux), Android, Chromium OS, MySQL, webOS, AIX, BlackBerry 10, iOS, macOS, *et al.*

The `tz` database is in the public domain. Time zones and daylight-saving rules are controlled by individual governments. Therefore, sometimes changes are introduced with little notice. `tz` is updated periodically to capture these changes made by political bodies. Proposed changes are sent to the `tz` mailing list. These changes are usually propagated to clients via OS updates.

In `tz`, time zones are named in the form *Area/Location*:

- **Area:** is the (English) name of a continent, an ocean, or "Etc". The continents and oceans currently used are *Africa, America, Antarctica, Arctic, Asia, Atlantic, Australia, Europe, Indian,* and *Pacific*. The special area of "Etc" is used for some administrative zones.

- **Location:** is the name of a specific location within the area (usually the most populous city although other cities may be selected if they are more widely known, due to disambiguation or if the name has more than 14 characters).

Observe that country names are not used in this scheme since they are affected by frequent political changes.

Some examples of time zone names are: America/New_York, Europe/Oslo, Asia/Beirut, etc. In some cases, three-level names are used where *Location* is itself a compound name. For example America/Argentina/Cordoba or America/Indiana/Indianapolis.

2 Formal Vindications Time Manager Behavior

In this section we shall explain the behavior of **FVTM**. The **FVTM** provides the following files:

- `FVTMnc.ml`;
- `FVTMnc.mli`;
- `FVTM.ml`;
- `timezones.ml`;
- `Coq Time Library`.

`FVTMnc.ml`

This file contains a version of **FVTM** which is directly extracted from Coq so that users can operate with it. In this version, the code shall not perform **input validation**.irectly extracted code does **not** perform proper testing of any input supplied by the user. This is because Coq, as a programming language, is completely pure, which means that the behavior of a program cannot change at execution time –in particular, it does not accept input at execution time. The only possible input comes from inside of Coq and it needs to be proven correct. We shall refer to this version of the **FVTM** as *Pure*.

`FVTMnc.mli`

A `*.mli` file is an OCaml Interface Source. This file is the exported signature of the module. The compiler enforces it in order to compile the `*.ml` code.

`FVTM.ml`

This OCaml file contains code which is almost directly extracted from Coq. The slight modification of the original extracted code is done in order to perform **input validation**. While directly extracted code does **not** perform proper testing of any input, when extracted to OCaml, we expect the program to accept input at execution time, and since input is inherently prompt to mistakes and errors, it needs to be validated, i.e., checked correct. The slight modification of the extracted code only does that: it takes an input, uses a function directly extracted from Coq to check the correctness of the input, and only after that executes the function over the input. Since this version of the **FVTM** requires an extra OCaml layer, we shall refer to it as *Impure*.

`timezones.ml`

The file `timezones.ml` is OCaml auxiliary code to communicate the `tz` database with the **FVTM**. The programmers using the manager do not need to use this file directly. This file is directly implemented in OCaml and thus, part of the *Impure* version.

Pure version files	*Impure version files*
`FVTMnc.ml`	`FVTM.ml`
`FVTMnc.mli`	`timezones.ml`

`Coq Time Library`

Contains the whole project written in Coq together with its own documentation.

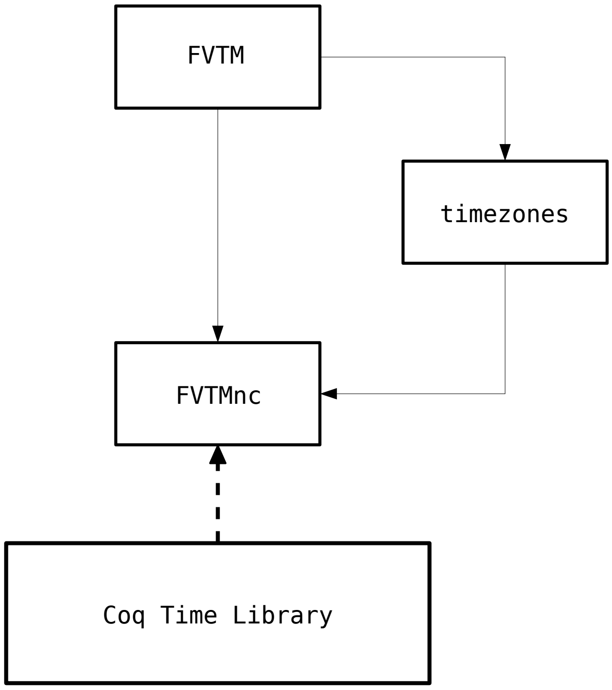

Figure 3: Flowchart of **FVTM** file dependencies.

2.1 The calendar in The Coq Time Library

In this section, we present technical details about the formal specification and implementation of the calendar in Coq that can be useful to understand the warnings that we give below for the use of the OCaml code. However, readers that are only interested in using the library and not in the details of how it works can skip this section.

The core functionalities of the **FVTM** are the conversions between times and timestamps, in both directions. The proof of correctness for these two conversions is where the mathematical strength lies – the rest of the proofs depend on those two. For that reason, a mathematically clean formal definition of the Gregorian calendar was needed, including the determination of leap years, and that is why we chose to represent the proleptic Gregorian calendar since 1-1-1 00:00:00 until 9999-12-31 00:00:00.

Since Coq is a mathematical-oriented language, we have types that are not usual in regular programming languages, for instance `nat`, which is the type of the natural numbers starting at 0, $\mathbb{N} = \{0, 1, 2, \ldots\}$. In Coq, the datatype `time` is roughly defined as six `nat`s (year, month, day, hour, minute, and second) and a proof that they satisfy the restrictions (i.e., that the date and time make sense and inside of the range from 1-1-1 00:00:00 to 9999-12-31 23:59:59). Then, we have the two core functions:

- `timestamp`: receives a `time` and returns a `nat` which represents the timestamp of the `time` with epoch 1-1-1 00:00:00. That means that timestamp 0 represents time 1-1-1 00:00:00.

- `from_timestamp`: performs the opposite conversion, receives a `nat` and returns a `time`, assuming again epoch 1-1-1 00:00:00, i.e., timestamp 0 represents time 1-1-1 00:00:00.

To prove correctness, we prove a theorem which says that `timestamp` behaves exactly as a formal, mathematical description of what timestamp is. Then, we prove that `from_timestamp` is the inverse function.

Then, since we are interested in extracting these functions with epoch 1970-1-1 00:00:00 because this is the standard epoch for UTC measured with atomic clocks, the Coq development continues as follows. It defines new versions, called `utc_timestamp` and `from_utc_timestamp`, of the above functions, which have the 1970 epoch and are defined using the above ones. In particular, `utc_timestamp` over a time `t` is defined as follows:

$$\texttt{utc\_timestamp}(t) = \texttt{timestamp}(t) - \texttt{timestamp}(\text{1970-1-1 00:00:00})$$

After that we prove them correct using the theorems for the above versions. This is important because, as we shall see in the next section, the Coq type `nat` gets extracted to the OCaml type `int`, with relevant consequences.

Hence, it should be clear now that The Coq Time Library understands dates starting at year 1, but the part that gets extracted to OCaml only treats with dates starting at year 1970.

2.2 General remarks on the extraction

Coq is a language capable of expressing both algorithms and mathematics, but it is too inefficient to run for industrial purposes. In order to have code efficient enough to run, Coq code is automatically extracted to OCaml code, and in this section we shall explain all the problems derived from the extraction process itself.

First of all, **the extraction process is not verified**. A software translates Coq code to OCaml code and this software could contain bugs. Therefore, the OCaml code is formally verified except for the extraction. However, there are good reasons to trust the extraction process if it is performed in a smart way[2]. Moreover, the extraction tool is widely used in the Coq community and no critical bugs are found. As a team, we look forward to the day verified extraction is reached.

Now, there is a particularity of Coq that has crucial consequences during extraction, so let us briefly visit it. Since Coq is designed as a mathematical environment, the natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$ are represented by the type `nat`, which is unary. This means that `nat` is defined as a type that contains the zero element `O`, and a function `S` called "successor" which generates more elements. In particular, number 1 is expressed as `S O`, number 5 is expressed as `S (S (S (S (S O))))`, number 5000 does not fit in a piece of paper, and number five million does not fit in the RAM of a modern computer. This definition makes sense in Coq because it is not meant to compute, it is meant to prove mathematical results. However, in OCaml, as in most programming languages, this type does not exist[3], and instead there is a type `int` that efficiently represents the integers $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ using the typical machine binary representation, and the extraction process takes Coq's `nat` to OCaml's `int`.

Thus, functions that in Coq receive as input a natural number, when extracted to OCaml receive an integer, opening the possibility for the user to introduce negative integers. This has some important consequences that we explain below as a warning. In the case of the impure version (`FVTM.ml`), this is not a problem, since input validation detects that the received integer is negative and issues an error message. In the pure version, since there is no input validation as the code is directly extracted from Coq, when an invalid input is given there are no error messages and the result of the functions can be misleading.

In short, we call this particularity of extraction "`nat->int`", and we give warnings in the tables below when functions are affected by this issue.

In a similar way, other datatypes that have restrictions in Coq (like `time`), when extracted purely to OCaml can accept invalid data, leading to unpredictable and confusing results.

> The functions of **FVTM** are supposed to work over valid data, i.e, data satisfying the restrictions expressed in sections 2.3 and 2.4. Using the functions of **FVTM** over invalid input:
>
> - In the pure version, may give misleading results, or results that do not satisfy the datatype restrictions.
> - In the impure version, triggers an exception and gives an error message.

The tables in sections 2.3 and 2.4 contain enough warnings to safely use the **FVTM** functions, but here we give an exhaustive list of the misleading consequences that using invalid inputs may have.

1. The first consequence is particularly relevant in cases like function 12 `from_utc_timestamp`. While the expected input should be greater or equal than 0, this input validation can not be performed in the pure version. Thus, a negative integer can be processed by the function e.g.

[2] In particular, if the part of the Coq code which is meant for extraction, i.e., the implementation Π, is written in a reduced fragment of the language of Coq, the translation to OCaml is just a syntactic transformation simple enough to have reasonable trust in the extraction process.

[3] We should observe that in `FVTMnc.ml` references to type `nat` might be found when calling some functions or constructors. However, this is a rename of the type `int` performed by OCaml (`type nat = int`). In other words, the `nat` references in OCaml are simply `int`.

FVTMnc.ml

```
# from_utc_timestamp (-10);;
- : time =
{rawDate_of_rawTime = {year = 1969; month = 12; day = 31}; hour = 23;
 minute = 59; second = 50}
```

This regression with respect to the epoch makes consistent the use of negative integers with the formal specification in Coq (recall that dates in the Coq Time Library range from year 1 up to year 9999), but it is not valid in OCaml, since our valid times in OCaml start in 1970.

Analogously, we could use `to_Formaltime` with a negative argument. The type `formalTime` only makes sense with natural numbers, but since inputs can not be filtered by the pure version, the function will reproduce a negative `formalTime`:

FVTMnc.ml

```
# to_FormalTime (-120);;
- : formalTime = {fY = 0; fM = 0; fD = 0; fh = 0; fm = -2; fs = 0}
```

In the *pure files*, the user is expected to make a correct use of this functions checking the input manually. **An inconsistent use of the functions will lead to misleading results**. Therefore, in functions:

- 03 - `max_second`;
- 12 - `from_utc_timestamp`;
- 28 - `days_of_month`;
- 29 - `is_leap_year`;
- 32 - `to_FormalTime`;

for a valid performance, the type `int` should be interpreted as a **non-negative integer**.

2. In the formal specification of the Coq Time Library, the `formalTime` type is defined as a structure of elements of type `nat` (see Table 5) and hence, `formalTime` is always positive. However, in *pure files* due to `nat->int`, negative values are available to use within the `formalTime` structure. As before, the correctness of these functions depends on the consistent use of them, and so, in functions:

   - 13 - `addFormal`;
   - 30 - `subtractFormal`;
   - 31 - `from_FormalTime`;

   `formalTime` should be use as a structure with **non-negative integers**. Actually, the type `formalTime` should always be used with non-negative integers, as expressed in the restrictions of the datatypes in Section 2.3.

3. A similar kind of restriction holds for the types `time` and `date`. In these cases the user is expected not only to use the previous `nat` restriction, but also the restrictions imposed by the Gregorian calendar, e.g. `mkDate 2020 2 30` is not a valid date since February 30th does not exist in the Gregorian calendar, where February contains only 28 days, or 29 days in a leap year.

4. During the extraction, some types and functions are renamed in order to make them syntactically coherent, to avoid clashing with some other OCaml objects with the same name or to respect OCaml syntax. However, internally the original Coq names are sometimes preserved by OCaml. References about Coq datatypes and function can be found in Section 3.

Future work Newer versions of Coq introduce a type for the usual machine integers with binary representation, the same integers OCaml uses. Our future work regards the possibility of providing mathematical proof of the equivalence of a bounded fragment of `nat` and the non-negative fragment of the integers, in such a way that we would solve two problems at once: first, we would avoid extracting from the unbounded type `nat` to the bounded type `int`[4]; and second, we would be able to control inside of Coq the behavior of the functions when negative inputs are given.

2.3 Datatypes

The following table contains the types provided by FVTM. Since the pure version does not validate inputs, the item Error messages only makes sense in the context of FVTM.ml. The following table contains the types provided by **FVTM**. Since the pure version does not validate inputs, the item Error messages only makes sense in the context of `FVTM.ml`. The Restrictions are automatically checked in the impure version `FVTM.ml`, but in the pure `FVTMnc.ml` the user should check them. Also recall that the `nat->int` remark and the Extraction remark apply only to the pure version `FVTMnc.ml`.

Name	Description	Constructors
`timestamp`	Kind of data: The number of seconds between a particular date and 1970-1-1 00:00:00 in UTC. Explanation: A type for natural numbers interpreted as seconds since 1970-1-1 00:00:00 including leap seconds. For example, the timestamp of 2020-1-1 00:00:00 is 1577836827. Recall that UNIX timestamp does not represent UTC seconds after 1970-1-1 00:00:00. Restrictions: naturals from 0 to 253402300826. The maximum value will change as leap seconds are introduced. Error messages: 03. Version: Impure.	`mkTimestamp n`
`date`	Kind of data: Format for points in the Gregorian UTC calendar time line with precision to days. Explanation: A structure of natural numbers `Y M D` representing a year, a month and a day respectively. Restrictions: $1970 \leq$ `Y` ≤ 9999, $1 \leq$ `M` ≤ 12 and $1 \leq$ `D` $\leq$ `days_of_month Y M`, where `days_of_month` is a function which assigns to a year and a month the number of days that the month has in that year. Error messages: 02. Version: Pure/Impure.	`mkDate Y M D` Extraction remark: In the pure version the constructor does not check that the Restrictions hold.
`time`	Kind of data: Format for points in the Gregorian UTC calendar time line with precision to seconds. Explanation: A structure of natural numbers `Y M D h m s`, where `Y` represents a year, `M` a month, `D` a day `h` an hour, `m` a minute and `s` a second. Restrictions: `Y M D` need to form a `date`, i.e. satisfy the Restrictions for `date`. $0 \leq$ `h` ≤ 23 $0 \leq$ `m` ≤ 59 $0 \leq$ `s` $\leq$ `max_second YMDhm`, where `max_second` is a function (see 3) that gives the last second of that minute on that hour and date; meaning 60 if it is a leap second or 59 otherwise. Error messages: 01. Version: Pure/Impure.	`mkTime Y M D h m s` Extraction remark: In the pure version the constructor does not check that the Restrictions hold.

[4] In our development, the maximum valid timestamp is seven orders of magnitude below the maximum integer representable in OCaml, so for the **FVTM** overflowing the machine integer is not an issue; but still, this is an important problem in many other industrial developments.

`clock`	Kind of data: Type for expressing times without dates. Explanation: A structure of natural numbers `ch cm cs`, for hours, minutes and seconds respectively. Restrictions: -. Error messages: -. Version: Pure/Impure.	Only used internally and as output
`formalTime`	Kind of data: Type for the duration of time interval between two points in Gregorian UTC calendar. Explanation: A structure of natural numbers `fY fM fD fh fm fs`, where `fY` represents an amount of formal years, `fM` represents an amount of formal months, `fD` an amount of formal days, `fh` an amount of formal hours, `fm` represents an amount of formal minutes and `fs` an amount of formal seconds. Restrictions: $0 \leq$ `fs` < 60, $0 \leq$ `fm` < 60, $0 \leq$ `fh` < 24, $0 \leq$ `fD` < 30 and $0 \leq$ `fM` < 12 or `fM` $= 12$ and $0 \leq$ `fD` < 5. Error messages: 10. Version: Pure/Impure.	`mkFormalTime fY fM fD fh fm fs` Extraction remark: In the pure version the constructor does not check that the Restrictions hold.
`localTime`	Kind of data: Format for points in the Gregorian calendar of any timezone with precision to seconds. Explanation: A string of natural numbers `Y M D h m s`, where `Y` represents a year, `M` a month, `D` a day `h` an hour, `m` a minute and `s` a second. Restrictions: $1970 \leq$ `Y` ≤ 9999, $1 \leq$ `M` ≤ 12 and $1 \leq$ `D` $\leq$ `days_of_month Y M`, where `days_of_month` is a function which assigns to a year and a month the number of days that the month has in that year. $0 \leq h \leq 23$, $0 \leq m \leq 59$, $0 \leq s \leq 60$, since in principle a leap second could occur at any minute depending on the timezone. Error messages: 11. Version: Impure.	`mkLocalTime Y M D h m s`

Apart from that, **FVTM** makes use of the following OCaml types:

Name	Description
`int`	Kind of data: OCaml basic built-in type. Explanation: The type for integer numbers. Restrictions: 31-bit signed int (roughly +/- 1 billion) on 32-bit processors, or 63-bit signed int on 64-bit processors. Error messages: Provided by OCaml. Version: Pure/Impure.
`bool`	Kind of data: OCaml basic built-in type. Explanation: The type for boolean values: `true` and `false`. Restrictions: Type restricted to elements `true` and `false`. Error messages: Provided by OCaml. Version: Pure/Impure.

`string`	Kind of data: OCaml basic built-in type. Explanation: A string is an immutable data structure that contains a fixed-length sequence of (single-byte) characters. Restrictions: The maximum string length is 16777211 on 32-bit processors and 144115188075855863 on 64-bit processors. Error messages: Provided by OCaml. Version: Impure.
`'a list`	Kind of data: OCaml basic built-in type. Explanation: The type of lists. It can be instantiated with elements of any type. The symbols `'a` mean that any type can take that place. We will use it as lists of dates `date list` and lists of times `time list`. Restrictions: -. Error messages: Provided by OCaml. Version: Pure/Impure.

2.4 Functions

The following table contains the functions provided by **FVTM**. Since the pure version does not validate inputs, as in the case of the datatypes table, the item Error messages only makes sense in the context of `FVTM.ml`. Furthermore, the type of functions 11 and 12 change according to its pure/impure version. With respect to the examples, the Usage Example references are obtained from the `FVTM.ml` file. Recall that the `nat->int` remark only applies to the pure version.

All the functions in the pure version are in the file `FVTMnc.ml`, and all the functions in the impure version are in the file `FVTM.ml`.

Time zones **FVTM** can be used with different UTC local time zones (see functions 33 and 34). Recall that these extra functions are directly implemented in OCaml and make use of the `tz` database. The string input accepted by these functions correspond to the time zones names in the `tz` databse, that is, the names that can be found in column **TIME ZONE** of the tables in Appendix B.

	Name	Description of the function:

01	`date_of_time`	Input: `time`
		Output: `date`
		Explanation: Projection of the `date` part of a `time`.
		Use: `date_of_time` `time` = `date`
		Usage Example:
		`# date_of_time (mkTime 2020 2 22 10 20 30);;`
		`- : date = {FVTLnc.year = 2020; month = 2; day = 22}`
		Error messages: 01; 02.
		Version: Pure/Impure.

02	`time_of_time`	Input: `time`
		Output: `clock`
		Explanation: Takes as input a `time` and returns the `clock` corresponding to that `time`.
		Use: `time_of_time` `time` = `clock`
		Usage Example:
		`# time_of_time (mkTime 2020 2 22 10 20 30);;`
		`- : clock = {FVTLnc.chour = 10; cminute = 20; csecond = 40}`
		Error messages: 01; 02.
		Version: Pure/Impure.

03	`max_second`	Input: `date` `int` `int`
		Output: `int`
		`nat->int` remark: Input elements of type `int` must be **non-negative** (see Section 2.2).
		Explanation: Takes as input a `date` together with two elements of type `int` representing the hour and minute respectively.
		It returns the maximum value of the second for that date at that hour and minute. Thus, the possible outcomes will be 59 for a regular minute, 60 for a positive leap second and 58 for a negative leap second.
		Use: `max_second` `time` `int` `int` = `int`
		Usage Example:
		`# max_second (mkDate 2020 2 22) 10 20;;`
		`- : int = 59`
		Error messages: 02; 04.
		Version: Pure/Impure.

04	`second`	Input: `time`
		Output: `int`
		Explanation: Projection of the second component of a `time`.
		Use: `second` `time` = `int`
		Usage Example:
		`# second (mkTime 2020 2 22 10 20 30);;`
		`- : int = 30`
		Error messages: 01; 02.
		Version: Pure/Impure.

05	`minute`	Input: `time`
		Output: `int`
		Explanation: Projection of the minute component of a `time`.
		Use: `minute` `time` = `int`
		Usage Example:
		`# minute (mkTime 2020 2 22 10 20 30);;`
		`- : int = 20`
		Error messages: 01; 02.
		Version: Pure/Impure.

06	`hour`	Input: `time`
		Output: `int`
		Explanation: Projection of the hour component of a `time`.
		Use: `hour` `time` = `int`
		Usage Example:
		`# hour (mkTime 2020 2 22 10 20 30);;`
		`- : int = 10`
		Error messages: 01; 02.
		Version: Pure/Impure.

07	`day`	Input: `date`
		Output: `int`
		Explanation: Projection of the day component of a `date`.
		Use: `day` `date` = `int`
		Usage Example:
		`# day (mkDate 2020 2 22);;`
		`- : int = 22`
		Error messages: 02.
		Version: Pure/Impure.

08	`day_of_week`	Input: `date`
		Output: `int`
		Explanation: Returns the day of the week of a `date` using numerical encoding 1 = Mon, 2 = Tue, etc.
		Use: `day_of_week` `date` = `int`
		Usage Example:
		`# day_of_week (mkDate 2020 2 22);;`
		`- :  int = 6`
		Error messages: 02.
		Version: Pure/Impure.

09	`month`	Input: `date`
		Output: `int`
		Explanation: Projection of the month component of a `date`.
		Use: `month` `date` = `int`
		Usage Example:
		`# month (mkDate 2020 2 22);;`
		`- :  int = 2`
		Error messages: 02.
		Version: Pure/Impure.

10	`year`	Input: `date`
		Output: `int`
		Explanation: Projection of the year component of a `date`.
		Use: `year` `date` = `int`
		Usage Example:
		`# year (mkDate 2020 2 22);;`
		`- :  int = 2020`
		Error messages: 02.
		Version: Pure/Impure.

11	`utc_timestamp`	Input: `time`
		Output: `int` (pure)
		`timestamp` (impure)
		Explanation: Conversion from `time` to its `timestamp` both types in UTC with leap seconds.
		Use: `utc_timestamp` `time` = `timestamp`
		Usage Example:
		`# utc_timestamp (mkTime 2020 2 22 10 20 30);;`
		`- :  timestamp = RawTimestamp 1582366857`
		Error messages: 01; 02.
		Version: Pure/Impure.

12	`from_utc_timestamp`	Input: `int` (pure) `timestamp` (impure) Output: `time` `nat->int` remark: Input element of type `int` must be **non-negative and at most 253402300826** (see Section 2.2). Negative inputs can give times between year 1 and 1969 which are implemented in Coq, or even negative years, which are not there by design. Inputs bigger than 253402300826 may give a `time` not satisfying the restrictions of the type. Explanation: Conversion from `timestamp` to its `time` both types in UTC with leap seconds. Use: `from_utc_timestamp` `timestamp` = `time`

		Usage Example:
		`# from_utc_timestamp (Pr_TS 1582366857);;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2020; month = 2; day = 22}; hour = 10; minute = 20; second = 30}`
		Error messages: 03; 07; 08.
		Version: Pure/Impure.

13	`addFormal`	Input: `time` `formalTime`
		Output: `time`
		Explanation: Adding to a `time` a duration interval of type `formalTime` we obtain a new position in the Gregorian calendar with an element of type `time`.
		Output control: In Coq. If the result of the operation would be bigger than the maximum time allowed, in the pure version the output is a special time 10000-01-01-00:00:00, and in the impure version an error message is issued.
		Use: `addFormal` `time` `formalTime` = `time`
		Usage Example:
		`# addFormal (mkTime 2020 2 22 10 20 30) (mkFormalTime 1 1 1 1 1 1);;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2021; month = 3; day = 24}; hour = 11; minute = 21; second = 31}`
		Error messages: 01; 02; 04; 07; 08; 10.
		Version: Pure/Impure.

14	`shiftUTCSeconds`	Input: `time` `int`
		Output: `time`
		`nat->int` remark: This function is not affected because the input `int` is an integer also in Coq. Negative values are accepted.
		Explanation: From an element of type `time`, take the component `second` and shift it (forward or backward) a number of times determined by the argument of type `int`.
		It is done accordingly to the UTC calendar and taking into account leap seconds. If the resultant `time` is not a valid `time` it takes the last existing `time` before of that one.
		Output control: In Coq. If the result of the operation would be bigger than the maximum time allowed, in the pure version the output is a special time 10000-01-01-00:00:00, and in the impure version an error message is issued.
		Use: `shiftUTCSeconds` `time` `int` = `time`
		Usage Example:
		`# shiftUTCSeconds (mkTime 2020 2 22 10 20 30) 125;;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2020; month = 2; day = 22}; hour = 10; minute = 22; second = 35}`
		
		`# shiftUTCSeconds (mkTime 2020 2 22 10 20 30) (-125);;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2020; month = 2; day = 22}; hour = 10; minute = 18; second = 25}`
		Error messages: 01; 02; 04; 07; 08.
		Version: Pure/Impure.

15	`addFormalSeconds`	Input: `time` `int`
		Output: `time`
		`nat->int` remark: This function is not affected because the input `int` is an integer also in Coq. Negative values are accepted.
		Explanation: Adding an amount of formal seconds to a `time`.
		Output control: In Coq. If the result of the operation would be bigger than the maximum time allowed, in the pure version the output is a special time 10000-01-01-00:00:00, and in the impure version an error message is issued.
		Use: `addFormalSeconds` `time` `int` = `time`
		Usage Example:
		`# addFormalSeconds (mkTime 2020 2 22 10 20 30) 125;;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2020; month = 2; day = 22}; hour = 10; minute = 22; second = 35}`
		
		`# addFormalSeconds (mkTime 2020 2 22 10 20 30) (-125);;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2020; month = 2; day = 22}; hour = 10; minute = 18; second = 25}`
		Error messages: 01; 02; 04; 07; 08.
		Version: Pure/Impure.

16	`shiftUTCMinutes`	Input: `time` `int`
		Output: `time`
		`nat->int` remark: This function is not affected because the input `int` is an integer also in Coq. Negative values are accepted.
		Explanation: From an element of type `time`, take the component `minute` and shift it (forward or backward) a number of times determined by the argument of type `int`.
		It is done accordingly to the UTC calendar and taking into account leap seconds. If the resultant `time` is not a valid `time` it takes the last existing `time` before of that one.
		Output control: In Coq. If the result of the operation would be bigger than the maximum time allowed, in the pure version the output is a special time 10000-01-01-00:00:00, and in the impure version an error message is issued.
		Use: `shiftUTCMinutes` `time` `int` = `time`
		Usage Example:
		`# shiftUTCMinutes (mkTime 2020 2 22 10 20 30) 125;;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2020; month = 2; day = 22}; hour = 12; minute = 25; second = 30}`
		
		`# shiftUTCMinutes (mkTime 2020 2 22 10 20 30) (-125);;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2020; month = 2; day = 22}; hour = 8; minute = 15; second = 30}`
		Error messages: 01; 02; 04; 07; 08.
		Version: Pure/Impure.

17	`addFormalMinutes`	Input: `time` `int`
		Output: `time`
		`nat->int` remark: This function is not affected because the input `int` is an integer also in Coq. Negative values are accepted.
		Explanation: Adding an amount of formal minutes to a `time`.
		Output control: In Coq. If the result of the operation would be bigger than the maximum time allowed, in the pure version the output is a special time 10000-01-01-00:00:00, and in the impure version an error message is issued.
		Use: `addFormalMinutes` `time` `int` = `time`
		Usage Example:
		`# addFormalMinutes (mkTime 2020 2 22 10 20 30) 125`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2020; month = 2; day = 22}; hour = 12; minute = 25; second = 30}`
		
		`# addFormalMinutes (mkTime 2020 2 22 10 20 30) (-125);;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2020; month = 2; day = 22}; hour = 8; minute = 15; second = 30}`
		Error messages: 01; 02; 04; 07; 08.
		Version: Pure/Impure.

18	`shiftUTCHours`	Input: `time` `int`
		Output: `time`
		`nat->int` remark: This function is not affected because the input `int` is an integer also in Coq. Negative values are accepted.
		Explanation: From an element of type `time`, take the component `hour` and shift it (forward or backward) a number of times determined by the argument of type `int`.
		It is done accordingly to the UTC calendar and taking into account leap seconds. If the resultant `time` is not a valid `time` it takes the last existing `time` before of that one.
		Output control: In Coq. If the result of the operation would be bigger than the maximum time allowed, in the pure version the output is a special time 10000-01-01-00:00:00, and in the impure version an error message is issued.
		Use: `shiftUTCHours` `time` `int` = `time`
		Usage Example:
		`# shiftUTCHours (mkTime 2020 2 22 10 20 30) 125;;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2020; month = 2; day = 27}; hour = 15; minute = 20; second = 30}`
		
		`# shiftUTCHours (mkTime 2020 2 22 10 20 30) (-125);;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2020; month = 2; day = 17}; hour = 5; minute = 20; second = 30}`
		Error messages: 01; 02; 04; 07; 08.
		Version: Pure/Impure.

19	`addFormalHours`	Input: `time` `int`
		Output: `time`
		`nat->int` remark: This function is not affected because the input `int` is an integer also in Coq. Negative values are accepted.
		Explanation: Adding an amount of formal hours to a `time`.
		Output control: In Coq. If the result of the operation would be bigger than the maximum time allowed, in the pure version the output is a special time 10000-01-01-00:00:00, and in the impure version an error message is issued.
		Use: `addFormalHours` `time` `int` = `time`
		Usage Example:
		`# addFormalHours (mkTime 2020 2 22 10 20 30) 125;;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2020; month = 2; day = 27}; hour = 15; minute = 20; second = 30}`
		
		`# addFormalHours (mkTime 2020 2 22 10 20 30) (-125);;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2020; month = 2; day = 17}; hour = 5; minute = 20; second = 30}`
		Error messages: 01; 02; 04; 07; 08.
		Version: Pure/Impure.

20	`shiftUTCDays`	Input: `time` `int`
		Output: `time`
		`nat->int` remark: This function is not affected because the input `int` is an integer also in Coq. Negative values are accepted.
		Explanation: From an element of type `time`, take the component `day` and shift it (forward or backward) a number of times determined by the argument of type `int`.
		It is done accordingly to the UTC calendar and taking into account leap seconds. If the resultant `time` is not a valid `time` it takes the last existing `time` before of that one.
		Output control: In Coq. If the result of the operation would be bigger than the maximum time allowed, in the pure version the output is a special time 10000-01-01-00:00:00, and in the impure version an error message is issued.
		Use: `shiftUTCDays` `time` `int` = `time`
		Usage Example:
		`# shiftUTCDays (mkTime 2020 2 22 10 20 30) 125;;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2020; month = 6; day = 26}; hour = 10; minute = 20; second = 30}`
		
		`# shiftUTCDays (mkTime 2020 2 22 10 20 30) (-125);;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2019; month = 10; day = 20}; hour = 10; minute = 20; second = 30}`
		Error messages: 01; 02; 04; 07; 08.
		Version: Pure/Impure.

21	`addFormalDays`	Input: `time` `int`
		Output: `time`
		`nat->int` remark: This function is not affected because the input `int` is an integer also in COQ. Negative values are accepted.
		Explanation: Adding an amount of formal days to a `time`.
		Output control: In COQ. If the result of the operation would be bigger than the maximum time allowed, in the pure version the output is a special time 10000-01-01-00:00:00, and in the impure version an error message is issued.
		Use: `addFormalDays` `time` `int` = `time`
		Usage Example:
		`# addFormalDays (mkTime 2020 2 22 10 20 30) 125;;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2020; month = 6; day = 26}; hour = 10; minute = 20; second = 30}`
		
		`# addFormalDays (mkTime 2020 2 22 10 20 30) (-125);;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2019; month = 10; day = 20}; hour = 10; minute = 20; second = 30}`
		Error messages: 01; 02; 04; 07; 08.
		Version: Pure/Impure.

22	`shiftUTCMonths`	Input: `time` `int`
		Output: `time`
		`nat->int` remark: This function is not affected because the input `int` is an integer also in COQ. Negative values are accepted.
		Explanation: From an element of type `time`, take the component `month` and shift it (forward or backward) a number of times determined by the argument of type `int`.
		It is done accordingly to the UTC calendar and taking into account leap seconds. If the resultant `time` is not a valid `time` it takes the last existing `time` before of that one.
		Output control: In COQ. If the result of the operation would be bigger than the maximum time allowed, in the pure version the output is a special time 10000-01-01-00:00:00, and in the impure version an error message is issued.
		Use: `shiftUTCMonths` `time` `int` = `time`
		Usage Example:
		`# shiftUTCMonths (mkTime 2020 2 22 10 20 30) 125;;`
		`- : time ={FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2030; month = 8; day = 22}; hour = 10; minute = 20; second = 30}`
		
		`# shiftUTCMonths (mkTime 2020 2 22 10 20 30) (-125);;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2009; month = 10; day = 22}; hour = 10; minute = 20; second = 30}`
		Error messages: 01; 02; 04; 07; 08.
		Version: Pure/Impure.

23	`addFormalMonths`	Input: `time` `int`
		Output: `time`
		`nat->int` remark: This function is not affected because the input `int` is an integer also in Coq. Negative values are accepted.
		Explanation: Adding an amount of formal months to a `time`.
		Output control: In Coq. If the result of the operation would be bigger than the maximum time allowed, in the pure version the output is a special time 10000-01-01-00:00:00, and in the impure version an error message is issued.
		Use: `addFormalMonths` `time` `int` = `time`
		Usage Example:
		`# addFormalMonths (mkTime 2020 2 22 10 20 30) 125;;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2030; month = 5; day = 30}; hour = 10; minute = 20; second = 30}`
		
		`# addFormalMonths (mkTime 2020 2 22 10 20 30) (-125);;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2009; month = 11; day = 16}; hour = 10; minute = 20; second = 33}`
		Error messages: 01; 02; 04; 07; 08.
		Version: Pure/Impure.

24	`shiftUTCYears`	Input: `time` `int`
		Output: `time`
		`nat->int` remark: This function is not affected because the input `int` is an integer also in Coq. Negative values are accepted.
		Explanation: From an element of type `time`, take the component `year` and shift it (forward or backward) a number of times determined by the argument of type `int`.
		It is done accordingly to the `UTC` calendar and taking into account leap seconds. If the resultant `time` is not a valid `time` it takes the last existing `time` before of that one.
		Output control: In Coq. If the result of the operation would be bigger than the maximum time allowed, in the pure version the output is a special time 10000-01-01-00:00:00, and in the impure version an error message is issued.
		Use: `shiftUTCYears` `time` `int` = `time`
		Usage Example:
		`# shiftUTCYears (mkTime 2020 2 22 10 20 30) 125;;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2145; month = 2; day = 22}; hour = 10; minute = 20; second = 30}`
		
		`# shiftUTCYears (mkTime 2020 2 22 10 20 30) (-25);;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 1995; month = 2; day = 22}; hour = 10; minute = 20; second = 30}`
		Error messages: 01; 02; 04; 07; 08.
		Version: Pure/Impure.

25	addFormalYears	Input: `time` `int`
		Output: `time`
		nat->int remark: This function is not affected because the input `int` is an integer also in Coq. Negative values are accepted.
		Explanation: Adding an amount of formal years to a `time`.
		Output control: In Coq. If the result of the operation would be bigger than the maximum time allowed, in the pure version the output is a special time 10000-01-01-00:00:00, and in the impure version an error message is issued.
		Use: addFormalYears `time` `int` = `time`
		Usage Example:
		`# addFormalYears (mkTime 2020 2 22 10 20 30) 125;;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2145; month = 1; day = 22}; hour = 10; minute = 20; second = 30}`
		
		`# addFormalYears (mkTime 2020 2 22 10 20 30) (-25);;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 1995; month = 2; day = 28}; hour = 10; minute = 20; second = 38}`
		Error messages: 01; 02; 04; 07; 08.
		Version: Pure/Impure.

26	timeDifference	Input: `time` `time`
		Output: `formalTime`
		Explanation: If the first argument is greater than or equal to the second argument, it computes the difference in `formalTime` between them, i.e. the duration. Otherwise, it returns 0 in the pure version, and it issues error message 09 in the impure version.
		Use: timeDifference `time` `time` = `formalTime`
		Usage Example:
		`# timeDifference (mkTime 2020 2 22 10 20 30) (mkTime 2020 2 11 5 10 25);;`
		`- : formalTime = {FVTLnc.fY = 0; fM = 0; fD = 11; fh = 5; fm = 10; fs = 5}`
		Error messages: 01; 02; 09.
		Version: Pure/Impure.

27	secTimeDifference	Input: `time` `time`
		Output: `int`
		Explanation: If the first argument is greater than or equal to the second argument, it computes the difference in seconds between them, i.e. the duration. Otherwise, it returns 0 in the pure version, and it issues error message 09 in the impure version.
		Usage Example:
		`# secTimeDifference (mkTime 2020 2 22 10 20 30) (mkTime 2020 2 11 5 10 25);;`
		`- : int = 969005`
		Error messages: 01; 02; 09.

28	`days_of_month`	Input: `int` `int`
		Output: `int`.
		`nat->int` remark: Input elements of type `int` must be **non-negative** and correspond to a valid year (1970-9999) and month (1-12) (see Section 2.2).
		Explanation: The number of days of a month with respect to that year. The first argument of the function is expected to be the year while the second argument should be the month.
		Use: `days_of_month` `int` `int` = `int`
		Usage Example:
		`# days_of_month 2020 2;;`
		`- : int = 29`
		Error messages: 06.
		Version: Pure/Impure.

29	`is_leap_year`	Input: `int`
		Output: `bool`
		`nat->int` remark: Input element of type `int` must be **non-negative** and correspond to a valid year (1970-9999) (see Section 2.2).
		Explanation: Whether a year is leap or not.
		Takes as input a natural number representing a year, namely `Y` and outputs a boolean with the meaning: 1 if it is leap and 0 otherwise.
		Use: `is_leap_year` `int` = `bool`
		Usage Example:
		`# is_leap_year 2020;;`
		`- : bool = true`
		Error messages: 05.
		Version: Pure/Impure.

30	`subtractFormal`	Input: `time` `formalTime`
		Output: `time`
		Explanation:
		Subtracting to a `time` a `formalTime`.
		Use: `SubtractFormal` `time` `formalTime` = `time`
		Usage Example:
		`# subtractFormal (mkTime 2020 2 22 10 20 30) (mkFormalTime 1 1 1 1 1 1);;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2019; month = 1; day = 22}; hour = 9; minute = 19; second = 29}`
		Error messages: 01; 02; 10.
		Version: Pure/Impure.

31	`from_FormalTime`	Input: `formalTime`
		Output: `int`
		Explanation:
		Convert to seconds an element of type `formalTime`.
		Use: `from_FormalTime` `formalTime` = `int`
		Usage Example:
		`# from_FormalTime (mkFormalTime 1 1 1 1 1 1);;`
		`- : int = 34218061`
		Error messages: 04; 10.
		Version: Pure/Impure.

32	to_FormalTime	Input: `int`
		Output: `formalTime`
		nat->int remark: Input element of type `int` must be **non-negative** (see Section 2.2).
		Explanation: Convert to `formalTime` an amount of time given in seconds.
		Use: To_FormalTime `int` = `formalTime`
		Usage Example:
		`# to_FormalTime 34218061;;`
		`- : formalTime = {FVTLnc.fY = 1; fM = 1; fD = 1; fh = 1; fm = 1; fs = 1}`
		Error messages: 04.
		Version: Pure/Impure.

33	utc_of_ltime_zname	Input: `localTime` `string`
		Output: `time`
		Explanation: Given a local time and the name of a timezone, it gives the corresponding UTC time.
		Use: utc_of_ltime_zname `localTime` `string` = `time`
		Usage Example:
		`# utc_of_ltime_zname (mkLocalTime 2020 2 22 10 20 30) ''Europe/Madrid'';;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2020; month = 2; day = 22}; hour = 9; minute = 20; second = 30}`
		Error messages: 11; 12; 13; 14; 15.
		Version: Impure.

34	ltime_of_utc_zname	Input: `time` `string`
		Output: `localTime`
		Explanation: Given a UTC time and the name of a timezone, it gives the corresponding local time.
		Use: ltime_of_utc_zname `time` `string` = `localTime`
		Usage Example:
		`# ltime_of_utc_zname (mkTime 2020 2 22 10 20 30) ''Europe/Madrid'';;`
		`- : time = {FVTLnc.rawDate_of_rawTime = {FVTLnc.year = 2020; month = 2; day = 22}; hour = 11; minute = 20; second = 30}`
		Error messages: 01; 02; 07; 08; 15.
		Version: Impure.

35	date_le	Input: `date` `date`
		Output: `bool`
		Explanation: Given two objects of type `date`, it gives `true` if the first one is less than or equal to the second one, and `false` otherwise. In other words, $\leq$ for dates.
		Use: date_le `date` `date` = `bool`
		Usage Example:
		`# date_le (mkDate 2015 10 11) (mkDate 2015 11 10);;`
		`- : bool = true`
		Error messages: 02.
		Version: Pure/Impure.

36	date_lt	Input: `date` `date`
		Output: `bool`
		Explanation: Given two objects of type `date`, it gives **true** if the first one is less than the second one, and **false** otherwise. In other words, < for dates.
		Use: date_lt `date` `date` = `bool`
		Usage Example:
		`# date_lt (mkDate 2015 10 11) (mkDate 2015 10 11);;`
		`- : bool = false`
		Error messages: 02.
		Version: Pure/Impure.

37	sort_dates	Input: `date list`
		Output: `date list`
		Explanation: Given a `date list`, it returns a list which contains the same elements in increasing order. In other words, it returns an ordered `date list` according to $\leq$ for dates.
		Use: sort_dates `date list` = `date list`
		Usage Example:
		`# sort_dates [(mkDate 2015 10 11); (mkDate 2015 11 10); (mkDate 2015 10 11)];;`
		`- : date list = [{year = 2015; month = 10; day = 11}; {year = 2015; month = 10; day = 11}; {year = 2015; month = 11; day = 10}]`
		Error messages: 02.
		Version: Pure/Impure.

38	time_le	Input: `time` `time`
		Output: `bool`
		Explanation: Given two objects of type `time`, it gives **true** if the first one is less than or equal to the second one, and **false** otherwise. In other words, $\leq$ for times.
		Use: time_le `time` `time` = `bool`
		Usage Example:
		`# time_le (mkTime 2007 03 31 10 56 43) (mkTime 2007 03 31 14 40 21);;`
		`- : bool = true`
		Error messages: 01; 02.
		Version: Pure/Impure.

39	time_lt	Input: `time` `time`
		Output: `bool`
		Explanation: Given two objects of type `time`, it gives **true** if the first one is less than the second one, and **false** otherwise. In other words, < for times.
		Use: time_lt `time` `time` = `bool`
		Usage Example:
		`# time_lt (mkTime 2007 03 31 10 56 43) (mkTime 2007 03 31 10 56 43);;`
		`- : bool = false`
		Error messages: 01; 02.
		Version: Pure/Impure.

40	`sort_times`	Input: `time list`
		Output: `time list`
		Explanation: Given a `time list`, it returns a list which contains the same elements in increasing order.
		Use: `sort_times` `time list` = `time list`
		Usage Example:
		`# sort_times [(mkTime 2007 03 31 10 56 43); (mkTime 2007 03 31 14 40 21)];;`
		`- : time list = [{rawDate_of_rawTime = {year = 2007; month = 3; day = 31}; hour = 10; minute = 56; second = 43}; {rawDate_of_rawTime = {year = 2007; month = 3; day = 31}; hour = 14; minute = 40; second = 21}]`
		Error messages: 01; 02.
		Version: Pure/Impure.

Advantages of the Formal Calendar

As we already explained in page 5, expressing durations of intervals of time in units higher than seconds can be problematic. The type `formalTime` represents the objects of our formal calendar, which is a standard way of expressing durations.

This solution makes the functions `addFormal`, `subtractFormal` and `timeDifference` consistent in the following sense: given `A1`, `A2` two objects of type `time`, assume `A2` is bigger than `A1` and `D = timeDifference A2 A1` is the difference between them given in `formalTime`; then we have:

- `A2 = addFormal A1 D`, and
- `A2 = substractFormal A2 D`.

The functions whose names start by `shiftUTC` are defined to fulfill classical applications needed for some activities, but they are not consistent in the above sense. Its use can lead to some kind of paradoxes.

Example of the paradoxes Informally, we can say that function `shiftUTCComponent` shifts the corresponding `Component` of a date forwards or backwards the number of times we give as input.

These functions have corrections that change higher components when we change `Component` enough times, i.e., when there is carry. Thus, it is not only a circular transformation of `Component` – which could be another option. For instance, if we change the second component a number of times bigger or equal than the number of seconds remaining to complete a minute, then the minute component will be also changed, and others if the carry goes on.

The functions also have corrections so that the output is always a valid `time`. When the basic shift gives a non-valid time, the function chooses the closest previous valid time. The paradoxes announced above arise from this fact.

Consider for instance `shiftUTCMonths` and a date like 2020-03-31 00:00:00. If we shift forwards once the month component by `shiftUTCMonths (mkTime 2020 3 31 0 0 0) 1` we get 2020-04-30 00:00:00. The raw shift gives 2020-04-31 00:00:00 but since this is not a valid time the function goes to the closest previous valid time, which is the given result. But now, if we apply again the function to shift backwards this time, `shiftUTCMonths (mkTime 2020 4 30 0 0 0) -1` gives 2020-03-30 00:00:00 and not 2020-03-31 00:00:00. This is one of the possible paradoxes, the so called $1 - 1 \neq 0$. This is an undesirable behavior from the arithmetical viewpoint, since we have checked that it is **not** always the case that

```
shiftUTCMonths (shiftUTCMonths t n) -n = t.
```

Arithmetic using the Formal Calendar Using our Formal Calendar as a system of units, with its corresponding `addFormalComponent` functions, we avoid the arithmetical paradoxes and problems that the `shiftUTCComponent` functions present. For instance, composition behaves as expected, e.g.

```
addFormalComponent (addFormalComponent t n) -n = t.
```

Going back to the previous example,

```
addFormalMonths (addFormalMonths (mkTime 2020 3 31 0 0 0) 1) -1 = 2020-03-31 00:00:00.
```

2.5 Error messages

	Error message	Description of the error
01	Only times in UTC (with leap seconds and starting in 1970, ending in 9999) are accepted	Raised when the components given for creating a `time` do not comply with the allowed bounds and hence, the time does not exist in UTC
02	Only dates in UTC (starting in 1970, ending in 9999) are accepted	Raised when the components given for creating a `date` do not comply with the allowed bounds and hence, the date does not exist in UTC
03	Only timestamps between 0 and 253402300826 are accepted	Raised when the timestamp given does not satisfy the bounds
04	Integer out of bounds: either the function does not accept negative inputs, or the value is too big for machine representation	Raised when the integer given is bigger than the maximum machine integer (depends on the user's system), or when it is negative and the function does not accept negative values
05	Only years between 1970 and 9999 are accepted	Raised when the year given to a function that expects a year (for example, `is_leap_year`) falls out of bounds
06	Months are a number between 1 and 12	Raised when the month given to a function that expects a month (for example, `days_of_month`) falls out of bounds
07	Underflow: the resulting time is before 1970	Raised when the result of shifting or adding a duration to a `time` is previous to the epoch
08	Overflow: the resulting time is after 9999	Raised when the result of shifting or adding a duration to a `time` is after the maximum time allowed
09	Time difference can only be computed if the first argument is greater than or equal to the second one	Raised when trying to compute the `timeDifference` or `secTimeDifference` with a first argument smaller than the second
10	The components of a formal time should not be negative. The second component should be less than 60. The minute component should be less than 60. The hour component should be less than 24. The day component should be less than 30. The month component should be either less than 12, or equal to 12 only in case the day component is less than 5. This is because formal years have 365 days, and 12 formal months are 360 days.	Raised when some component given for creating a `formalTime` does not satisfy the restrictions
11	Accepted local times are valid UTC times with the exception of a possible second = 60	Raised when the components given for creating a `localTime` do not comply with the allowed bounds, which are the same as for `time` except for the `second`, which can be equal to 60
12	The timezone does not have a leap second occurring at that time	Raised when the local time given by the user has `second` = 60 and the corresponding minute does not have any leap second at the given timezone

13	The local time you introduced is ambiguous in [the given timezone] due to DST. It happened first at [first UTC time it happened]. It happened again at [second UTC time it happened].	Raised when the given local time is ambiguous because there was a change of time for daily saving time (DST). For example, 2019-11-03 01:15:00 America/New_York
14	The local time you introduced does not exist in [the given timezone] due to DST. The change of time was at [UTC time DST occured].	Raised when the given local time does not exist because there was a change of time for daily saving time (DST). For example, 2019-03-31 02:00:00 Europe/Berlin
15	The timezone [given zone name] was not found	Raised when the time zone introduced does not correspond to any valid time zone name in the tz database

3 Coq References

OCaml	Coq
date	rawDate = { year : nat; month : nat; day : nat; }
time	rawTime = {rawDate_of_rawTime : rawDate; hour : nat; minute : nat; second : nat;}
clock	rawClock = { chour : nat; cminute : nat; csecond : nat; }
formalTime	formalTime = {fY : nat; fM : nat; fD : nat; fh : nat; fm : nat; fs : nat;}

Table 5: OCaml - Coq types correspondence.

OCaml		Coq	
	Name:	Name:	References:
1	date_of_time	rawDate_of_rawTime	Part of the ontology
2 - 10	Direct extraction from Coq		
11	utc_timestamp	Unix_timestamp	Theorem Unix_timestampE
12	from_UTC_timestamp	from_Unix_timestamp	Theorem from_Unix_timestampE
13	addFormal	AddFormal	Lemma addP Lemma add_from_timestampP Lemma add_valid
14	shiftUTCSeconds	ShiftUTCSeconds	Lemma ShiftUTCSeconds_valid Lemma ShiftUTCSecondsP
15	addFormalSeconds	AddFormalSeconds	Lemma AddFormalSeconds_valid Lemma AddFormalSeconds_timestamp Lemma AddFormalSeconds_no_Overflow Lemma AddFormalSeconds_no_Underflow Lemma AddFormalSeconds_from_timestamp
16	shiftUTCMinutes	ShiftUTCMinutes	Lemma ShiftUTCMinutes_valid Lemma ShiftUTCMinutesP
17	addFormalMinutes	AddFormalMinutes	Depends on AddFormalSeconds
18	shiftUTCHours	ShiftUTCHours	Lemma ShiftUTCHours_valid Lemma ShiftUTCHoursP
19	addFormalHours	AddFormalHours	Depends on AddFormalSeconds
20	shiftUTCDays	ShiftUTCDays	Lemma ShiftUTCDays_valid Lemma ShiftUTCDaysP
21	addFormalDays	AddFormalDays	Depends on AddFormalSeconds
22	shiftUTCMonths	ShiftUTCMonths	Lemma ShiftUTCMonths_valid Lemma ShiftUTCMonthsP
23	addFormalMonths	AddFormalMonths	Depends on AddFormalSeconds
24	shiftUTCYears	ShiftUTCYears	Lemma ShiftUTCYears_valid Lemma ShiftUTCYearsP
25	addFormalYears	AddFormalYears	Depends on AddFormalSeconds
26	timeDifference	TimeDifference	Lemma time_difference_addK Lemma add_time_differenceK Lemma time_difference_subtractK Lemma subtract_time_differenceK
27 - 29	Direct extraction from Coq		
30	subtractFormal	SubtractFormal	Lemma subtract_timestamp

			Lemma subtract_from_formalTime Lemma subtractK Lemma addK
31	from_FormalTime	from_FormalTime	Lemma from_formalTime_nil Lemma from_formalTime_cons Lemma from_formalTimeK
32	to_FormalTime	to_FormalTime	Lemma to_formalTimeK Lemma to_formalTimeP Lemma to_formalTime_not_empty
33; 34	Not available in Coq		
35	date_le	led	Lemma rawDate_leE
36	date_lt	ltd	Lemma rawDate_ltE
37	sort_dates	sort_rawDates	Lemma sort_sorted Lemma mem_sort
38	time_le	let	Lemma rawTime_leE
39	time_lt	ltt	Lemma rawTime_ltE
40	sort_times	sort_rawTimes	Lemma sort_sorted Lemma mem_sort

Appendix A Tables

Year	Jun 30	Dec 31		Year	Jun 30	Dec 31
1972	1	1		1998	0	1
1973	0	1		1999	0	0
1974	0	1		2000	0	0
1975	0	1		2001	0	0
1976	0	1		2002	0	0
1977	0	1		2003	0	0
1978	0	1		2004	0	0
1979	0	1		2005	0	1
1980	0	0		2006	0	0
1981	1	0		2007	0	0
1982	1	0		2008	0	1
1983	1	0		2009	0	0
1984	0	0		2010	0	0
1985	1	0		2011	0	0
1986	0	0		2012	1	0
1987	0	1		2013	0	0
1988	0	0		2014	0	0
1989	0	1		2015	1	0
1990	0	1		2016	0	1
1991	0	0		2017	0	0
1992	1	0		2018	0	0
1993	1	0			Jun 30	Dec 31
1994	1	0		Total	11	16
1995	0	1		27		
1996	0	0		Current TAI − UTC		
1997	1	0		37		

Table 7: Leap seconds as of December 2018.

America/St_Johns	America/Bogota	UTC+0	Europe/Madrid	Europe/Moscow
1972-06-30 21:29:60	1972-06-30 18:59:60	1972-06-30 23:59:60	1972-07-01 00:59:60	1972-07-01 02:59:60
1994-06-30 21:29:60	1994-06-30 18:59:60	1994-06-30 23:59:60	1994-07-01 01:59:60	1994-07-01 03:59:60
2005-12-31 20:29:60	2005-12-31 18:59:60	2005-12-31 23:59:60	2006-01-01 00:59:60	2006-01-01 02:59:60

Table 8: Examples of local leap seconds

A General interpretation of the n-th position the clock can take	B Positions that can be taken by the central element with repetitive movements in the round clock:	C Time format in UTC linked to each position:	D Natural number linked to the position in time real format, that is, accumulated seconds:	E Interpretation of time real concept linked to the position. That is, we can interpret the time real as: How many periodic movements the central element did to reach the corresponding position?
n=1	1 (Twelve o'clock position)	1970/1/1/00:00:00	0	movements to reach the position 1? In this case 0 movements because is the starting point.
n=2	2	1970/1/1/00:00:01	1	movements to reach the position 2? 1 movement.
3	3	1970/1/1/00:00:02	2	2 movements.
4	4	1970/1/1/00:00:03	3	3 movements.
60	60	1970/1/1/00:00:59	59	59 movements.
61	1 (Twelve o'clock position again)	1970/1/1/00:01:00	60	60 movements.
62	2	1970/1/1/00:01:01	61	61 movements.
120	60	1970/1/1/00:01:59	119	119 movements.
121	1	1970/1/1/00:02:00	120	120 movements.
151	31	1970/1/1/00:02:30	150	150 movements.
181	1	1970/1/1/00:03:00	180	180 movements.
241	1	1970/1/1/00:04:00	240	240 movements.
256	16	1970/1/1/00:04:15	255	255 movements.
301	1	1970/1/1/00:05:00	300	300 movements.
86401	1	1970/1/2/00:00:00	86400	86400 movements.

Table 9: For the following table we consider we are in the beginning of our time, we recall this is 1970-01-01 00:00:00.

Appendix B `tz` Tables

Time Zones List

Here we have the list of Time Zones considered. We list them by continents and countries. When in a time zone it is said that is **alias** of another is because they are equivalent. The principal one is marked as **canonical**.

AFRICA				
COUNTRY CODE	COUNTRY NAME	OFFSET WITHOUT DST	TIME ZONE	CATEGORY
[CI]	IVORY COAST	[+0]	Africa/Abidjan	Canonical
[GH]	GHANA	[+0]	Africa/Accra	Canonical
[ET]	ETHIOPIA	[+3]	Africa/Addis_Ababa	alias of Africa/Nairobi
[DZ]	ALGERIA	[+1]	Africa/Algiers	Canonical
[ER]	ERITREA	[+3]	Africa/Asmara	alias of Africa/Nairobi
[ML]	MALI	[+0]	Africa/Bamako	alias of Africa/Abidjan
[CF]	CENTRAL AFRICAN REPUBLIC	[+1]	Africa/Bangui	alias of Africa/Lagos
[GM]	GAMBIA	[+0]	Africa/Banjul	alias of Africa/Abidjan
[GW]	GUINEA-BISSAU	[+0]	Africa/Bissau	Canonical
[MW]	MALAWI	[+2]	Africa/Blantyre	alias of Africa/Maputo
[CG]	CONGO (REPUBLIC OF CONGO)	[+1]	Africa/Brazzaville	alias of Africa/Lagos
[BI]	BURUNDI	[+2]	Africa/Bujumbura	alias of Africa/Maputo
[EG]	EGYPT	[+2]	Africa/Cairo	Canonical
[MA]	MOROCCO	[+1]	Africa/Casablanca	Canonical
[GN]	GUINEA	[+0]	Africa/Conakry	Alias of Africa/Abidjan
[SN]	SENEGAL	[+0]	Africa/Dakar	Alias of Africa/Abidjan
[TZ]	TANZANIA	[+3]	Africa/Dar_es_Salaam	alias of Africa/Nairobi
[DJ]	DJIBOUTI	[+3]	Africa/Djibouti	alias of Africa/Nairobi
[CM]	CAMEROON	[+1]	Africa/Douala	alias of Africa/Lagos
[BW]	BOTSWANA	[+2]	Africa/Gaborone	alias of Africa/Maputo
[ZW]	ZIMBAWE	[+2]	Africa/Harare	alias of Africa/Maputo
[ZA]	SOUTH AFRICA	[+2]	Africa/Johannesburg	Canonical
[SS]	SOUTH SUDAN	[+3]	Africa/Juba	Canonical
[UG]	UGANDA	[+3]	Africa/Kampala	alias of Africa/Nairobi
[SD]	SUDAN	[+2]	Africa/Khartoum	Canonical
[RW]	RWANDA	[+2]	Africa/Kigali	alias of Africa/Maputo
[CD]	CONGO (DEMOCRATIC REPUBLIC OF CONGO)	[+1]	Africa/Kinshasa	alias of Africa/Lagos
[NG]	NIGERIA	[+1]	Africa/Lagos	Canonical
[GA]	GABON	[+1]	Africa/Libreville	alias of Africa/Lagos
[AO]	ANGOLA	[+1]	Africa/Luanda	alias of Africa/Lagos
[CD]	CONGO (DEMOCRATIC REPUBLIC OF CONGO)	[+2]	Africa/Lubumbashi	alias of Africa/Maputo
[ZM]	ZAMBIA	[+2]	Africa/Lusaka	alias of Africa/Maputo
[GQ]	EQUATORIAL GUINEA	[+1]	Africa/Malabo	alias of Africa/Lagos
[MZ]	MOZAMBIQUE	[+2]	Africa/Maputo	Canonical
[LS]	LESOTHO	[+2]	Africa/Maseru	alias of Africa/Johannesburg
[SZ]	SWAZILAND	[+2]	Africa/Mbabane	alias of Africa/Johannesburg
[SO]	SOMALIA	[+3]	Africa/Mogadishu	alias of Africa/Nairobi
[LR]	LIBERIA	[+0]	Africa/Monrovia	Canonical

AFRICA continuation				
COUNTRY CODE	COUNTRY NAME	OFFSET WITHOUT DST	TIME ZONE	CATEGORY
[KE]	KENYA	[+3]	Africa/Nairobi	Canonical
[TD]	CHAD	[+1]	Africa/Ndjamena	Canonical
[NE]	NIGER	[+1]	Africa/Niamey	alias of Africa/Lagos
[BJ]	BENIN	[+1]	Africa/Porto-Novo	alias of Africa/Lagos
[ST]	SAO TOME AND PRINCIPE	[+1]	Africa/Sao_Tome	alias of Africa/Lagos
[LY]	LIBYA	[+2]	Africa/Tripoli	Canonical
[TN]	TUNISIA	[+1]	Africa/Tunis	Canonical
[NA]	NAMIBIA	[+1]	Africa/Windhoek	Canonical
[CV]	CABO VERDE	[-1]	Atlantic/Cape_Verde	Canonical
[SH]	SAINT HELENA, ASCENSION AND TRISTAN DA CUNHA	[+0]	Atlantic/St_Helena	alias of Africa/Abidjan
[MG]	MADAGASCAR	[+3]	Indian/Antananarivo	alias of Africa/Nairobi
[KM]	COMOROS	[+3]	Indian/Comoro	alias of Africa/Nairobi
[TF]	FRENCH SOUTHERN AND ANTARTIC ISLANDS	[+5]	Indian/Kerguelen	Canonical
[SC]	SEYCHELLES	[+4]	Indian/Mahe	Canonical
[MU]	MAURITIUS	[+4]	Indian/Mauritius	Canonical
[YT]	MAYOTTE	[+3]	Indian/Mayotte	alias of Africa/Nairobi
[RE]	RÉUNION	[+4]	Indian/Reunion	Canonical

AMERICA				
COUNTRY CODE	COUNTRY NAME	OFFSET WITHOUT DST	TIME ZONE	CATEGORY
[US]	USA (ALEUTIAN ISLANDS)	[-10]	America/Adak	Canonical
[US]	USA (ALASKA)	[-9]	America/Anchorage	Canonical
[AI]	ANGUILLA	[-4]	America/Anguilla	alias of America/Port_of_Spain
[AG]	ANTIGUA AND BARBUDA	[-4]	America/Antigua	alias of America/Port_of_Spain
[BR]	BRAZIL (STATE OF TOCANTINS)	[-3]	America/Araguaina	Canonical
[AR]	ARGENTINA	[-3]	America/Argentina/Buenos_Aires	Canonical
[AR]	ARGENTINA	[-3]	America/Argentina/Catamarca	Canonical
[AR]	ARGENTINA	[-3]	America/Argentina/ComodRivadavia	alias of Catamarca
[AR]	ARGENTINA	[-3]	America/Argentina/Cordoba	Canonical
[AR]	ARGENTINA	[-3]	America/Argentina/Jujuy	Canonical
[AR]	ARGENTINA	[-3]	America/Argentina/La_Rioja	Canonical
[AR]	ARGENTINA	[-3]	America/Argentina/Mendoza	Canonical
[AR]	ARGENTINA	[-3]	America/Argentina/Rio_Gallegos	Canonical
[AR]	ARGENTINA	[-3]	America/Argentina/Salta	Canonical
[AR]	ARGENTINA	[-3]	America/Argentina/San_Juan	Canonical
[AR]	ARGENTINA	[-3]	America/Argentina/San_Luis	Canonical
[AR]	ARGENTINA	[-3]	America/Argentina/Tucuman	Canonical
[AR]	ARGENTINA	[-3]	America/Argentina/Ushuaia	Canonical
[AW]	ARUBA	[-4]	America/Aruba	alias of America/Curasao
[PY]	PARAGUAY	[-4]	America/Asuncion	Canonical
[CA]	CANADA	[-5]	America/Atikokan	Canonical
[US]	USA (ALASKA)	[-10]	America/Atka	alias of America/Adak
[BR]	BRAZIL (BAHIA)	[-3]	America/Bahia	Canonical
[MX]	MEXICO (CENTRAL TIME)	[-6]	America/Bahia_Banderas	Canonical
[BB]	BARBADOS	[-4]	America/Barbados	Canonical
[BR]	BRAZIL (AMAPA)	[-3]	America/Belem	Canonical
[BZ]	BELIZE (BAHIA)	[-6]	America/Belize	Canonical
[CA]	CANADA	[-4]	America/Blanc-Sablon	Canonical
[BR]	BRAZIL (RORAIMA)	[-4]	America/Boa_Vista	Canonical
[CO]	COLOMBIA	[-6]	America/Bogota	Canonical
[US]	USA	[-7]	America/Boise	Canonical
[AR]	ARGENTINA	[-3]	America/Buenos_Aires	alias of America/Argentina/Buenos_Aires
[CA]	CANADA	[-7]	America/Cambridge_Bay	Canonical
[BR]	BRAZIL (MATO GROSSO DO SUL)	[-4]	America/Campo_Grande	Canonical
[MX]	MEXICO (EASTERN STANDARD TIME)	[-5]	America/Cancun	Canonical
[VE]	VENEZUELA	[-4]	America/Caracas	Canonical
[AR]	ARGENTINA	[-3]	America/Catamarca	alias of America/Argentina/Catamarca
[GF]	FRENCH GUIANA	[-3]	America/Cayenne	Canonical
[KY]	CAYMAN ISLANDS	[-5]	America/Cayman	alias of America/Panama
[US]	USA (CENTRAL)	[-6]	America/Chicago	Canonical

AMERICA continuation				
COUNTRY CODE	COUNTRY NAME	OFFSET WITHOUT DST	TIME ZONE	CATEGORY
[MX]	MEXICO (MOUNTAIN TIME/CHIHUAHUA)	[-7]	America/Chihuahua	Canonical
[CA]	CANADA	[-5]	America/Coral_Harbour	alias of America/Atikokan
[AR]	ARGENTINA	[-3]	America/Cordoba	alias of America/Argentina/Cordoba
[CR]	COSTA RICA	[-6]	America/Costa_Rica	Canonical
[CA]	CANADA	[-7]	America/Creston	Canonical
[BR]	BRAZIL (MATO GROSSO)	[-3]	America/Cuiaba	Canonical
[CW]	CURAÇAO	[-4]	America/Curacao	Canonical
[GL]	GREENLAND	[+0]	America/Danmarkshavn	Canonical
[CA]	CANADA	[-8]	America/Dawson	Canonical
[CA]	CANADA	[-7]	America/Dawson_Creek	Canonical
[US]	USA (MOUNTAIN)	[-7]	America/Denver	Canonical
[US]	USA (EASTERN)	[-5]	America/Detroit	Canonical
[DM]	DOMINICA	[-4]	America/Dominica	alias of America/Port_of_Spain
[CA]	CANADA	[-7]	America/Edmonton	Canonical
[BR]	BRAZIL (AMAZONAS)	[-5]	America/Eirunepe	Canonical
[SV]	EL SALVADOR	[-6]	America/El_Salvador	Canonical
[MX]	MEXICO	[-8]	America/Ensenada	alias of America/Tijuana
[CA]	CANADA	[-7]	America/Fort_Nelson	Canonical
[US]	USA (EASTERN-IN)	[-5]	America/Fort_Wayne	alias of America/Indiana/Indianapolis
[BR]	BRAZIL (NORTHEAST)	[-3]	America/Fortaleza	Canonical
[CA]	CANADA	[-4]	America/Glace_Bay	Canonical
[GL]	GREENLAND	[-3]	America/Godthab	Canonical
[CA]	CANADA	[-4]	America/Goose_Bay	Canonical
[TC]	TURKS AND CAICOS	[-5]	America/Grand_Turk	Canonical
[GD]	GRENADA	[-4]	America/Grenada	alias of Europe/Port_of_Spain
[GP]	GUADELOUPE	[-4]	America/Guadeloupe	alias of Europe/Port_of_Spain
[GT]	GUATEMALA	[-6]	America/Guatemala	Canonical
[EC]	ECUADOR	[-5]	America/Guayaquil	Canonical
[GY]	GUYANA	[-4]	America/Guyana	Canonical
[CA]	CANADA	[-4]	America/Halifax	Canonical
[CU]	CUBA	[-5]	America/Havana	Canonical
[MX]	MEXICO (MOUNTAIN STANDARD TIME/SONORA)	[-7]	America/Hermosillo	Canonical
[US]	USA (EASTERN-IN)	[-5]	America/Indiana/Indianapolis	Canonical
[US]	USA (CENTRAL-IN/STARKE)	[-6]	America/Indiana/Knox	Canonical
[US]	USA (EASTERN-IN/CRAWFORD)	[-5]	America/Indiana/Marengo	Canonical
[US]	USA (EASTERN-IN/PIKE)	[-5]	America/Indiana/Petersburg	Canonical
[US]	USA (CENTRAL-IN/PERRY)	[-6]	America/Indiana/Tell_City	Canonical
[US]	USA (EASTERN-IN/SWITZERLAND)	[-5]	America/Indiana/Vevay	Canonical

AMERICA continuation				
COUNTRY CODE	COUNTRY NAME	OFFSET WITHOUT DST	TIME ZONE	CATEGORY
[US]	USA (EASTERN-IN)	[-5]	America/Indiana/Vincennes	Canonical
[US]	USA (EASTERN-IN)	[-5]	America/Indiana/Winamac	Canonical
[US]	USA (EASTERN-IN)	[-5]	America/Indianapolis	alias of America/Indiana/Indianapolis
[CA]	CANADA	[-7]	America/Inuvik	Canonical
[CA]	CANADA	[-5]	America/Iqaluit	Canonical
[JM]	JAMAICA	[-5]	America/Jamaica	Canonical
[AR]	ARGENTINA	[-3]	America/Jujuy	alias of America/Argentina/Jujuy
[US]	USA (ALASKA)	[-9]	America/Juneau	Canonical
[US]	USA (EASTERN-KY/LOUISVILLE)	[-5]	America/Kentucky/Louisville	Canonical
[US]	USA (EASTERN-KY/WAYNE)	[-5]	America/Kentucky/Monticello	Canonical
[US]	USA	[-6]	America/Knox_IN	alias of America/Indiana/Knox
[BQ]	CARIBBEAN NETHERLANDS	[-4]	America/Kralendijk	alias of America/Curasao
[BO]	BOLIVIA	[-4]	America/La_Paz	Canonical
[PE]	PERU	[-5]	America/Lima	Canonical
[US]	USA (PACIFIC)	[-8]	America/Los_Angeles	Canonical
[US]	USA	[-5]	America/Louisville	alias of America/Kentucky/Louisville
[SX]	SINT MAARTEN	[-4]	America/Lower_Princes	alias of America/Curacao
[BR]	BRAZIL (ALAGOAS, SERGIPE)	[-3]	America/Maceio	Canonical
[NI]	NICARAGUA	[-6]	America/Managua	Canonical
[BR]	BRAZIL (AMAZONAS EAST)	[-4]	America/Manaus	Canonical
[MF]	SAINT MARTIN	[-4]	America/Marigot	alias of America/Port_of_Spain
[MQ]	MARTINIQUE	[-4]	America/Martinique	Canonical
[MX]	MEXICO (CENTRAL TIME/US)	[-6]	America/Matamoros	Canonical
[MX]	MEXICO (MOUNTAIN TIME/BAJA CALIFORNIA)	[-7]	America/Mazatlan	Canonical
[AR]	ARGENTINA	[-3]	America/Mendoza	alias of America/Argentina/Mendoza
[US]	USA (CENTRAL-MI/WISCONSIN)	[-6]	America/Menominee	Canonical
[MX]	MEXICO (CENTRAL TIME/YUCATAN)	[-6]	America/Merida	Canonical
[US]	USA (ALASKA)	[-9]	America/Metlakatla	Canonical
[MX]	MEXICO (CENTRAL TIME)	[-6]	America/Mexico_City	Canonical
[PM]	SAINT PIERRE AND MIQUELON	[-3]	America/Miquelon	Canonical
[CA]	CANADA	[-4]	America/Moncton	Canonical
[MX]	MEXICO (CENTRAL TIME/DURANGO)	[-6]	America/Monterrey	Canonical
[UY]	URUGUAY	[-3]	America/Montevideo	Canonical
[CA]	CANADA	[-5]	America/Montreal	alias of America/Toronto
[MS]	MONTSERRAT	[-4]	America/Montserrat	alias of America/Port_of_Spain
[BS]	BAHAMAS	[-5]	America/Nassau	Canonical
[US]	USA (EASTERN)	[-5]	America/New_York	Canonical
[CA]	CANADA	[-5]	America/Nipigon	Canonical
[US]	USA (ALASKA)	[-9]	America/Nome	Canonical
[BR]	BRAZIL (ATLANTIC ISLANDS)	[-2]	America/Noronha	Canonical

AMERICA continuation				
COUNTRY CODE	COUNTRY NAME	OFFSET WITHOUT DST	TIME ZONE	CATEGORY
[US]	USA (CENTRAL-ND/MERCER)	[-6]	America/North_Dakota/Beulah	Canonical
[US]	USA (CENTRAL-ND/OLIVER)	[-6]	America/North_Dakota/Center	Canonical
[US]	USA (CENTRAL-ND/MORTON RURAL)	[-6]	America/North_Dakota/New_Salem	Canonical
[MX]	MEXICO (MOUNTAIN TIME/CHIHUAHUA)	[-7]	America/Ojinaga	Canonical
[PA]	PANAMA	[-5]	America/Panama	Canonical
[CA]	CANADA	[-5]	America/Pangnirtung	Canonical
[SR]	SURINAME	[-3]	America/Paramaribo	Canonical
[US]	USA (ARIZONA)	[-7]	America/Phoenix	Canonical
[HT]	HAITI	[-5]	America/Port-au-Prince	Canonical
[TT]	TRINIDAD AND TOBAGO	[-4]	America/Port_of_Spain	Canonical
[BR]	BRAZIL	[-5]	America/Porto_Acre	alias of America/Rio_Branco
[BR]	BRAZIL (RONDONIA)	[-4]	America/Porto_Velho	Canonical
[PR]	PUERTO RICO	[-4]	America/Puerto_Rico	Canonical
[CL]	CHILE (MAGALLANES)	[-3]	America/Punta_Arenas	Canonical
[CA]	CANADA	[-6]	America/Rainy_River	Canonical
[CA]	CANADA	[-6]	America/Rankin_Inlet	Canonical
[BR]	BRAZIL (PERNAMBUCO)	[-3]	America/Recife	Canonical
[CA]	CANADA	[-6]	America/Regina	Canonical
[CA]	CANADA	[-6]	America/Resolute	Canonical
[BR]	BRAZIL (ACRE)	[-5]	America/Rio_Branco	Canonical
[AR]	ARGENTINA	[-3]	America/Rosario	alias of America/Argentina/Cordoba
[MX]	MEXICO	[-8]	America/Santa_Isabel	alias of America/Tijuana
[BR]	BRAZIL (PARA)	[-3]	America/Santarem	Canonical
[CL]	CHILE	[-4]	America/Santiago	Canonical
[DO]	DOMINICAN REPUBLIC	[-4]	America/Santo_Domingo	Canonical
[BR]	BRAZIL (SOUTHEAST)	[-3]	America/Sao_Paulo	Canonical
[GL]	GREENLAND	[-1]	America/Scoresbysund	Canonical
[US]	USA	[-7]	America/Shiprock	alias of America/Denver
[US]	USA (ALASKA)	[-9]	America/Sitka	Canonical
[BL]	SAINT BARTHELEMY	[-4]	America/St_Barthelemy	alias of America/Port_of_Spain
[CA]	CANADA	[-3:30]	America/St_Johns	Canonical
[KN]	SAINT KITTS AND NEVY	[-4]	America/St_Kitts	alias of America/Port_of_Spain
[LC]	SAINT LUCIA	[-4]	America/St_Lucia	alias of America/Port_of_Spain
[VI]	VIRGIN ISLANDS OF USA	[-4]	America/St_Thomas	alias of America/Port_of_Spain
[VC]	SAINT VINCENT	[-4]	America/St_Vincent	alias of America/Port_of_Spain
[CA]	CANADA	[-6]	America/Swift_Current	Canonical
[HN]	HONDURAS	[-6]	America/Tegucigalpa	Canonical
[GL]	GREENLAND	[-4]	America/Thule	Canonical
[CA]	CANADA	[-5]	America/Thunder_Bay	Canonical

AMERICA continuation				
COUNTRY CODE	COUNTRY NAME	OFFSET WITHOUT DST	TIME ZONE	CATEGORY
[MX]	MEXICO (PACIFIC TIME/BAJA CALIFORNIA)	[-8]	America/Tijuana	Canonical
[CA]	CANADA	[-5]	America/Toronto	Canonical
[VG]	VIRGIN ISLANDS	[-4]	America/Tortola	alias of America/Port_of_Spain
[CA]	CANADA	[-8]	America/Vancouver	Canonical
[CA]	CANADA	[-8]	America/Whitehorse	Canonical
[CA]	CANADA	[-6]	America/Winnipeg	Canonical
[US]	USA (ALASKA)	[-9]	America/Yakutat	Canonical
[CA]	CANADA	[-7]	America/Yellowknife	Canonical
[BM]	BERMUDA	[-4]	Atlantic/Bermuda	Canonical
[GS]	SOUTH GEORGIA AND SOUTH SANDWICH ISLANDS	[-2]	Atlantic/South_Georgia	Canonical
[FK]	FALKAN ISLANDS	[-3]	Atlantic/Stanley	Canonical
[CL]	CHILE (EASTER ISLANDS)	[-6]	Pacific/Easter	Canonical
[EC]	ECUADOR	[-6]	Pacific/Galapagos	Canonical
[GU]	GUAM	[+10]	Pacific/Guam	Canonical
[US]	USA (HAWAII)	[-10]	Pacific/Honolulu	Canonical
[US]	USA (HAWAII)	[-10]	Pacific/Johnston	alias of Pacific/Honolulu
[UM]	US MINOR OUTLAYING ISLANDS	[-11]	Pacific/Midway	Link to Pacific/Pago_Pago
[AS]	AMERICAN SAMOA	[-11]	Pacific/Pago_Pago	Canonical
[MP]	NORTHERN MARIANA ISLANDS	[+10]	Pacific/Saipan	alias of Pacific/Guam
[AS]	AMERICAN SAMOA	[-11]	Pacific/Samoa	alias of Pacific/Pago_Pago
[UM]	US MINOR OUTLAYING ISLANDS	[+12]	Pacific/Wake	Canonical

ANTARCTICA				
COUNTRY CODE	COUNTRY NAME	OFFSET WITHOUT DST	TIME ZONE	CATEGORY
[AQ]	ANTARCTICA	[+11]	Antarctica/Casey	Canonical
[AQ]	ANTARCTICA	[+7]	Antarctica/Davis	Canonical
[AQ]	ANTARCTICA	[+10]	Antarctica/DumontDUrville	Canonical
[AQ]	ANTARCTICA	[+11]	Antarctica/Macquarie	Canonical
[AQ]	ANTARCTICA	[+5]	Antarctica/Mawson	Canonical
[AQ]	ANTARCTICA	[+12]	Antarctica/McMurdo	alias of Pacific/Auckland
[AQ]	ANTARCTICA	[-3]	Antarctica/Palmer	Canonical
[AQ]	ANTARCTICA	[-3]	Antarctica/Rothera	Canonical
[AQ]	ANTARCTICA	[+12]	Antarctica/South_Pole	alias of Pacific/Auckland
[AQ]	ANTARCTICA	[+3]	Antarctica/Syowa	Canonical
[AQ]	ANTARCTICA	[+0]	Antarctica/Troll	Canonical
[AQ]	ANTARCTICA	[+6]	Antarctica/Vostok	Canonical

ASIA				
COUNTRY CODE	COUNTRY NAME	OFFSET WITHOUT DST	TIME ZONE	CATEGORY
[YE]	YEMEN	[+3]	Asia/Aden	alias of Asia/Riyahd
[KZ]	KAZAKHSTAN	[+6]	Asia/Almaty	Canonical
[JO]	JORDAN	[+2]	Asia/Amman	Canonical
[RU]	RUSSIA	[+12]	Asia/Anadyr	Canonical
[KZ]	KAZAKHSTAN	[+5]	Asia/Aqtau	Canonical
[KZ]	KAZAKHSTAN	[+5]	Asia/Aqtobe	Canonical
[TM]	TURKMENISTAN	[+5]	Asia/Ashgabat	Canonical
[KZ]	KAZAKHSTAN	[+5]	Asia/Atyrau	Canonical
[IQ]	IRAQ	[+3]	Asia/Baghdad	Canonical
[BH]	BAHRAIN	[+3]	Asia/Bahrain	Canonical
[AZ]	AZERBAIJAN	[+4]	Asia/Baku	Canonical
[TH]	THAILAND	[+7]	Asia/Bangkok	Canonical
[RU]	RUSSIA	[+7]	Asia/Barnaul	Canonical
[LB]	LEBANON	[+2]	Asia/Beirut	Canonical
[KG]	KYRGYSTAN	[+6]	Asia/Bishkek	Canonical
[BN]	BRUNEI	[+8]	Asia/Brunei	Canonical
[IN]	INDIA	[+5:30]	Asia/Calcutta	alias of Asia/Kolkata
[RU]	RUSSIA	[+9]	Asia/Chita	Canonical
[MN]	MONGOLIA	[+8]	Asia/Choibalsan	Canonical
[CN]	CHINA	[+8]	Asia/Chongqing	alias of Asia/Sanghai
[CN]	CHINA	[+8]	Asia/Chungking	alias of Asia/Sanghai
[LK]	SRI LANKA	[+5:30]	Asia/Colombo	Canonical
[BD]	BANGLADESH	[+6]	Asia/Dacca	alias of Asia/Dhaka
[TM]	TURKMENISTAN	[+5]	Asia/Damascus	Canonical
[BD]	BANGLADESH	[+6]	Asia/Dhaka	Canonical
[TL]	EAST TIMOR	[+9]	Asia/Dili	Canonical
[AE]	EMIRATES	[+5]	Asia/Dubai	Canonical
[TJ]	TAJIKISTAN	[+5]	Asia/Dushanbe	Canonical
[CY]	CYPRUS	[+2]	Asia/Famagusta	Canonical
[PS]	PALESTINE	[+2]	Asia/Gaza	Canonical
[CN]	CHINA	[+8]	Asia/Harbin	alias of Asia/Sanghai
[PS]	PALESTINE	[+2]	Asia/Hebron	Canonical
[VN]	VIETNAM	[+7]	Asia/Ho_Chi_Minh	Canonical
[HK]	HONG KONG	[+8]	Asia/Hong_Kong	Canonical
[MN]	MONGOLIA	[+7]	Asia/Hovd	Canonical
[RU]	RUSSIA	[+8]	Asia/Irkutsk	Canonical
[TR]	TURKEY	[+3]	Asia/Istanbul	alias of Europe/Istanbul
[ID]	INDONESIA	[+7]	Asia/Jakarta	Canonical
[ID]	INDONESIA	[+9]	Asia/Jayapura	Canonical

ASIA continuation				
COUNTRY CODE	COUNTRY NAME	OFFSET WITHOUT DST	TIME ZONE	CATEGORY
[IL]	ISRAEL	[+2]	Asia/Jerusalem	Canonical
[AF]	AFGHANISTAN	[+4:30]	Asia/Kabul	Canonical
[RU]	RUSSIA	[+12]	Asia/Kamchatka	Canonical
[PK]	PAKISTAN	[+5]	Asia/Karachi	Canonical
[CN]	CHINA	[+6]	Asia/Kashgar	Canonical
[NP]	NEPAL	[+5:45]	Asia/Kathmandu	Canonical
[NP]	NEPAL	[+5:45]	Asia/Katmandu	alias of Asia/Kathmandu
[SY]	SYRIA	[+2]	Asia/Katmandu	alias of Asia/Sanghai
[RU]	RUSSIA	[+9]	Asia/Khandyga	Canonical
[IN]	INDIA	[+5:30]	Asia/Kolkata	Canonical
[RU]	RUSSIA	[+7]	Asia/Krasnoyarsk	Canonical
[MY]	MALASYA	[+8]	Asia/Kuala_Lumpur	Canonical
[MY]	MALASYA	[+8]	Asia/Kuching	Canonical
[KW]	KWAIT	[+3]	Asia/Kuwait	alias of Asia/Riyahd
[MO]	MACAU	[+8]	Asia/Macao	alias of Asia/Macau
[MO]	MACAU	[+8]	Asia/Macau	Canonical
[RU]	RUSSIA	[+11]	Asia/Magadan	Canonical
[ID]	INDONESIA	[+8]	Asia/Makassar	Canonical
[PH]	PHILIPPINES	[+8]	Asia/Manila	Canonical
[OM]	OMAN	[+4]	Asia/Muscat	alias of Asia/Dubai
[RU]	RUSSIA	[+7]	Asia/Novokuznetsk	Canonical
[RU]	RUSSIA	[+7]	Asia/Novosibirsk	Canonical
[RU]	RUSSIA	[+6]	Asia/Omsk	Canonical
[KZ]	KAZAKHSTAN	[+5]	Asia/Oral	Canonical
[KH]	CAMBODIA	[+7]	Asia/Phnom_Penh	alias of Asia/Bangkok
[ID]	INDONESIA	[+7]	Asia/Pontianak	Canonical
[KP]	NORTH COREA	[+9]	Asia/Pyongyang	Canonical
[QA]	QATAR	[+3]	Asia/Qatar	Canonical
[KZ]	KAZAKHSTAN	[+5]	Asia/Qyzylorda	Canonical
[MM]	MYANMAR	[+6:30]	Asia/Rangoon	alias of Asia/Yangon
[SA]	SAUDI ARABIA	[+3]	Asia/Riyadh	Canonical
[VN]	VIETNAM	[+7]	Asia/Saigon	alias of Asia/Ho_Chi_Minh
[RU]	RUSSIA	[+11]	Asia/Sakhalin	Canonical
[UZ]	UZBEKISTAN	[+5]	Asia/Samarkand	Canonical
[KR]	SOUTH KOREA	[+9]	Asia/Seoul	Canonical
[CN]	CHINA	[+8]	Asia/Shanghai	Canonical
[SG]	SINGAPORE	[+8]	Asia/Singapore	Canonical
[RU]	RUSSIA	[+11]	Asia/Srednekolymsk	Canonical
[TW]	TAIWAN	[+8]	Asia/Taipei	Canonical
[UZ]	UZBEKISTAN	[+5]	Asia/Tashkent	Canonical

ASIA continuation				
COUNTRY CODE	COUNTRY NAME	OFFSET WITHOUT DST	TIME ZONE	CATEGORY
[GE]	GEORGIA	[+4]	Asia/Tbilisi	Canonical
[IR]	IRAN	[+3:30]	Asia/Tehran	Canonical
[IL]	ISRAEL	[+2]	Asia/Tel_Aviv	alias of Asia/Jerusalem
[BT]	BUTHAN	[+3:30]	Asia/Thimbu	alias of Asia/Thimphu
[BT]	BUTHAN	[+3:30]	Asia/Thimphu	Canonical
[JP]	JAPAN	[+9]	Asia/Tokyo	Canonical
[RU]	RUSSIA	[+7]	Asia/Tomsk	Canonical
[ID]	INDONESIA	[+8]	Asia/Ujung_Pandang	alias of Asia/Makassar
[MN]	MONGOLIA	[+8]	Asia/Ulaanbaatar	Canonical
[MN]	MONGOLIA	[+8]	Asia/Ulan_Bator	alias of Asia/Ulaanbaatar
[CN]	CHINA	[+6]	Asia/Urumqi	Canonical
[RU]	RUSSIA	[+10]	Asia/Ust-Nera	Canonical
[LA]	LAOS	[+7]	Asia/Vientiane	Canonical
[RU]	RUSSIA	[+10]	Asia/Vladivostok	Canonical
[RU]	RUSSIA	[+9]	Asia/Yakutsk	Canonical
[MM]	MYANMAR	[+6:30]	Asia/Yangon	Canonical
[RU]	RUSSIA	[+5]	Asia/Yekaterinburg	Canonical
[AM]	ARMENIA	[+4]	Asia/Yerevan	Canonical
[MV]	MALDIVES	[+5]	Indian/Maldives	Canonical

EUROPE				
COUNTRY CODE	COUNTRY NAME	OFFSET WITHOUT DST	TIME ZONE	CATEGORY
[ES]	SPAIN (CEUTA, MELILLA)	[+1]	Africa/Ceuta	Canonical
[PT]	PORTUGAL (AZORES)	[-1]	Atlantic/Azores	Canonical
[BM]	BERMUDA	[-4]	Atlantic/Bermuda	Canonical
[ES]	SPAIN (CANARY ISLANDS)	[+0]	Atlantic/Canary	Canonical
[FO]	FAROE ISLANDS	[+1]	Atlantic/Faeroe	alias of Atlantic/Faroe
[FO]	FAROE ISLANDS	[+1]	Atlantic/Faroe	Canonical
[NO]	NORWAY	[+1]	Atlantic/Jan_Mayen	alias of Europe/Oslo
[PT]	PORTUGAL (MADEIRA)	[+0]	Atlantic/Madeira	Canonical
[IS]	ICELAND	[+0]	Atlantic/Reykjavik	Canonical
[GS]	SOUTH GEORGIA AND SOUTH SANDWICH ISLANDS	[-2]	Atlantic/South_Georgia	Canonical
[SH]	SAINT HELENA, ASCENSION AND TRISTAN DA CUNHA	[+0]	Atlantic/St_Helena	alias of Africa/Abidjan
[FK]	FALKAN ISLANDS	[-3]	Atlantic/Stanley	Canonical
[NL]	NETHERLANDS	[+1]	Europe/Amsterdam	Canonical
[AD]	ANDORRA	[+1]	Europe/Andorra	Canonical
[RU]	RUSSIA	[+4]	Europe/Astrakhan	Canonical
[GR]	GREECE	[+1]	Europe/Athens	Canonical
[GB]	UNITED KINGDOM	[+0]	Europe/Belfast	Alias of Europe/London
[RS]	SERBIA	[+1]	Europe/Belgrade	Canonical
[DE]	GERMANY	[+1]	Europe/Berlin	Canonical
[SK]	SLOVAKIA	[+1]	Europe/Bratislava	Canonical
[BE]	BELGIUM	[+1]	Europe/Brussels	Canonical
[RO]	ROMANIA	[+2]	Europe/Bucharest	Canonical
[HU]	HUNGARY	[+1]	Europe/Budapest	Canonical
[DE]	GERMANY	[+1]	Europe/Busingen	Alias of Europe/Zurich
[MD]	MOLDOVA	[+2]	Europe/Chisinau	Canonical
[DK]	DENMARK	[+1]	Europe/Copenhagen	Canonical
[IE]	IRELAND	[+1]	Europe/Dublin	Canonical
[GI]	GIBRALTAR	[+1]	Europe/Gibraltar	Canonical
[GG]	GUENRSEY	[+0]	Europe/Guernsey	Alias of Europe/London
[FI]	FINLAND	[+2]	Europe/Helsinki	Canonical
[IM]	ISLE OF MAN	[+0]	Europe/Isle_of_Man	Alias of Europe/London
[TR]	TURKEY	[+3]	Europe/Istanbul	Canonical
[JE]	JERSEY	[+0]	Europe/Jersey	Alias of Europe/London
[RU]	RUSSIA	[+2]	Europe/Kaliningrad	Canonical
[UA]	UKRAINE	[+2]	Europe/Kiev	Canonical
[RU]	RUSSIA	[+3]	Europe/Kirov	Canonical
[PT]	PORTUGAL	[+0]	Europe/Lisbon	Canonical
[SI]	SLOVENIA	[+1]	Europe/Ljubljana	Canonical
[GB]	UNITED KINGDOM	[+0]	Europe/London	Canonical
[LU]	LUXEMBURG	[+1]	Europe/Luxembourg	Canonical

EUROPE continuation				
COUNTRY CODE	COUNTRY NAME	OFFSET WITHOUT DST	TIME ZONE	CATEGORY
[ES]	SPAIN	[+1]	Europe/Madrid	Canonical
[MT]	MALTA	[+1]	Europe/Malta	Canonical
[AX]	ALAND ISLANDS	[+2]	Europe/Mariehamn	Alias of Europe/Helsinki
[BY]	BELARUS	[+3]	Europe/Minsk	Canonical
[MC]	MONACO	[+1]	Europe/Monaco	Canonical
[RU]	RUSSIA	[+3]	Europe/Moscow	Canonical
[CY]	CYPRUS	[+2]	Europe/Nicosia	Canonical
[NO]	NORWAY	[+1]	Europe/Oslo	Canonical
[FR]	FRANCE	[+1]	Europe/Paris	Canonical
[ME]	MONTENEGRO	[+1]	Europe/Podgorica	Alias of Europe/Belgrade
[CZ]	CZECH REPUBLIC	[+1]	Europe/Prague	Canonical
[LV]	LATVIA	[+2]	Europe/Riga	Canonical
[IT]	ITALY	[+1]	Europe/Rome	Canonical
[RU]	RUSSIA	[+4]	Europe/Samara	Canonical
[SM]	SAN MARINO	[+1]	Europe/San_Marino	Canonical
[BA]	BOSNIA AND HERZEGOVINA	[+1]	Europe/Sarajevo	Canonical
[RU]	RUSSIA	[+4]	Europe/Saratov	Canonical
[UA]	UKRAINE (CRIMEA)	[+3]	Europe/Simferopol	Canonical
[MK]	NORTH MACEDONIA	[+1]	Europe/Skopje	Alias of Europe/Belgrade
[BG]	BULGARIA	[+2]	Europe/Sofia	Canonical
[SE]	SWEDEN	[+1]	Europe/Stockholm	Canonical
[EE]	ESTONIA	[+2]	Europe/Tallinn	Canonical
[AL]	ALBANIA	[+1]	Europe/Tirane	Canonical
[MD]	MOLDOVA	[+2]	Europe/Tiraspol	Alias of Europe/Chisinau
[RU]	RUSSIA	[+4]	Europe/Ulyanovsk	Canonical
[UA]	UKRAINE	[+2]	Europe/Uzhgorod	Canonical
[LI]	LIETCHESTEIN	[+1]	Europe/Vaduz	Alias of Europe/Zurich
[VA]	VATICAN CITY	[+1]	Europe/Vatican	Alias of Europe/Rome
[AT]	AUSTRIA	[+1]	Europe/Vienna	Canonical
[LT]	LITHUANIA	[+2]	Europe/Vilnius	Canonical
[RU]	RUSSIA	[+4]	Europe/Volgograd	Canonical
[PL]	POLAND	[+1]	Europe/Warsaw	Canonical
[HR]	CROATIA	[+1]	Europe/Zagreb	Alias of Europe/Belgrade
[UA]	UKRAINE	[+2]	Europe/Zaporozhye	Canonical
[CH]	SWITZERLAND	[+1]	Europe/Zurich	Canonical

EUROPE continuation

COUNTRY CODE	COUNTRY NAME	OFFSET WITHOUT DST	TIME ZONE	CATEGORY
[TF]	FRENCH SOUTHERN AND ANTARTIC ISLANDS	[+5]	Indian/Kerguelen	Canonical
[RE]	RÉUNION	[+4]	Indian/Reunion	Canonical
[PF]	FRENCH POLINESIA	[-9]	Pacific/Gambier	Canonical
[PF]	FRENCH POLINESIA	[-9:30]	Pacific/Marquesas	Canonical
[NC]	NEW CALEDONIA	[+11]	Pacific/Noumea	Canonical
[PN]	PITCAIRN ISLANDS	[-8]	Pacific/Pitcairn	Canonical
[PF]	FRENCH POLINESIA	[-10]	Pacific/Tahiti	Canonical
[WF]	WALLIS AND FUTUNA	[+12]	Pacific/Wallis	Canonical

OCEANIA				
COUNTRY CODE	COUNTRY NAME	OFFSET WITHOUT DST	TIME ZONE	CATEGORY
[AU]	AUSTRALIA (QUEENSLAND, MOST TERRITORIES)	[+10]	Australia/Brisbane	Canonical
[AU]	AUSTRALIA (NEW SOUTH WALES)	[+9:30]	Australia/Broken_Hill	Canonical
[AU]	AUSTRALIA	[+10]	Australia/Canberra	alias of Australia/Sydney
[AU]	AUSTRALIA (TASMANIA KING ISLAND)	[+9:30]	Australia/Currie	Canonical
[AU]	AUSTRALIA (NORTHERN TERRITORY)	[+9:30]	Australia/Darwin	Canonical
[AU]	AUSTRALIA (WESTERN AUSTRALIA)	[+8:45]	Australia/Eucla	Canonical
[AU]	AUSTRALIA (TASMANIA)	[+10]	Australia/Hobart	Canonical
[AU]	AUSTRALIA (QUEENSLAND, WHITSUNDAY ISLANDS)	[+10]	Australia/Lindeman	Canonical
[AU]	AUSTRALIA (SOUTH)	[+10:30]	Australia/Lord_Howe	Canonical
[AU]	AUSTRALIA (VICTORIA)	[+10]	Australia/Melbourne	Canonical
[AU]	AUSTRALIA (WESTERN AUSTRALIA)	[+8]	Australia/Perth	Canonical
[AU]	AUSTRALIA (NEW SOUTH WALES)	[+10]	Australia/Sydney	Canonical
[AU]	AUSTRALIA (SOUTH)	[+9:30]	Australia/Yancowinna	of alias Australia/Broken_Hill
[IO]	BRITISH INDIAN OCEAN TERRITORY (BIOT)	[+6]	Indian/Chagos	Canonical
[CX]	CHRISTMAS ISLAND	[+7]	Indian/Christmas	Canonical
[CC]	COCOS ISLANDS	[+6:30]	Indian/Cocos	Canonical
[TF]	FRENCH SOUTHERN AND ANTARTIC ISLANDS	[+5]	Indian/Kerguelen	Canonical
[WS]	SAMOA	[+13]	Pacific/Apia	Canonical
[NZ]	NEW ZEALAND	[+12]	Pacific/Auckland	Canonical
[PG]	PAPUA NEW GUINEA	[+11]	Pacific/Bougainville	Canonical
[NZ]	NEW ZEALAND (CHATHAMAN ISLANDS)	[+12:45]	Pacific/Chatham	Canonical
[FM]	FEDERATED STATES OF MICRONESIA	[+10]	Pacific/Chuuk	Canonical
[VU]	VANUATU	[+11]	Pacific/Efate	Canonical
[KI]	KIRIBATI	[+13]	Pacific/Enderbury	Canonical
[TK]	TOKELAU	[+13]	Pacific/Fakaofo	Canonical
[FJ]	FIJI	[+12]	Pacific/Fiji	Canonical
[TV]	TUVALU	[+12]	Pacific/Funafuti	Canonical
[PF]	FRENCH POLINESIA	[-9]	Pacific/Gambier	Canonical
[SB]	SOLOMON ISLANDS	[+11]	Pacific/Guadalcanal	Canonical
[KI]	KIRIBATI	[+14]	Pacific/Kiritimati	Canonical
[FM]	FEDERATED STATES OF MICRONESIA	[+11]	Pacific/Kosrae	Canonical
[MH]	MARSHALL ISLANDS	[+12]	Pacific/Kwajalein	Canonical
[MH]	MARSHALL ISLANDS	[+12]	Pacific/Majuro	Canonical
[PF]	FRENCH POLINESIA	[-9:30]	Pacific/Marquesas	Canonical
[NR]	NAURU	[+12]	Pacific/Nauru	Canonical
[NU]	NIUE	[-11]	Pacific/Niue	Canonical
[NF]	NORFOLK ISLAND	[+11]	Pacific/Norfolk	Canonical
[NC]	NEW CALEDONIA	[+11]	Pacific/Noumea	Canonical

OCEANIA CONTINUATION				
COUNTRY CODE	COUNTRY NAME	OFFSET WITHOUT DST	TIME ZONE	CATEGORY
[PW]	PALAU	[+9]	Pacific/Palau	Canonical
[PN]	PITCAIRN ISLANDS	[-8]	Pacific/Pitcairn	Canonical
[FM]	FEDERATES STATES OF MICRONESIA	[+11]	Pacific/Pohnpei	Canonical
[FM]	FEDERATES STATES OF MICRONESIA	[+11]	Pacific/Ponape	alias of Pacific/Pohnpei
[PG]	PAPUA NEW GUINEA	[+10]	Pacific/Port_Moresby	Canonical
[CK]	COOK ISLANDS	[-10]	Pacific/Rarotonga	Canonical
[PF]	FRENCH POLINESIA	[-10]	Pacific/Tahiti	Canonical
[KI]	KIRIBATI	[+12]	Pacific/Tarawa	Canonical
[TO]	TONGA	[+13]	Pacific/Tongatapu	Canonical
[FM]	FEDERATED STATES OF MICRONESIA	[+10]	Pacific/Truk	alias of Pacific/Chuuk
[WF]	WALLIS AND FUTUNA	[+12]	Pacific/Wallis	Canonical
[FM]	FEDERATED STATES OF MICRONESIA	[+10]	Pacific/Yap	alias of Pacific/Chuuk

SPECIAL AREAS		
OFFSET WITHOUT DST	TIME ZONE	CATEGORY
[-12]	Etc/GMT+12	Canonical
[-11]	Etc/GMT+11	Canonical
[-10]	Etc/GMT+10	Canonical
[-9]	Etc/GMT+9	Canonical
[-8]	Etc/GMT+8	Canonical
[-7]	Etc/GMT+7	Canonical
[-6]	Etc/GMT+6	Canonical
[-5]	Etc/GMT+5	Canonical
[-4]	Etc/GMT+4	Canonical
[-3]	Etc/GMT+3	Canonical
[-2]	Etc/GMT+2	Canonical
[-1]	Etc/GMT-1	Canonical
[+0]	Etc/GMT	Canonical
[+0]	Etc/GMT-0	alias of Etc/GMT
[+0]	Etc/GMT+0	alias of Etc/GMT
[+0]	Etc/GMT0	alias of Etc/GMT
[+0]	Etc/UTC	Canonical
[+0]	GMT	alias of Etc/GMT
[+1]	Etc/GMT+1	Canonical
[+2]	Etc/GMT-2	Canonical
[+3]	Etc/GMT-3	Canonical
[+4]	Etc/GMT-4	Canonical
[+5]	Etc/GMT-5	Canonical
[+6]	Etc/GMT-6	Canonical
[+7]	Etc/GMT-7	Canonical
[+8]	Etc/GMT-8	Canonical
[+9]	Etc/GMT-9	Canonical
[+10]	Etc/GMT-10	Canonical
[+11]	Etc/GMT-11	Canonical
[+12]	Etc/GMT-12	Canonical
[+13]	Etc/GMT-13	Canonical
[+14]	Etc/GMT-14	Canonical

The following are codes which are not being used in the present, we include it because they appear in the database and maybe one can use them for some purpose.

Time Zone deprecated	Corresponding Actual Time Zone
Australia/ACT	to Australia/Sydney
Australia/LHI	Australia/Lord_Howe
Australia/NSW	Australia/Darwin
Australia/North	Australia/Sydney
Australia/Queensland	Australia/Brisbane
Australia/South	Australia/Adelaide
Australia/Tasmania	Australia/Hobart
Australia/Victoria	Australia/Melbourne
Australia/West	Australia/Perth
Brazil/Acre	Brazil/Rio_Branco
Brazil/DeNoronha	Brazil/Noronha
Brazil/East	Brazil/Sao_Paulo
Brazil/West	Brazil/Manaus
Canada/Atlantic	America/Halifax
Canada/Central	America/Winipeg
Canada/Eastern	America/Toronto
Canada/Mountain	America/Edmonton
Canada/Newfoundland	America/St_Johns
Canada/Pacific	America/Vancouver
Canada/Saskatchewan	America/Regina
Canada/Yukon	America/Whitehorse
CET	Europe/Paris
Chile/Continental	America/Santiago
Chile/EasterIsland	Pacific/Easter
CST6CDT	America/Chicago
Cuba	America/Havana
EET	Europe/Sofia
Egypt	Africa/Cairo
Eire	Europe/Dublin
EST	America/Cancun
EST5EDT	America/New_York
Etc/Greenwich	Etc/GMT
Etc/UCT	
Etc/Universal	Etc/UTC
Etc/Zulu	Etc/UTC
GB	Europe/London
GB-Eire	Europe/London
GMT+0	Etc/GMT
GMT-0	Etc/GMT
GMT0	Etc/GMT
Greenwich	Etc/GMT
HST	Pacific/Honolulu
Hongkong	Asia/Hong_kong

Time Zone deprecated	Corresponding Actual Time Zone
Iceland	Atlantic/Reykjavik
Iran	Asia/Teheran
Israel	Asia/Jerusalem
Jamaica	America/Jamaica
Japan	Asia/Tokyo
Kwajalein	Pacific/Kwajalein
Libya	Africa/Tripoli
MET	Europe/Paris
MST	America/Phoenix
MST7MDT	America/Denver
Mexico/BajaNorte	America/Tijuana
Mexico/BajaSur	America/Mazatlan
Mexico/General	America/Mexico_City
NZ	America/Auckland
NZ-CHAT	America/Chatham
Navajo	America/Denver
PRC	Asia/Shanghai
Poland	Europe/Warsaw
Portugal	Europe/Lisbon
PST8PDT	America/Los_Angeles
ROC	Asia/Taipei
ROK	Asia/Seul
Singapore	Asia/Singapore
Turkey	Europe/Istanbul
UCT	Etc/UCT
Universal	Etc/UTC
US/Alaska	America/Anchorage
US/Aleutian	America/Adak
US/Arizona	America/Phoenix
US/Central	America/Chicago
US/East-Indiana	America/Indiana/Indianapolis
US/Eastern	America/New_York
US/Hawaii	America/Honolulu
US/Indiana-Starke	America/Indiana/Knox
US/Michigan	America/Detroit
US/Mountain	America/Denver
US/Pacific	America/Los_Angeles
US/Pacific-New	America/Los_Angeles
US/Samoa	Pacific/Pago_Pago
WET	Europe/Lisbon
W-SU	Europe/Moscow
Zulu	Etc/UTC