Extended Legacy Format (ELF): Date, Age and Time Microformats

30 December 2018

> *Editorial note —* This is a **first public draft** of the microformats used for dates, ages and times in FHISO's proposed suite of Extended Legacy Format (ELF) standards. This document is not endorsed by the FHISO membership, and may be updated, replaced or obsoleted by other documents at any time.
>
> Comments, discussion and other feedback on this draft should be directed to the tsc-public@fhiso.org mailing list.
>
> | | |
> |---|---|
> | Latest public version: | `https://fhiso.org/TR/elf-dates` |
> | This version: | `https://fhiso.org/TR/elf-dates-20181230` |

FHISO's **Extended Legacy Format** (or **ELF**) is a hierarchical serialisation format and genealogical data model that is fully compatible with GEDCOM, but with the addition of a structured extensibility mechanism. It also clarifies some ambiguities that were present in GEDCOM, and documents best current practice.

The **GEDCOM** file format developed by The Church of Jesus Christ of Latter-day Saints is the *de facto* standard for the exchange of genealogical data between applications and data providers. Its most recent version is GEDCOM 5.5.1 which was produced in 1999, but despite many technological advances since then, GEDCOM has remained unchanged.

> *Note —* Strictly, [GEDCOM 5.5] was the last version to be publicly released back in 1996. However a draft dated 2 October 1999 of a proposed [GEDCOM 5.5.1] was made public; it is generally considered to have the status of a standard and has been widely implemented as such.

FHISO are undertaking a program of work to produce a modernised yet backward-compatible reformulation of GEDCOM under the name ELF, the new name having been chosen to avoid confusion with any other updates or extensions to GEDCOM, or any future use of the name by The Church of Jesus Christ of Latter-day Saints. This document is one of three that form the initial suite of ELF standards, known collectively as ELF 1.0.0:

— **ELF: Serialisation Format**. This standard defines a general-purpose serialisation format based on the GEDCOM data format which encodes a *dataset* as a hierarchical series of *lines*, and provides low-level facilities such as escaping and extensibility mechanisms.
— **ELF: Date, Age and Time Microformats**. This standard defines microformats for representing dates, ages and times in arbitrary calendars, together with how they are applied to the Gregorian, Julian, French Republican and Hebrew calendars.

— **ELF: Data Model**. This standard defines a data model based on the lineage-linked GEDCOM form, reformulated in terms of the serialisation model described in this document. It is not a major update to the GEDCOM data model, but rather a basis for future extension.

> *Editorial note —* At the time this draft was published, neither [ELF Data Model] nor [ELF Serialisation] were yet at the stage of having a first public draft available, however FHISO's Technical Standing Committee (TSC) are working on them and hope to have first drafts available soon.

An explanation of the conventions used in this standard can be found in §1, and the general concepts associated with time, calendars and uncertainty are defined in §2. A generic syntax for expressing dates in arbitrary calendars is given in §3.1, which §3.2 extends to support imprecisely known dates and dates written in natural language, and §3.4 extends to support date periods representing extended states of being. Four specific calendars are defined in §4, allowing the Gregorian, Julian, French Republican and Hebrew calendars to be used in the generic date syntax.

This standard defines five datatypes so that the formats defined here can be used in ELF. The `elf:DateValue` datatype defined in §3.3 is ELF's standard datatype for representing historical dates, while the `elf:DatePeriod` datatype defined in §3.4 is used to represent date periods, such as the period of coverage of a source. The `elf:DateExact` datatype defined in §4.1.1 is a more restricted format for expressing exact dates in the Gregorian calendar, and is used to record when ELF objects were created or last modified, typically in conjunction with the `elf:Time` format defined in §5. The `elf:Age` datatype defined in §6 is used to represent individuals' ages. These datatypes contain only modest changes from GEDCOM, but should serve as a basis for future work on calendars.

1 Conventions used

Where this standard gives a specific technical meaning to a word or phrase, that word or phrase is formatted in bold text in its initial definition, and in italics when used elsewhere. The key words **MUST**, **MUST NOT**, **REQUIRED**, **SHALL**, **SHALL NOT**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **NOT RECOMMENDED**, **MAY** and **OPTIONAL** in this standard are to be interpreted as described in [RFC 2119].

An application is **conformant** with this standard if and only if it obeys all the requirements and prohibitions contained in this document, as indicated by use of the words MUST, MUST NOT, REQUIRED, SHALL and SHALL NOT, and the relevant parts of its normative references. Standards referencing this standard MUST NOT loosen any of the requirements and prohibitions made by this standard, nor place additional requirements or prohibitions on the constructs defined herein.

> *Note* — Derived standards are not allowed to add or remove requirements or prohibitions on the facilities defined herein so as to preserve interoperability between applications. Data generated by one *conformant* application must always be acceptable to another *conformant* application, regardless of what additional standards each may conform to.

If a *conformant* application encounters data that does not conform to this standard, it MAY issue a warning or error message, and MAY terminate processing of the document or data fragment.

This standard depends on FHISO's **Basic Concepts for Genealogical Standards** standard. To be *conformant* with this standard, an application MUST also be *conformant* with [Basic Concepts]. Concepts defined in that standard are used here without further definition.

> *Note* — In particular, precise meaning of *character*, *string*, *whitespace*, *whitespace normalisation*, *language tag*, *term*, *prefix notation*, *prefix*, *property*, *datatype* and *subtype* are given in [Basic Concepts].

Indented text in grey or coloured boxes does not form a normative part of this standard, and is labelled as either an example or a note.

> *Editorial note* — Editorial notes, such as this, are used to record outstanding issues, or points where there is not yet consensus; they will be resolved and removed for the final standard. Examples and notes will be retained in the standard.

The grammar given here uses the form of EBNF notation defined in §6 of [XML], except that no significance is attached to the capitalisation of grammar symbols. *Conforming* applications MUST NOT generate data not conforming to the syntax given here, but non-conforming syntax MAY be accepted and processed by a *conforming* application in an implementation-defined manner.

> *Note* — In this form of EBNF, *whitespace* is only permitted where it is explicitly stated in the grammar. It is not automatically permitted between arbitrary tokens in the grammar.

The grammar productions in this standard uses the `S` production defined in §2 of [Basic Concepts] to match any non-empty sequence of *whitespace characters*.

This standard defines five *datatypes* to represent time-related concepts in ELF, each of which is identified by a *term name*, which is simply an IRI. The concept of a *datatype*, as used by FHISO, is defined in §6 of [Basic Concepts], and the definition of each *datatype* in this standard includes a table listing its formal *properties*.

> *Note* — These *properties* include a formal statement that the *datatype* is *datatype*, define the *pattern* and *non-trivial supertypes* of the *datatype*, and say whether it is an *abstract datatype*. These concepts are defined in §6.1, §6.2 and §6.3 of [Basic Concepts]. The *pattern* is a regular expression written in FHISO's `types:Pattern` *datatype* defined in [FHISO Patterns]. This information forms part of an abstraction which allows applications to use a *discovery* mechanism to find out about unknown components, thus allowing them to be processed in

> more sophisticated ways than could be done with a truly unknown component. To support this, FHISO's web server has been configured to provide [Triples Discovery] on all *terms* defined in this standard. Such functionality is outside the scope of this standard, and is entirely OPTIONAL in ELF. Most readers can safely ignore this formalism and the tables of *properties* given for each *datatype*.

This standard uses the *prefix notation*, as defined in §4.3 of [Basic Concepts], when discussing specific *terms*. The following *prefix* bindings are assumed in this standard:

`elf`	`https://terms.fhiso.org/elf/`
`xsd`	`http://www.w3.org/2001/XMLSchema#`
`types`	`https://terms.fhiso.org/types/`

> *Note* — The particular *prefixes* assigned above have no relevance outside this standard document as *prefix notation* is not used in the formal data model as defined by this standard. This notation is simply a notational convenience which makes the standard easier to read.

2 General concepts

> *Editorial note* — It is anticipated that this section will be moved to [Basic Concepts] in a future draft of these documents.

2.1 Time

An **instant** is defined as an infinitesimally brief point in time.

> *Note* — Although defined as an infinitesimally brief point in time, it may be subject to the various forms of uncertainty described in §2.3.

> *Example* — King Alfred's birth occurred at some particular *instant* in the middle of the ninth century. Even though the year is not known with any great certainty, it is still an *instant*.

A **time interval** is defined as the section of time spanning between two specific *instants*.

> *Example* — The interval lasting from midday on 1 Feb 2018 until midday on 14 Feb 2018 is a *time interval*.

> *Example* — The lifetime of a particular individual is another example of a *time interval*, beginning at the *instant* of their birth and ending with the *instant* of their death.

A **duration** is the length of time elapsing between two *instants*, but without reference to any specific choice of start and end *instants*.

> *Example —* "3 days", and "34 years, 2 months" are two examples of *durations* expressed in natural language.

> *Note —* *Durations* differ from *time intervals* in that *time intervals* are *durations* with specific start and end *instants*. A *time inteval* has a *duration* associated with it, quantifying how long it lasts.

> *Note —* ELF does not provide a general-purpose *duration datatype*, but the `elf:Age` *datatype* defined in §6 is a *datatype* customised for representing the *duration* of an individual's life.

Fundamental to ELF's handling of dates is a set of *time intervals* called **calendar days**, each of which spans from one midnight until the next.

> *Note —* A *calendar day* lasts for 24 hours, except when leap seconds is inserted or deleted, or when the local time zone changes, as in the transition to or from daylight saving time. In practical terms, it is a period during which the sun rises and sets exactly once, except in the polar regions.

> *Note —* Because midnight does not occur simultaneously around the world, the set of *calendar days* in one region may be offset compared to those in another region. The details vary depending on local legislation and custom. Currently, there can be three different *calendar days* happening simultaneously in various parts of the world: when a *calendar day* is just beginning in the Line Islands of Kiribati, it is still the previous *calendar day* in most of world's landmasses, and the *calendar day* before that in American Samoa. This means it is possible for a person to participate in an event on one *calendar day*, travel to another region, and subsequently participate in an event on the previous *calendar day*. If the second event is the person's death, this could theoretically result in a living person participating in an event the *calendar day* after their death.

> *Editorial note —* Does this definition need loosening? Not all cultures consider the day to be begin at midnight. The Hebrew calendar defined in §4.4, for example, is normally used with *calendar days* beginning at sunset, which is defined as 6pm using variable length hours. The Islamic and Bahá'í *calendars* do similarly. The definition of a *calendar day* is currently taken from [ISO 8691], but should it be loosened to allow such definitions?

A **date** is a way of identifying a particular *calendar day*.

> *Note —* ELF *dates* do not include an indication of either the time zone or the locale which leaves some ambiguity into the exact points in time that are meant. The [ISO 8601] concept of a *date* has the same ambiguity.

> *Editorial note —* This standard deliberately does not specify whether the *calendar day* 1 January 2018 in Vancouver is the same *calendar day* as 1 January 2018 in Paris, which started nine hours earlier than the *calendar day* in Vancouver. This is left unspecified because [ISO 8601] is similarly vague. However if they are regarded as distinct *calendar days*, they are represented by the same *date* in ELF.

> *Note —* The definitions of an *instant*, a *time interval*, a *duration*, a *calendar day* and a *date* given here are intended to be fully compatible with the definitions of these concepts in [ISO 8601]. Any incompatibility between the definitions here and those in [ISO 8601] is unintentional.

> *Editorial note —* These concepts have been defined here rather than by normative reference to [ISO 8601] because of the cost involved in obtaining a legal copy of [ISO 8601], and the likelihood that implementers will not do so.

A **time of day** is a way of identifying an instant within a *calendar day*, done by dividing an ordinary *calendar day* into 24 **hours**, each of which is subdivded into 60 **minutes**, each of which is further divided into 60 **seconds**. In ELF, a *time of day* is represented by the `elf:Time` *datatype* defined in §5.

> *Note —* A *calendar days* may exceptionally be divided differently if a leap second is inserted or deleted, or when the local time zone changes.

2.2 Calendars

Many different systems for reckoning *dates* have been used throughout history and in different parts of the world. Such systems are called **calendars**, and ELF allows historical *dates* to be specified using many different *calendars*.

> *Example —* The simplest form of *calendar* is to count the number of *calendar days* which have elapsed since a particular day zero. The most popular such *calendar* is called the Julian Day (which is unconnected to the similar-sounding Julian Calendar). Its day zero is 24 November 4714 BC in the proleptic Gregorian Calendar, a day chosen to be before all recorded history. Written as a Julian Day, 1 January 2000 can be represented by the integer 2451545. Such *calendars* are not commonly used for writing historical *dates* as they are cumbersome and error-prone.

> *Editorial note —* Nevertheless, FHISO might consider standardising the Julian day as a lightweight *calendar* for use as a common intermediate *calendar* during the conversion of *dates* from one *calendar* to another.

Many *calendars* make use of units of time which are longer than a *calendar day*, and the general framework for dates in ELF allows for two such units of time, a **calendar month** and a **calendar year**, whose definitions will be dependent on the particular *calendar*.

> *Note* — It is intended that a *calendar year* will typically be unit of time roughly equal to the time it takes the Earth to orbit the Sun, and a *calendar month* will be a unit of time intermediate in duration between a *calendar day* and a *calendar year*, and often loosely based on the time it takes the Moon to orbit the Earth. However these are not requirements, nor is it a requirement that all *calendar years* or all *calendar months* be of approximately equal length in a given *calendar*.

> *Editorial note* — The flexibility to define *calendar years* and *calendar months* arbitrarily might be exploited in the future. FHISO are considering whether there is merit to defining a calendar for the Julian day. If defined it would not be for general use expressing historical *dates*, but rather to provide a way of expressing *epochs* in a *calendar*-neutral way when defining *calendars*. Because there is no applicable notion of a *calendar month* or *calendar year* with Julian days, and because the generic *date* syntax defined in §3.1 is most natural with *calendars* that have a *calendar year*, it is quite likely this *calendar* might define a *calendar year* and a *calendar month* to be identical to a *calendar day*. This standard does not prohibit this.

An **incomplete date** is a way of identifying a particular *calendar month* or *calendar year* without identifying a specific *calendar day*.

> *Example* — In the Gregorian *calendar*, "June 1953" is an *incomplete date* as it identifies a particular *calendar month*, but not a specific *calendar day* within that month.

> *Note* — Under this definition, an *incomplete date* is a *date* when it is being used to identify a particular *calendar date*, but with limited *precision*.

An **epoch** is an *instant* which serves as a reference point for a given *calendar* from which *calendar years* are numbered consecutively with an integer called the **logical year**, which either increases or decreases with time. When the *logical year* number increases with time, the *epoch* SHALL be first *instant* of the *calendar year* with the *logical year* number 1. When thee *logical year* number decreases with time, the *epoch* SHALL be the last *instant* of the *calendar year* with *logical year* number 1.

> *Example* — The *epoch* used in many forms of the Islamic Calendar is an *instant* during the Gregorian year AD 622, the year of the Hijrah when Muhammad moved from Mecca to Medina. The first *calendar year* of the Islamic *calendar*, called Anno Hegiræ 1 began at this *epoch* and has the *logical year* number 1. Subsequent Islamic *calendar years* have been numbered sequentially, AH 2, AH 3, etc., and have *logical year* numbers 2, 3, etc. The *calendar year* immediately before the *epoch* is commonly labelled 1 BH (standing for Before the Hijrah), and earlier *calendar years* are numbered backwards from the *epoch*. These have *logical year* numbers 1, 2, 3, etc. too.

> *Note* — This definition does not limit a *calendar* to having a single *epoch*.

A *logical year* number is not sufficient to identify a specific *calendar year*: it is also necessary to state the particular *epoch* from which *calendar years* are numbered, and whether *logical year* numbers increase or decrease with time. An **epoch name** is an identifier which represents these two things. An *epoch name* where *logical year* numbers increase with time is called a **forwards epoch name**, while an *epoch name* where *logical year* numbers decrease with time is called a **reverse epoch name**.

> *Example* — The Gregorian calendar has two distinct *epoch names*, "AD", standing for Anno Domini, which is a *forwards epoch name*, and "BC", standing for Before Christ, which is a *reverse epoch name*. Both use the same *epoch*, defined as midnight at start of 1 January AD 1.

The way years were counted in the past does not always fully coincide with the modern reckoning of years in that calendar, which is what defines the *logical year*, even though they are nominally reckoned from the same *epoch*. The year number according to the historical reckoning is called the **historical year**.

> *Example* — Prior to the adoption of the Gregorian *calendar* in many parts of the world, the year was considered to begin on the Feast of the Assumption (otherwise called Lady Day) which falls on 25 March. This was the case in England before 1752. Sources contemporary to the event record the execution Charles I as happening on 30 January 1648. This was in the month following December 1648 and would now be considered to be in 1649, as a result of which modern accounts usually describe it as happening on 30 January 1649. (This is not a result a change of the Julian *calendar* to the Gregorian one: in the Gregorian *calendar* this *date* is 9 February 1649.) In this example, 1649 is the *logical year*, while 1648 is the *historical year*.

> *Example* — The Byzantine *calendar* counted years since the supposed beginning of the world with an *epoch* on 1 September 5509 BC, however in early times some alternative *dates* were assigned to this *epoch*, including 25 March 5493 BC, sometimes known as the "Alexandrian" *epoch*. A definition of this *calendar* which used the traditional Byzantine *epoch* in 5509 BC would have *logical years* counted from then, but might also allow *historical years* to be counted from the Alexandrian *epoch*. This is an example where the *logical year* and *historical year* differ by more than one.

A *calendar* defines how the number of *calendar days* in each *calendar month* and the number of *calendar months* in each *calendar year* are determined. Stylistic and linguistic variations in the presentation of a *date* do not constitute separate *calendars*.

> *Example* — "31st August, 2018", "31 авг. 2018", "8/31/2018" and "2018年8月31日" are various ways in which the *date* which is represented in [ISO 8601] as "2018-08-31" might be presented. The differences between these presentations are merely stylistic or linguistic ones, and therefore these difference are not separate *calendars*: they are all written using the Gregorian *calendar*.

> *Editorial note —* In due course, FHISO will need to clarify and perhaps revise this definition of what constitutes a distinct *calendar*. Does Roman day reckoning (e.g. "Prid. Kal. Sept" for 31 Aug) count as a separate *calendar* or is just a stylistic variation? What about regnal years (e.g. "31 Aug 67 Eliz II")? These are not strictly separate *calendars*, but it could be convenient to consider them as such in ELF if it is considered desirable for ELF to preserve the fact that the dates were recorded in these forms.

2.3 Uncertainty

The **precision** of a stated value, such as a *date*, is a measure of how specificity with which the value has been specified: the more specifically, the greater the *precision*. Values with relatively high or low *precision* may be described as relatively **precise** or **imprecise**, respectively.

> *Example —* It is more *precise* to say that the Battle of Agincourt was on St Crispin's Day, 1415, than it is to say that the battle occured during Henry V's reign. Both statements are true, but the former has greater *precision* because it identifies the specific day of the battle, while the latter identifies it only as falling within that nine-year reign.
>
> Saying the battle was in the autumn of 1415 has an intermediate level of *precision*. This year might described as *precise* in comparision to the whole of Henry V's reign, or as *imprecise* when compared to the specific day.

> *Note —* Values are stated *imprecisely* for many reasons, including when a more *precise* value is not known and when greater *precision* is considered irrelevant. Another reason is when the value being stated is *inherently ambiguous*.

When the value being stated is the *instant* at which some entity changed state, it is common for this *instant* not to be defined with arbitrary *precision* because there is a *time interval* during the transition when it is not well-defined what the state is. The *duration* of this *time interval* is known as the **inherent ambiguity** of the *instant* of the change.

> *Example —* Depending on the jurisdiction, the *precise instant* during a wedding when the couple become married may be ill-defined as there are several obvious possibilities. It could be argued to occur when the couple complete their vows, or when the priest declares the couple husband and wife; or it might be when the last signature is put on marriage certificate has been completed, or when the ceremony ends. If there is no single accepted definition, then there is likely several minutes of *inherent ambiguity* between the first and the last possibilities. This would be true even if the wedding had been videoed and carefully timed, as it is not due to lack of information on what happened and when.

A stated value is either **exactly stated** or **approximately stated**. An *exactly stated* value is one where it is well-defined exactly what values are considered to be consistent with the stated value.

> *Example* — The *date* of the Battle of Agincourt was stated in three different ways in the earlier example. In order of decreasing *precision* these were "St Crispin's Day, 1415", "the autumn of 1415", and "during Henry V's reign". The meaning of St Cripin's Day is well-defined: it is 25 October. Had the Battle of Agincourt in fact occurred on 24 October 1415, this would not be consistent with the statement that it happened on St Crispin's Day. "St Crispin's Day, 1415" is therefore an *exactly stated* value. "During Henry V's reign" is similarly *exactly stated*.
>
> "The autumn of 1415" is very likely not well-defined. Some people define it as stretching from the autumnal equinox in late September to the winter solstice in late December, but this definition is by no means universal. Often it is used a more vague manner to refer to the later part of the year in the northern hemisphere. Unless context makes it clear that a specific, well-defined meaning of word "autumn" was intended, this is not an *exactly stated* value: it is therefore an *approximately stated* value.
>
> The battle might also be described as happening in about 1415. This statement is true as the battle did in fact occur in 1415, but it is an *approximately stated* value. Had the battle actually been in 1414, would this be consistent with the description "about 1415"? Probably. But what about 1411? Or 1401? There is no general answer, and as a result "about 1415" is an *approximately stated* value.

> *Example* — These concepts do not only apply to quantitative values. It is more *precise* to say that a person was born in the commune of Coutances, than to say say that person was born in metropolitan France. "The commune of Coutances" and "metropolitan France" are both *exactly stated* values. The person might also be described as born in northern France, which would normally be interpreted as an *approximately stated* value.

The **precision range** of an exactly stated value is defined as the set of values which would be considered consistent with the stated value. One specific measure of *precision* is the **precision range width**, which is defined as the difference between the two most widely separated values in the *precision range*. When the value being specified is an *instant*, its *precision range* is a *time interval*, and its *precision range width* is a *duration*.

> *Example* — If a person is said to have died in 1967, this is consistent with the *instant* of their death being at any time between midnight at the start of 1 January 1967 and midnight at the end of 31 December 1967. The *time interval* between these two *instants* is the *precision range* and is a *calendar year*. In this example, the stated value is 1967 and its *precison range* is the *duration* 1 year.

> *Example —* If a person is said to have married in the 1910s, it is fairly clear this refers to a decade and therefore the *precision range* is 10 years. However if the person is said to have married in the 1900s, this might mean the decade or the century. Without further context, the intended *precision range* is unclear.

The **accuracy** of a value is a measure of how close a stated value is to the true value. A *exactly stated* value is said to **accurate** if the true value lies within the *precision range* of the stated value, and **inaccurate** otherwise. For an *approximately stated* value, the *accuracy* is relative: the further the stated value is from the true value, the less *accurate* or more *inaccurate* the stated value.

> *Note —* The *precision* of a value is unrelated to its *accuracy*. A value may be *precise* or *imprecise* independently of whether it is *accurate* or *inaccurate*.

> *Example —* The following table gives example *instants* of birth for Queen Victoria which are variously *precise* or *imprecise*, and *accurate* or *inaccurate*.
>
> | | | |
> |---|---|---|
> | Precise | Accurate | 24 May 1819 at 4am |
> | Precise | Inaccurate | 19 Jun 1833 at 9pm |
> | Imprecise | Accurate | During the 1810s |
> | Imprecise | Inaccurate | During the 1790s |
>
> It is generally accepted that Queen Victoria was in fact born at 4.15am on 24 May 1819.

> *Note —* In principle, the *accuracy* of any stated value is unknowable, though in practice some facts are so well established they can be regarded as proven for all practical purposes as the alternative would require there to have been a vast conspiracy. Much of the time the situation is not so clear.

The likelihood that a stated value is *accurate* is referred to as its **reliability**.

> *Note —* Although this is in theory a probability, *reliabilities* are usually described comparatively or quantitatively. A researcher may gauge the *reliability* of a stated value by considering the reliability of the sources in which it is stated, and the corroborating or contradicting evidence. Different researchers might reasonably reach different conclusions on the *reliability* of a stated value.

> *Note —* A stated value can be considered *unreliable* by virtue of being stated with excessive *precision*.

> *Example —* Suppose a man was last seen on 1 January and his corpse found on 31 January. The coroner determined the man had been dead for one to two weeks when found, but that no more *precise* date of death could be established. A newspaper obituary simply said he died in January, but a gravestone was erected giving his date of death as 21 January.

A researcher might conclude the bare month given in the obituary is *reliable* because the relatively *imprecise date* is very likely *accurate* as it is consistent with the other evidence. The gravestone might be *accurate*, and it is not directly contradicted by the other evidence, but if the researcher believes the *date* "21 January" was made up, perhaps so that something could be put on the grave, it might be judged *unreliable* as there is a high likelihood that the true *instant* of death was not actually on 21 January.

This is not an example of *inherent ambiguity*. Depending on the circumstances of the death, there may have been a few minutes of *inherent ambiguity* as the man's life slowly ebbed away, but the bulk of the uncertainty is from lack of knowledge of what happened and when.

2.4 Date concepts in other formats

Editorial note — This whole section may vanish in a future draft.

The *datatypes* defined in this standard are NOT RECOMMENDED for us in serialisation formats other than with ELF.

Editorial note — In due course we need to decide FHISO's preferred way of handling *dates* and *durations* in other serialisation formats. GEDCOM X, for example, uses a format more closely aligned with [ISO 8601], and in early discussion on *dates* and in our call for paper submissions, we were erring in that direction too.

If we have one *date* format for ELF, another for GEDCOM X, and possibly even a third one for a future format of our own, we will probably want to make sure we don't end up with *dates* formatted for GEDCOM X appearing in ELF, or vice versa, otherwise an ELF application will need to know about every *date* format rather than just the ELF *date* formats. At some level, this requires converting *dates* between formats when data is transferred between systems. For data in the [ELF Data Model] or in the GEDCOM X data model, this is no problem as a data conversion stage will be required when converting between data models, and it can convert the *date* formats too.

A problem arises in FHISO's component standards, such as [CEV Concepts], which are intended to be usable in ELF, GEDCOM X and other data models. An ELF application will not necessarily know about CEV and will then see the CEV *structures* as unknown extensions, so there needs to some way of indicating that the ELF structure it is reading contains a *date*. This could be done by requiring an ELF schema to be present and have it specify the datatype for the payload, though that might be too onerous a requirement. Another option is to require a specific tag like `DATE` to be used, and special case this. The same issue may exist for *ages* too, but it is not general to all *datatypes* — just those which have to be formatted differently in different serialisations, which will hopefully be a minority of *datatypes*.

3 Date formats

ELF uses three different *datatypes* to represent *dates*, depending on the context.

— `elf:DateValue` is used for historical *dates*, and is defined in §3.3.
— `elf:DatePeriod` is used to record the period of coverage a source, and is defined in §3.4.
— `elf:DateExact` is used to record the creation or modification *date* of various objects in the data model, and is defined in §4.1.1.

As the first two of these *datatypes* are used to record historical *dates*, and ELF allows historical *dates* to be expressed using many different *calendars*, these two *datatypes* each allow *dates* in arbitrary *calendars*. This is achieved by providing a generic *date* syntax which all *dates* MUST match, regardless of *calendar*, and which begins with a *calendar escape* indicating the specific *calendar* in use. This generic syntax is defined in §3.1, and extensions to it to support imprecisely known dates and dates written in natural language are given in §3.2.

> *Editorial note* — An earlier draft of this standard used a separate *datatype* for each *calendar*, and [ELF Serialisation] used `@#D...@` as a way of tagging the *datatype*. This approach was eventually abandoned because it could not cope with *date ranges* and *date periods* where the two end points used different *calendars*. [GEDCOM 5.5.1] permits this, and although many applications do not support it, the TSC considered the following use case to be important enough that ELF needed to support it too:
>
> ```
> 0 INDI
> 1 NAME George II
> 1 TITL King of Great Britain
> 2 DATE FROM @#DJULIAN@ 11 JUN 1727 TO @#DGREGORIAN@ 25 OCT 1760
> ```
>
> Were there *datatype* per *calendar*, what should the *datatype* of the above `DATE` element's payload be? It's neither wholly Julian nor wholly Gregorian. Because of the difficulties with such constructs, we dropped the idea of making each *calendar* a separate *datatype*.
>
> One option for solving this which the TSC seriously considered is to introduce **compound calendars**, along the lines of the proposal in CFPS 38, but this adds complexity due to the need to add a mechanism for defining *compound calendars*.
>
> A separate complication comes from the fact that `elf:DateValue` and `elf:DatePeriod` are separate *datatypes*, and *calendar*-specific *subtypes* of each would likely be required. This could be solved by removing periods from the data model entirely, perhaps by having the serialisation layer split up `DATE` tags containing a period. This may make sense at a date model level too, if we model periods as two implicit events: one intiating and one concluding the period being discussed. There is less of a case for doing the same wtih ranges, so a solution to the problem of ranges with multiple calendars would still be required.
>
> The TSC believe an approach along these lines could be made to work, but it would be too big a change to include in ELF 1.0. We also feel we should investigate alternative approaches

> before committing to a specific solution. Deferring this functionality until a future version of ELF will give us time to do give suitable consideration to the options available.

3.1 Generic date syntax

A *date* is represented in the generic *date* syntax as a sequence of five components: a *calendar escape*, followed by encodings of the *calendar day*, *calendar month*, *calendar year* and *epoch name*. Only the *calendar year* is REQUIRED; the other components are OPTIONAL, except that the *calendar day* cannot be present if the *calendar month* has been omitted. It matches the `Date` production.

```
Date   ::= (CalEsc S)? ((Day S)? Month S)? Year (S? Epoch)?

CalEsc ::= "@#D" [A-Z] [A-Z ]* "@"
Day    ::= [0-9]+
Month  ::= [A-Z] [A-Z0-9] [A-Z0-9]+
Year   ::= "-"? [0-9]+ ( "/" "-"? [0-9]+ )?
Epoch  ::= [A-Z] ( [A-Z] | [A-Z0-9._]* [._] [A-Z0-9._]* )
           | "$" [^ #x9#xA#xD]+
```

> *Example —* The following are examples of *dates* which match the `Date` production:
>
> ```
> 63 B.C.
> 21 JAN 1793
> @#DJULIAN@ 29 MAY 1453
> ```
>
> The first of these includes only a *calendar year* and *epoch name*. The following two both have a *calendar day*, *calendar month* and *calendar year*, and neither specifies an *epoch name*. Only the third *date* includes a *calendar escape*.

> *Note —* The `Month` and `Epoch` productions are more complex than might at first seem necessary to ensure that no *string* can match both productions. This ensures a *string* containing just a *day* and a *month*, such as "`1 JAN`", cannot match the `Date` production. A future version of ELF might allow the year to be omitted in *dates* where it is unknown.

A *conformant* application serialising a *date* using this syntax SHOULD use a single space *character* (U+0020) wherever *whitespace* is permitted in the `Date` production.

> *Note —* [GEDCOM 5.5.1] under-specifies how *whitespace* is allowed in *dates*. The `Date` production is somewhat permissive in its treatment of *whitespace*, though does not allow it to be omitted entirely. The preceding recommendation ensures that *conformant* ELF applications will be maximally compliant with current GEDCOM application which typically use a single space *character*.

3.1.1 Calendar escapes

The `CaleEsc` production encodes the **calendar escape**, which identifies the particular *calendar* being used in the *date*.

> *Note* — Syntactically, the *calendar escape* is an ELF escape, as defined §XX of [ELF Serialisation]. When such escapes occur in the payload of a `DATE` line, they are passed through to the data model unaltered.

> *Editorial note* — Check the above is accurate once [ELF Serialisation] has been updated, and update the reference. Note that this means historical dates MUST appear on `DATE` lines.

The following *calendar escapes* are defined in §4 of this standard.

`@#DGREGORIAN@`	The Gregorian *calendar* defined in §4.1
`@#DJULIAN@`	The Julian *calendar* defined in §4.2
`@#DFRENCH R@`	The French Republican *calendar* defined in §4.3
`@#DHEBREW@`	The Hebrew *calendar* defined in §4.4

> *Note* — The *calendar escape* for the French Republican *calendar* contains exactly one space *character* (U+0020). It MUST NOT be written with any alternative form of *whitespace*.

> *Editorial note* — An earlier draft of this standard included `@#FRENCHR@` (without a space) as an alias for `@#DFRENCH R@`, but we made no record of the use case that lead to its inclusion and have removed it again.

> *Note* — [GEDCOM 5.5.1] includes one further *calendar escape*, `@#ROMAN@`, which it reserved for future use, presumably for use with Roman Republican *calendar*. This standard does not reserve this *calendar escape*.

The `@#DUNKNOWN@` *calendar escape* is permanently reserved. Third parties MUST NOT define calendars with this name and applications SHOULD NOT generate *dates* using this *calendar escape*, however *conformant* applications MUST NOT discard *dates* using this *calendar escape*. *Conformant* applications MUST NOT assume that two dates with a `@#DUNKNOWN@` *calendar escape*, whether explicitly written or inferred as described below, are expressed in the same *calendar*.

> *Example* — The generic *date* syntax puts sufficient constraints on how *calendars* can be represented that applications MAY make certain assumption about *dates* written in unknown *calendars*, for example that `@#DISLAMIC@ 1 RAJ 1420` is an earlier date than `@#DISLAMIC@ 9 RAM 1422` if both are *well-formed dates*. This is because year numbers MUST increase with time. *Conformant* applications MUST NOT make similar assumptions for two *dates* with the `@#DUNKNOWN@` *calendar escape* as they might not be represented using the same *calendar*.

> *Note* — It is the intention of FHISO to allow third parties to define their own *calendar escapes* in order to support additional *calendars*. However this version of ELF provides no means of avoiding conflict between separate third-party *calendar escapes*. This is particularly problematic when there a several variants of a calendar, and if different vendors choose to implement a different variant using the same *calendar escape*. FHISO intend to introduce a mechanism to avoid such conflicts in a future version of ELF. This is likely to work by assigning a *term* to each *calendar*, and a syntax for binding *calendar escapes* to *term name* IRIs.

> *Editorial note* — This functionality was dropped from ELF 1.0 because the specification proved more complicated than expected. The intention is to add *calendar* bindings to the ELF schema, such as this:
>
> ```
> 1 SCHMA
> 2 PRFX elf https://terms.fhiso.org/elf/
> 2 IRI elf:JulianCalendar
> 3 DTYPE JULIAN
> ```
>
> Because *dates* are likely to copied around, once in the data model the *date* needs to reference the *calendar term name* rather than the *calendar escape*. This means the serialisation layer needs to convert *calendar escapes* to *term names*, and vice versa. Finding a clean way of doing this proved problematic.
>
> The TSC had been considering making *calendars* a specific sort of *datatype*, and then have the serialisation layer treat the *calendar escape* as way of tagging the *datatype* of the `DATE` tag's payload. However, as noted in an earlier editorial note, this caused problems with *date periods* and *date ranges* which used multiple *calendars*, and has been deferred to a future version of ELF.

This generic *date* syntax defines only some basic syntactic constraints on the representation of the *calendar day*, *calendar month*, *calendar year* and *epoch name* components. The party defining each *calendar* SHOULD define further constraints on these components to define what constitutes a **well-formed** *date* in that *calendar*. Where possible, the set of *well-formed dates* SHOULD be the same as the set of *dates* that actually existed.

> *Example* — The *date* "`12 AUGUST 2000`" is not a *well-formed date* in the Gregorian calendar, as defined in §4.1, because the specified month, "`AUGUST`", is not one of the twelve allowed months names: it ought to have been written "`12 AUG 2000`".

> *Example* — The *date* "`29 FEB 1973`" is not a *well-formed date* in the Gregorian calendar, because February 1973 only had 28 days.

> *Note* — This standard does not prohibit *calendars* from defining *dates* which never occurred to be *well-formed*, though this is generally *not recommended*. It is allowed to accommodate *calendars* where the exact sequence of *dates* is either unknown or cannot be determined algorithmically.

> *Example* — Certain versions of the Islamic *calendar* define the start of each *calendar month* by when the new moon is actually observed. This results in unpredictable month lengths. If such a *calendar* were defined for use in ELF, it would likely regard days beyond the expected end of the month as *well-formed dates* to accommodate the possibility that bad weather had prevented the new moon from being observed.

All *dates* written using a *calendar escape* with which the application is not familiar are assumed to be *well-formed*. This includes all *dates* using the `@#DUNKNOWN@` *calendar escape*.

Conformant applications MUST NOT generate *dates* which are known not to be *well-formed*. Applications encountering *dates* which are known not to be *well-formed* MAY delete the *date* or signal an error to the user. If an application cannot determine whether or not a *date* is *well-formed*, it MUST assume it is *well-formed*.

> *Note* — Other than for *dates* written in the Gregorian *calendar*, no part of this standard requires an application to be able to determine whether a *date* is *well-formed*. In some *calendars*, such as the Hebrew *calendar* defined in §4.4, the rules for determining this are fairly complex. Applications are encouraged to implement these rules in full, but are not required to.

The *calendar escape* is OPTIONAL in the generic *date* syntax. If the *calendar escape* is omitted and if the *date* if *well-formed* in the Gregorian *calendar*, then the *date* is treated as if it used the `@#DGREGORIAN@` *calendar escape*. If the *calendar escape* is otherwise omitted, the *date* is treated as if it used the `@#DUNKNOWN@` *calendar escape*.

> *Note* — [GEDCOM 5.5.1] simply says that *dates* written without a *calendar escape* default to the Gregorian calendar. ELF's approach is more nuanced. If a *date* explicitly uses the Gregorian *calendar escape* then an invalid *date* MAY be deleted; if it is written without a *calendar escape* then it MUST NOT be, assuming it conforms to the generic *date* syntax. ELF deviates from [GEDCOM 5.5.1] in this regard because many current applications fail to include a *calendar escape* when the Julian *calendar* is used. As a result, *dates* like "`29 FEB 1700`" can be found written without a *calendar escape*. This date did not exist in the Gregorian calendar and is not a *well-formed date* in that *calendar*, however this rule prevents applications from deleting it as invalid.

It is RECOMMENDED that, where possible, dates should be entered in ELF datasets using the calendar in which they were written in the source.

> *Example* — A contemporary record of an event occurring in seventeenth century Massachusetts would almost certainly be recoded using the Julian calendar, as Massachusetts, like all the British colonies, did not adopt the Gregorian calendar until 1752. The *date* of this event SHOULD therefore be recorded in ELF using the Julian calendar and not converted to the Gregorian calendar.

3.1.2 Days

The `Day` production encodes the *calendar day* component of the *date*. It is an positive integer which *calendars* SHOULD use to count how many *calendar days* into the *calendar month* the specified *calendar day* is, with the first *calendar day* being "1". Leading zeros preceding a non-zero digit are permitted; *conformant* applications MUST attach no significance to them and MAY remove them.

> *Editorial note* — An earlier draft of this standard allowed non-integer *calendar day* components, providing the first *character* was a decimal digit. The motivation for allowing this came from consideration of Roman day reckoning. In this system, days are reckoned backwards from three fixed points in each month, called the kalends, nones and ides. For example, the day described as "ante diem quintum kalendas Septembres" or "a.d. V Kal. Sept", meaning five days before the kalends of September (1 Sept), counting inclusively, is 28 August. If this were considered a separate *calendar* and non-integer *calendar day* components allowed, the day could be represented as "`5K SEP`". However on balance it was felt this was an unnecessary complication and the facility was removed. The framework is still flexible enough to handle Roman day reckoning by using separate *calendar month* components for the kalends, nones and ides, thus giving 28 August a representation similar to "`5 KSEP`".

> *Editorial note* — Should there be a requirement that days numbers increase monotonically? Roman day reckoning, which counts days backwards, can still be supported if negative day numbers are allowed: e.g. "`-5 KSEP`". This has the advantage of allowing applications to sort *calendar days* without knowing the calendar, and all that would be required to sort days completely would be to know the order of *month names* and *epoch names*.

3.1.3 Months

The `Month` production encodes the *calendar month* component of the *date*. The set of permitted *calendar month* components in a given *calendar* is called the set of **month names** for the *calendar*. *Month names* SHOULD normally be abbreviated forms of their common names, and MUST be at least three *characters* long.

> *Example* — The French Republican *calendar* has twelve months named Vendémiaire, Brumaire, Frimaire, Nivôse, Pluviôse, Ventôse, Germinal, Floréal, Prairial, Messidor, Thermidor and Fructidor. If ELF, as described in §4.3, these are abbreviated `VEND`, `BRUM`, `FRIM`,

> `NIVO`, `PLUV`, `VENT`, `GERM`, `FLOR`, `PRAI`, `MESS`, `THER` and `FRUC`. In addition, each year had five or six consecutive intercalary days, or jours complémentaires, which were not part of any month. For the purposes of representing this calendar in ELF, the intercalary days are considered a thirteenth month, `COMP`.

> *Note* — *Month names* are always upper-case, a constraint guaranteed by the `Month` production. Lower-case, mixed-case, or non-ASCII *month names* MUST NOT be used.

> *Note* — *Month names* are REQUIRED to be at least three *characters* long to avoid conflicting with *epoch names*.

The following words are reserved and MUST NOT be used as *month names* in any calendar: `ABT`, `AFT`, `AND`, `BEF`, `BET`, `CAL`, `EST`, `EVERY`, `FOR`, `FROM`, `INT`, `POS`, `REP`, `TIME`, `UNCERT`, `UNK` and `ZONE`.

> *Note* — Many of these words have specific meanings in the ELF *date datatypes*. The words `EVERY`, `FOR`, `POS`, `REP`, `TIME`, `UNCERT`, `UNK` and `ZONE` are reserved for possible future use because they describe concepts in [GEDCOM X Dates] or [ISO 8601-2] which are not currently in ELF. This does not necessarily mean FHISO will add such functionality to a future version of ELF.

3.1.4 Years

The `Year` production encodes the *calendar year* component of the *date*. The *string* matching this production SHALL be an integer, and MAY be followed by a solidus (U+002F) and another integer. Either integer MAY begin with a minus sign (U+002D).

> *Note* — Negative years and the year zero are supported by this generic *date* syntax, but not by any *calendar* defined in this standard. The intention is that they provide a way of representing a *dates* before the *epoch* of a particular *calendar* where no *reverse epoch name* has been defined. This is not expected to be common with real *calendars*.

> *Editorial note* — One specific example where it might occur is if a future version of ELF defines a *calendar* to represent Julian days, for use in defining *epochs*. The *epoch* for the Julian day is in the Gregorian year 4714 BC, but some other *calendars*, such as the Byzantine calendar, have earlier *epochs* meaning their *epoch* occurs on a negative Julian day.

When the *calendar year* component contains no solidus and second integer value, the integer given is the *logical year* number. A *conformant* application MUST attach no significance to its particular lexical form, and MAY add or remove leading zeros preceding a non-zero digit.

A *calendar year* component with two integers is called a **dual year**, and is used when the historical and modern reckoning of years differ. The first integer in a *dual year* is the *historical year*; the second integer is the *logical year* which MAY be given in abbreviated form.

> *Example* — The principal use of *dual years* is to encode *dates* which were recorded using years beginning on 25 March, as was the practice in many parts of the world in mediæval and early modern times. Charles I's execution occurred on 30 January in the *logical year* 1649, but the *historical year* 1648. To avoid ambiguity, the year can be written 1648/1649. ELF supports this and allows the *date* to be recorded as "`30 JAN 1648/1649`". In this case the *logical year* is not abbreviated.

The *dual year* syntax is only used when there are genuine differences in the conventional reckoning of years. *Dual years* SHOULD be used during periods when use of a *historical year* differing from the *logical year* was common, even if the source in question did not use the *historical year*. This is to remove any potential ambiguity.

> *Example* — The *date* of Charles I's execution SHOULD NOT be written "`30 JAN 1649`", even though this is technically correct as 1649 is the *logical year*, as reckoned with 1 January as the start of the new year. This is true even if the *date* was quoted from a modern book which wrote it using the *logical year*. The form "`30 JAN 1648/1649`" avoids any possible ambiguity over which new year was being used.

Dual years MUST NOT be used simply to record an error in the year as stated in a source.

> *Example* — If a parish register includes a baptism entry dated 12 Jan 1842, but the context and other circumstantial evidence makes it clear that the year was incorrectly written in the register and was in fact 1845, this MUST NOT be recorded in ELF as "`12 JAN 1842/1845`".

When serialising a *date*, a *dual year* MUST be used if the *historical year* and *logical year* differ, and MUST NOT be used if they are equal.

When serialising a *dual year*, applications MAY abbreviate the *logical year* to just the last two digits if the difference between the *historical year* and the *logical year* is less than 10 years, or MAY abbreviate it to just the last one digit if the difference between the *historical year* and the *logical year* is no more than 1 year. If the *historical year* and the *logical year* differ by 10 or more years, abbreviated form MUST NOT be used. When the *logical year* is abbreviated, any minus sign is dropped in the abbreviated form. Where possible, *logical years* SHOULD be abbreviated to two digits.

> *Example* — The *date* of Charles I's execution MAY be written in abbreviated form as "`30 JAN 1648/49`" or as "`30 JAN 1648/9`". The former is RECOMMENDED as this is compatible with the [GEDCOM 5.5.1] standard, though both forms can be found in current GEDCOM files, as can the unabbreviated form.

> *Example —* Abbreviated form MUST NOT be used if *dual years* are being used in the Byzantine *calendar* to express years using standard Byzantine *epoch* and the Alexandrian *epoch*. This is because the *historical year* and *logical year* will differ by 16 or 17 years, depending on the specific *date*.

A specific *calendar* SHOULD normally place further restrictions on how *dual years* can be used, and any *dates* using *dual years* which fail to conform to any such *calendar*-specific restrictions are not *well-formed dates*.

> *Note* — This is necessary if applications are to have understanding of what the *historical date* means. A *date* of "`@#DJULIAN@ 1740/1620`" is perfectly well-defined insofar that it represents the Julian year A.D. 1620 and which for some reason was conventionally recorded as 1740 at the time, but without an understanding of why this was done, it is difficult for an application to know what to do with this information. In fact, the Julian calendar defined in §4.2 only allows *dual years* when the *historical date* and *logical date* differ by exactly one year. "`@#DJULIAN@ 1740/1620`" is not therefore a *well-formed date*.

Calendars MUST NOT place further restrictions on when the abbreviated format can be used.

> *Note* — This is because abbreviated form is considered part of the generic *date* syntax, and not specific to any *calendar*. It is intended that applications can parse *dual years* to extract the *historical date* and *logical date* even in unknown *calendars*.

> *Note* — [GEDCOM 5.5.1] includes support for *dual years*, though only with *dates* in the Gregorian *calendar*. This is unfortunate as *dual years* are principally used with *dates* in the Julian *calendar*, and the example given in [GEDCOM 5.5.1] is seemingly in the Julian *calendar*. Nevertheless, many applications do implement *dual year* support. In [GEDCOM 5.5.1], *dual years* MUST be written in abbreviated form with the *logical year* shortened to two years. ELF relaxes this restriction as *dual years* are commonly found unabbreviated or with a single-digit *logical date*.
>
> *Dual years* in [GEDCOM 5.5.1] were probably only intended for years beginning on Lady Day, though the standard is not quite explicit in saying so; however the same facility readily accommodates years beginning on arbitrary days, and some current applications implement this. ELF generalises this further by allowing the *historical year* and *logical year* to differ arbitrarily in the generic *date* syntax, though of the *calendars* currently defined, only the Julian *calendar* allows *dual years*, and then only when the *historical year* and *logical year* differ by exactly one year.

When parsing a *dual year*, a *conformant* application MUST treat the second integer as an abbreviated *logical year* if it could have been produced by the serialisation rule for abbreviated *logical years*, above, and otherwise MUST NOT.

> *Example* — The *dual year* "`1616/8`" is not in abbreviated form because there is no *logical year* number ending in 8 which is at most one year away from the *historical year* 1618. This is therefore considered an unabbreviated *dual year*, representing the *historical year* 1618 and the *logical year* 8. Very probably this will result in the *date* not being *well-formed* in the applicable *calendar*, in which case the application MAY reject it.

Conformant applications MAY alter the lexical form of a *dual year* in any way, providing the *logical year* and *historical year* it represents is unchanged, and MUST NOT attach any significance to its particular lexical form.

> *Note* — This allows applications to store the *calendar year* component of the *date* as two integers internally, and regenerate its lexical form upon serialisation.

The *calendar* MAY define further restrictions on the range of integers permitted as *logical years*. Any such restrictions apply regardless of whether *dual years* are in used.

3.1.5 Epochs

The `Epoch` production encodes the *epoch name* component of the *date*, which specifies both the particular *epoch* from which *calendar years* are numbered and the direction of the numbering. Each *calendar* specifies the set of permitted *epoch names* components for that *calendar*.

> *Example* — The Julian *calendar* defined in §4.2 includes the *epoch names* "`B.C.`" and "`A.D.`", standing for Before Christ and Anno Domini, respectively. Both *epoch names* state that *calendar years* are counted from an *epoch* on the Julian date 1 January A.D. 1, nominally the date for the birth of Jesus Christ, but for "`B.C.`" *calendar years* are counted backwards from that *epoch*, while for "`A.D.`" *calendar years* are counted forwards.

> *Note* — The *epoch name* always comes after the *calendar year* component in the generic *date* syntax, despite certain *epoch names* being conventionally written before the *logical year* number in English. For example, it is normal to write "AD 1752" in English, but the ELF representation is "`1752 A.D.`". A *conformant* application MUST NOT accept "`A.D. 1752`" as a valid *date*, as it does not match the `Date` production.

The `Epoch` production requires *epoch names* either to be exactly two *characters* long, or to include at least one full stop (U+002E) or underscore (U+005F), or to begin with a dollar sign (U+0024).

> *Note* — This ensure that no *month name* is syntactically allowed as an *epoch name*, or vice versa. This ensures that a *string* like "`1 JAN`" cannot match the `Date` production.

Epoch names SHOULD normally be an acronym or initialism with full stops between letters. The two-letter form of *epoch names*, without a full stop, underscore or leading dollar, is deprecated. If an application encounters such a *epoch name*, it SHOULD convert it into an initialism by inserting a full stop between the two letters and another after the second letter.

> *Note* — Two-letter *epoch names* without full stops are allowed in ELF because “BC” is moderately frequently found in current GEDCOM files, even though [GEDCOM 5.5.1] requires it to be spelled “`B.C.`”. This rule converts “BC” to “`B.C.`”.

Epoch names beginning with a dollar sign are reserved for future use. *Conformant* applications MUST accept them, but *calendars* MUST NOT define any such *epoch names*.

> *Editorial note* — This specification of *epoch names* is designed to be usable to implement regnal years. In such a scheme, the *instant* of accession of each monarch would be an *epoch*, and a *forwards epoch name* would be used to reference it, for example “`25 OCT 3 HENRY_V`”. However to accommodate monarchs with a single name and no regnal number (e.g. Victoria or Tutankhamun), as well as to allow monarch names in foreign scripts, the dollar sign is reserved as a possible sigil for epoch names. This would allow “`$昭和`”, for example, to represent the *epoch* at the start of the Shōwa period in Japan.

The words “`TO`” and “`AT`” are reserved and MUST NOT be used as *epoch names* in any *calendar*.

> *Note* — The word “`AT`” is included here for future compatibility; the word “`TO`” is reserved because it is used in *date periods* in §3.4.

> *Editorial note* — An earlier draft of this standard included the following paragraph:
>
> > If a *calendar* defines multiple *epoch names* for the same *epoch*, and which are either all *forwards epoch names* or all *reverse epoch names*, applications SHOULD treat those *epoch names* as interchangeable.
>
> This was added to allow the Julian and Gregorian *calendars* to define “`B.C.E.`” as an alias for the “`B.C.`”, however the current draft does not do that, and it seems an unnecessary complication. If the BCE form is added, this paragraph needs reinstating.

A *calendar* MAY require an *epoch name* to be present, MAY allow it to be omitted, and MAY define no *epoch names* thereby requiring it to be omitted. In calendars where the *epoch name* is OPTIONAL, the *calendar* SHOULD define an explicit *epoch name* that is equivalent to an omitted *epoch name*. This *epoch name* is called the **default epoch name** for the calendar.

> *Example* — The Julian calendar defined in §4.2 provides the “`A.D.`” *epoch name* to serve as its *default epoch name*.

> *Editorial note* — The French Republican calendar, as currently defined in §4.3, does not properly define an *epoch name*, or equivalently leaves its sole *epoch name* anonymous. This is allowed, but is *not recommended*. If the French Republican *epoch name* remains anonymous, the above text should perhaps be changed.

3.2 Date modifiers

The generic *date* syntax described in §3.1 provides a means of specifying a particular *date*, but does not provide a way of describing *dates* which are *approxiately stated*, nor of giving natural language descriptions, nor of stating a *imprecise date* via its *precision range*. ELF provides several extensions to the generic *date* syntax to provide this functionality. Syntactically, these are expressed by prefixing the generic *date* syntax with a token called **date modifier**.

> *Example —* An "ABT" *date modifier* is introduced in §3.2.1. It is placed before the *date*, as in "ABT JAN 1901".

> *Editorial note —* This draft, like [GECOM 5.5.1], does not allow more than one of these facilities to be used on the same *date*. It seems potentially useful to remove this restriction, for example by allowing "FROM ABT 1798 TO 2 JAN 1842". There would be no ambiguity in the grammar with such constructs and their meaning would generally be well defined. Backwards compatibility could be handled by saying that applications SHOULD NOT produce such constructs but MUST be able to read them. This would also allow constructs such as "EST ABT 1881", which the GEDCOM grammar does not allow but which appears as an example in [GEDCOM 5.5.1], albeit as an example of an unnecessary circumlocution. Should the initial version of ELF support such things?

3.2.1 Approximated dates

A *date* MAY be preceded by one of the tokens ABT, CAL or EST, as shown in the following DateApprox production:

```
DateApprox  ::= ( "ABT" | "CAL" | "EST" ) S Date
```

The ABT token indicates that the *date* is *approximately stated*, and that its *precision* is lower than would have been the case without the ABT token. It is typically used when there is evidence that the *date* given is roughly correct, and SHOULD NOT be used when the *date* is estimated using statistical likelihoods or cultural norms.

> *Example —* If it is known that the first and third children in a family were born in 1897 and 1900, the second child's birth MAY be recorded as "ABT 1899" as the *date* of birth is fairly well bounded, even if twins were involved.

> *Note —* The ABT token is currently by far the most commonly used of the three approximated tokens.

The EST token also indicates that *date* is *approximately stated*, and is an estimate perhaps based only on statistical likelihoods or cultural norms. An application MAY assume a *date* written with the EST token has lower *precision* than one with the ABT token.

> *Example* — If it is known that the first three children in a family of four were born between 1897 and 1900, a researcher might conclude that it is probable that next child was born shortly afterwards and estimate the *date* of birth as "`EST 1903`". The `ABT` token SHOULD NOT be used in this case, unless there is additional evidence that the fourth child was not much younger.

> *Example* — It can sometimes be useful to provide a crude estimate of an individual's *date* of birth. If a baptism register records a baptism in 1692 and gives the father's name, a researcher might wish to record an estimate of the father's *date* of birth, perhaps to help disambiguate him from other people of the same name. If such an estimate is to recorded, it should use the `EST` token, for example, "`EST 1660`".

The `CAL` token is used to record a *date* that is not directly stated in a source, but instead has been calculated from other known values, typically another *date* and the *duration* separating the two *dates*. An application MAY assume a *date* written with the `CAL` token has a similar or higher *precision* than one written with the `ABT` token.

> *Example* — If a newspaper reports on a golden wedding celebration in the summer of 1948, this is evidence that the wedding happened 50 years previously, and the date of the marriage MAY be entered as "`CAL 1898`".

When a *date* is calculated from another *date* and an *imprecise duration* separating them, use of the `ABT` token is RECOMMENDED instead of the `CAL` token.

> *Example* — Like many censuses, the 1880 census of Iceland records individuals' *ages* on their most recent birthday. A person recorded as 72 could have been born in 1807 or 1808. Because the *age* has been rounded down and is therefore somewhat *imprecise*, it is RECOMMENDED that the *date* of birth be recorded as "`ABT 1808`" rather than "`CAL 1808`".

> *Note* — A `CAL` token might sometimes be an indicator that the *date* is of lower *reliability* than one without this token, everything else being equal, due to it not being directly stated.

> *Editorial note* — Should FHISO deprecate the `CAL` token and allow uses of it to be replaced with `ABT`? It's not clear it adds anything useful.

3.2.2 Date phrases and interpreted dates

ELF allows *dates* to be recorded in natural language. The natural language description of a *date* is called a **date phrases** and is written between parentheses (U+0028 and U+0029).

```
DateInterp  ::= ( "INT" S Date S )? "(" DatePhrase ")"
DatePhrase  ::= [^#xA#xD()]*
```

The *date phrase* is an arbitrary *language-tagged string* except that it MUST NOT contain parentheses (U+0028 or U+0029), line feeds (U+000A) or carriage returns (U+000D).

> *Note* — [GEDCOM 5.5.1] only requires that *date phrases* be "enclosed in matching parentheses", which could be interpreted as requiring that any parentheses in the *date phrase* be balanced. ELF goes further and prohibits parentheses from occurring in *date phrases* at all.

> *Editorial note* — The prohibition on these *characters* is not strictly needed, at least while the grammar does not allow *date phrases* within *date ranges* or *date periods*, though that is a possible extension which might be made in a future version of ELF. A data model layer escape mechanism along the lines of the `%{…}` syntax used in FHISO's draft Creator's Name microformat could allow arbitrary *characters* to appear in it.

A *date phrase* SHOULD be a fragment of text quoted from a source, possibly in translation, and SHOULD only normally be used if the text cannot readily be converted into a *date*, or where that conversion would lose useful information. *Date phrases* SHOULD NOT be used for arbitrary annotations or commentary, and MUST NOT be used in a negative sense to say the event did not occur.

> *Example* — If a source describes an event as happening on "The Feast of St John" in a particular year, this might be recorded in a *date phrase* if there was insufficient context to establish whether it referred to the Feast of Nativity of St John the Baptist on 24 June, or the Feast of St John the Evangelist on 27 December.

> *Note* — Historically, some applications have used a *date phrase* like "Not married" to indicate a marriage did not occur. Such usage is prohibited in ELF and was not allowed in [GEDCOM 5.5.1]. *Conformant* applications are permitted to assume a *date phrase* describes an actual *date*, even if the *date phrase* cannot be interpreted. Fortunately, uses like "Not married" is largely confined to [GEDCOM 5.3] and earlier which omitted the parentheses around *date phrases*. *Date phrases* written without parentheses will not normally match the generic *date* syntax, and will therefore not be parsed as *dates* in ELF.

The *date phrase* MAY be accompanied by a *date* in the generic *date* syntax preceded by the `INT` token. The combination is called an **interpreted date** and its *date* SHOULD be an interpretation of the associated *date phrase* in the context of source containing it.

> *Example* — "`INT @#DJULIAN@ 18 JUNE 1502 (Saturday before the Feast of the Nativity of St John the Baptist)`" is a valid *interpreted date*. In this case the *date phrase* does not mention the year: perhaps it was inferred from its context or was stated elsewhere in the document.

> *Note* — An *interpreted date* might sometimes be of lower *reliability* than if the *date* were written without a *date phrase*, as the presence of a *data phrase* typically indicates some subtlety in interpreting the *date*.

Being a *language-tagged string*, a *date phrase* has a *language tag* associated with it. This *language tag* is not embedded in the *date* but is provided externally by the serialisation format.

> *Note* — In [ELF Serialisation], the *language tag* is provided by a `LANG` tag, which MAY be a substructure of the `DATE` tag, failing which it MAY be a substructure of one of the superstructure of the `DATE` tag, failing which it MAY be in the ELF header. For example,
>
> ```
> 2 DATE INT 25 JAN 1840 (L'an mil huit cent quarante, le
>   → vingt-cinquième jour du mois de janvier)
> 3 LANG French
> ```

> *Editorial note* — Review the previous note once [ELF Serialisation] has been developed further.

3.2.3 Date ranges

A **date range** is a *time interval* used to record an unknown *date* which can be placed within certain bounds. A *date range* MAY be bounded from below, from above, or both. It matches the `DateRange` production:

```
DateRange   ::= ( "BEF" | "AFT" ) S Date | "BET" S Date S "AND" S Date
```

A *date range* beginning with a `BEF` token represents an unknown *instant* on or before the specified *date*. A *date range* beginning with a `AFT` token represents an unknown *instant* on or after the specified *date*.

It is RECOMMENDED that the `BEF` and `AFT` forms of *date ranges* are only used if it is believed the specified *date* is probably within a few years of the unknown *date* being represented.

> *Example* — If an individual was elected mayor of a city in 1745 and the cultural norms of the time mean the person was likely a middle-aged adult, their *date* of birth SHOULD NOT be recorded as "`BEF 1745`". Even though this would literally be true, insofar as the person was indeed born before 1745, and this representation is not prohibited, it is NOT RECOMMENDED. The person could well have been born in the previous century which would be stretching

> the definition of “a few years”. If there is need to give a date of birth, a crude estimate given with the `EST` token might be preferable.

A *date range* using the `BET` … `AND` construct represents an unknown *instant* on or after the first specified *date*, and on or before the second specified *date*. This is a way of specifying an unknown *date* in terms of its *precision range*. In this form of *date range*, the first specified *date* MUST NOT be later than the second specified *date*.

> *Example* — If someone’s *age* is recorded as 26 on a census conducted on 3 April 1881, their *date* of birth could be recorded as “`BET 4 APR 1854 AND 3 APR 1855`”. However the usual practice in GEDCOM is to record this as “`ABT 1855`”.

> *Editorial note* — Should ELF encourage users to prefer the *date range* form in the previous example?

Date ranges using the `BET` … `AND` format contain two *dates* and therefore potentially two *calendar escapes*. The second *date* does not inherit the *calendar escape* from the first date, and it MUST be respecified when required. It is RECOMMENDED that, where possible, the same *calendar escape* be used on both *dates*.

> *Note* — Although [GEDCOM 5.5.1] supports *date ranges* where the two *dates* use different *calendar escapes*, not all current implementations support them. They are best avoided where possible.

3.3 The `elf:DateValue` datatype

The `elf:DateValue` *datatype* is used to record historical *dates*. It allows *dates* to be represented using the generic *date* syntax or any of the modifier forms given in §3.2. It MAY also be a *date period* as defined in §3.4. Its *lexical space* is the set of *strings* which match the following `DateValue` production:

```
DateValue ::= Date | DateApprox | DateInterp | DateRange | DatePeriod
```

> *Editorial note* — In the many places in ELF, it is wrong to use a *date period* where an `elf:DateValue` is required. This is not the case with the other options allowed in the `DateValue` production. The difference is that a *date period* represents a state which persisted for an extended *time interval*, while the others represent a single *instant* (or at least an event of short duration that is conveniently approximated to an *instant*). FHISO are considering removing `DatePeriod` from the `DateValue` production, above, and allowing it as an explicit option in certain contexts. However this introduces the same complications as introducing separate `elf:AgeWord` *datatype*, and which are discussed in an editorial note at the end of §6.

The `elf:DateValue` *datatype* is a *language-tagged datatype*, however the associated *language tag* is ignored unless the *date* contains a *date phrase*. *Conformant* applications are not required to preserve the *language tag* of *dates* without a *date phrase*, and MAY replace it with an arbitrary *language tag*.

> *Note* — This allows applications to store the *language tag* as part of the *date phrase*, and allows them to serialise an `elf:DateValue` that does not contain a *date phrase* using an arbitrary *language tag*. In particular, this means a *date* read from an ELF file in one language MAY be serialised as part of an ELF file in another language without explicitly annotating it with a `LANG` substructure, unless the *date* contains a *date phrase*.

Formally, the `elf:DateValue` *datatype* is a *structured language-tagged datatype* which has the following properties:

Datatype definition

Name	`https://terms.fhiso.org/elf/DateValue`
Type	`http://www.w3.org/2000/01/rdf-schema#Datatype`
Pattern	*See below*
Supertype	`http://www.w3.org/1999/02/22-rdf-syntax-ns#langString`
Abstract	`false`

The *pattern* for this *datatype* is as follows:

```
((ABT|CAL|EST|BEF|AFT|TO)[ \t\r\n]+)?(@#D[A-Z][A-Z ]*@[ \t\r\n]+)?(([0-9]+[ \t\r\n]+)?[A-Z]
[A-Z0-9][A-Z0-9]+[ \t\r\n]+)?-?[0-9]+(/-?[0-9]+)?([ \t\r\n]*([A-Z]([A-Z]|[A-Z0-9._]*[._]
[A-Z0-9._]*)|\$[^ #x9#xA#xD]+)|(INT[ \t\r\n]+(@#D[A-Z][A-Z ]*@[ \t\r\n]+)?(([0-9]+[ \t\r\n]+
)?[A-Z][A-Z0-9][A-Z0-9]+[ \t\r\n]+)?-?[0-9]+(/-?[0-9]+)?([ \t\r\n]*([A-Z]([A-Z]|[A-Z0-9._]*
[._][A-Z0-9._]*)|\$[^ #x9#xA#xD]+)[ \t\r\n]+?)\([^\r\n()]*\)|BET[ \t\r\n]+(@#D[A-Z][A-Z ]*
@[ \t\r\n]+)?(([0-9]+[ \t\r\n]+)?[A-Z][A-Z0-9][A-Z0-9]+[ \t\r\n]+)?-?[0-9]+(/-?[0-9]+)?
([ \t\r\n]*([A-Z]([A-Z]|[A-Z0-9._]*[._][A-Z0-9._]*)|\$[^ #x9#xA#xD]+)[ \t\r\n]+AND[ \t\r\n]+
(@#D[A-Z][A-Z ]*@[ \t\r\n]+)?(([0-9]+[ \t\r\n]+)?[A-Z][A-Z0-9][A-Z0-9]+[ \t\r\n]+)?-?[0-9]+
(/-?[0-9]+)?([ \t\r\n]*([A-Z]([A-Z]|[A-Z0-9._]*[._][A-Z0-9._]*)|\$[^ #x9#xA#xD]+)
|FROM[ \t\r\n]+(@#D[A-Z][A-Z ]*@[ \t\r\n]+)?(([0-9]+[ \t\r\n]+)?[A-Z][A-Z0-9][A-Z0-9]+
[ \t\r\n]+)?-?[0-9]+(/-?[0-9]+)?([ \t\r\n]*([A-Z]([A-Z]|[A-Z0-9._]*[._][A-Z0-9._]*)
|\$[^ #x9#xA#xD]+)([ \t\r\n]+TO[ \t\r\n]+(@#D[A-Z][A-Z ]*@[ \t\r\n]+)?(([0-9]+[ \t\r\n]+)?
[A-Z][A-Z0-9][A-Z0-9]+[ \t\r\n]+)?-?[0-9]+(/-?[0-9]+)?([ \t\r\n]*([A-Z]([A-Z]|[A-Z0-9._]*
[._][A-Z0-9._]*)|\$[^ #x9#xA#xD]+))?
```

> *Note* — The grammar productions given in this standard provide a more useful description of *datatype* for a human reader. The *pattern* above is used during *datatype correction* in certain OPTIONAL contexts. It was generated automatically from the grammar productions, and any discrepancy between the grammar and this *pattern* is unintentional.

3.4 The `elf:DatePeriod` datatype

A **date period** is a *time interval* used to record a state of being that persisted throughout the stated *time interval* which represents an extended period of history; they are also used to record the period of coverage of a source. They are represented in ELF using the `elf:DatePeriod` *datatype* whose *lexical space* is set set of *strings* which match the following `DatePeriod` production:

```
DatePeriod  ::= "FROM" S Date ( S "TO" S Date )? | "TO" S Date
```

Use of a *date period* does not necessarily mean the state was believed to have begun on the specified "from" *date*, nor that it ended on the specified "to" *date*, though where possible this is RECOMMENDED. Either the start *date* or the end *date* MAY be omitted from the *date period*, in which case the missing *date* is interpreted as an unknown *date*.

> *Example* — Abraham Lincoln was President of the United States from 1861 to 1865. This could recorded using the *date period* "`FROM 1861 TO 1865`", or more *precisely* as "`FROM 4 MAR 1861 TO 15 APR 1865`". If it were unknown when Lincoln's presidency began, it could be written "`TO 15 APR 1865`". If it was also known that Lincoln was president at the time of the Battle of Gettysburg in July 1863, the *date period* of his presidency could be written "`FROM JUL 1863 TO 15 APR 1865`". This does not imply Lincoln became president in 1863, only that he was president for the whole of the period from 1863 to 1865.

> *Editorial note* — Neither GEDCOM nor this draft of ELF has a way of saying whether the *date period* is believed to be the complete period during which the state persisted. This is perhaps a shortcoming that a future version of ELF should resolve. The obvious solution is to say that a *date period* always represents the full period, and allow the various *date modifiers* from §3.2 to be used in the end *instants* of the *date period*. In the previous example where Lincoln was known to have been president for the Battle of Gettysburg, and to have died in office on 15 April 1865, this could be written "`FROM BEF JUL 1863 TO 15 APR 1865`". This could be done without allowing arbitrary nested *date modifiers*.

> *Note* — The difference between a *date range* and a *date period* is sometimes misunderstood, and applications should be alert to the wrong representation being used. A *date range* is used to record an *instant*, or an event of short duration, when its *date* is not *precisely* known, whereas a *date period* is used to record a state that persisted throughout the specified *time interval*.

Example — Civil birth, marriage and death registrations in England and Wales are indexed in quarterly volumes which do not list the *precise date* of the event. The coverage of this source is represented in ELF using a *date period*:

```
0 @S1@ SOUR
1 TITL GRO Marriages Index
1 DATA
2 EVEN MARR
3 DATE FROM 1 JAN 1898 TO 31 MAR 1898
```

However the event itself MUST NOT be recorded using a *date period* because the marriage did not take place over an extended period: it happened on one particular *date* which is not *precisely* known. For this purpose a *date range* SHOULD be used.

```
0 FAM
1 MARR
2 DATE BET 1 JAN 1898 AND 31 MAR 1898
```

A less *precise* alternative would be to just give a year.

Datatype definition

Name	`https://terms.fhiso.org/elf/DatePeriod`
Type	`http://www.w3.org/2000/01/rdf-schema#Datatype`
Pattern	*See below*
Supertype	`http://www.w3.org/1999/02/22-rdf-syntax-ns#langString`
Abstract	`false`

The *pattern* for this *datatype* is as follows:

```
FROM[ \t\r\n]+(@#D[A-Z][A-Z ]*@[ \t\r\n]+)?(([0-9]+[ \t\r\n]+)?[A-Z][A-Z0-9][A-Z0-9]+
[ \t\r\n]+)?-?[0-9]+(/-?[0-9]+)?([ \t\r\n]*([A-Z]([A-Z]|[A-Z0-9._]*[._][A-Z0-9._]*)
|\$[^ #x9#xA#xD]+)([ \t\r\n]+TO[ \t\r\n]+(@#D[A-Z][A-Z ]*@[ \t\r\n]+)?(([0-9]+[ \t\r\n]+)?
[A-Z][A-Z0-9][A-Z0-9]+[ \t\r\n]+)?-?[0-9]+(/-?[0-9]+)?([ \t\r\n]*([A-Z]([A-Z]|[A-Z0-9._]*
[._][A-Z0-9._]*)|\$[^ #x9#xA#xD]+))?
```

4 Calendar definitions

This section defines four standard *calendars* for use with ELF: the Gregorian, Julian, Hebrew and French Republican *calendars*. *Conformant* applications are REQUIRED to support all four.

Editorial note — There is no technical need for applications to support anything other than the Gregorian *calendar*. Would it be better to make only this one REQUIRED and the others RECOMMENDED? Ideally, applications would do more than simply check *dates* are syntactically valid and would offer conversions between supported *calendars*, *date* arithmetic, the ability to look up the day of the week and the *dates* of important feast days. Few applica-

> tions currently offer this. Would reducing the number of REQUIRED *calendars* make it easier for vendors to provide such functionality?

> *Editorial note* — The definitions given in this section only specify those details which are necessary to determine whether a *date* is *well-formed* in the given *calendar*. Should these descriptions be expanded to give more details to help applications convert between *calendars*?

4.1 The Gregorian calendar

> *Note* — The Gregorian *calendar* is the name given to the now ubiquitous *calendar* introduced by Pope Gregory XIII in 1582 to correct the Julian *calendar* which was slowly drifting relative to the seasons. It is represented by the `@#DGREGORIAN@` *calendar escape*, and is also ELF's default *calendar*, used whenever a *date* has no *calendar escape* and is *well-formed* in the Gregorian *calendar*.

The Gregorian *calendar* has an *epoch* at the start of the *calendar day* 1 January 1 AD, as expressed in the Gregorian *calendar*, and two *epoch names* relative to that *epoch*: a *reverse epoch name* "`B.C.`", and a *forwards epoch name* "`A.D.`". The latter is the *default epoch name* for the *calendar*, and its name SHOULD be omitted.

> *Example* — The *date* "`24 DEC 2018 A.D.`" is equivalent to "`24 DEC 2018`". The latter is RECOMMENDED for compatibility with [GEDCOM 5.5.1] which does not support the "`A.D.`" *epoch name*.

> *Editorial note* — Do we want "`B.C.E.`" and "`C.E.`" (standing for Before Common Era and Common Era, respectively) as aliases? There is no technical justification for adding them.

Regardless of *epoch name*, the *logical year* SHALL be an integer greater than 0.

> *Note* — This prohibits negative or zero year numbers as they are not needed. The year before "`1 A.D.`" is "`1 B.C.`".

> *Note* — [GEDCOM 5.5.1] says that *logical years* MUST be 3 or 4 digits long, and presumably requires dates in the first century to be zero padded. This standard has no such requirement, and many current applications do not enforce this requirement.

Dual years MUST NOT be used in the Gregorian *calendar*.

> *Note* — This is a deviation from [GEDCOM 5.5.1] which allows *dual years* only on Gregorian *dates*. In this standard, a *date* with a *dual year* is not *well-formed* in the Gregorian *calendar*. This means a *date* using a *dual year* and no explicit *calendar escape* will be assigned the `@#DUNKNOWN@` *calendar escape*.

> *Editorial note* — In practice there is a very strong likelihood that the Julian *calendar* is intended. This draft could have altered the default *calendar* rules in §3.1.1 so that *dates* using *dual years* and no explicit *calendar escape* were automatically labelled `@#DJULIAN@`. The reason this was not done is that an ELF file containing such *dates* is likely to have many other miscalendared *dates* but which are *well-formed* in the Gregorian *calendar* and so go undetected. Flagging those with *dual years* with `@#DUNKNOWN@` will hopefully bring this to the researcher's attention, with the result that all the miscalendared *dates* are fixed.

Every *calendar year* in the Gregorian *calendar* consists of 12 *calendar months*. Their *month names* are given in the table below in order of their occurrence in the *calendar year*. The table also gives the usual form of their name in English, and the number of *calendar days* in each month. The *calendar days* in each *calendar month* are numbered sequentially starting with 1.

`JAN`	January	31 days
`FEB`	February	28 or 29 days — see below
`MAR`	March	31 days
`APR`	April	30 days
`MAY`	May	31 days
`JUN`	June	30 days
`JUL`	July	31 days
`AUG`	August	31 days
`SEP`	September	30 days
`OCT`	October	31 days
`NOV`	November	30 days
`DEC`	December	31 days

The number of *calendar days* in February varies depending on the *logical year*. The rules for determining this number in years with the "`A.D.`" *epoch name* are as follows:

— If the *logical year* number is exactly divisible by 400, then February has 29 days.
— Otherwise, if the *logical year* number is exactly divisible by 100, then February has 28 days.
— Otherwise, if the *logical year* number is exactly divisible by 4, then February has 29 days.
— Otherwise, February has 28 days.

In the Gregorian *calendar*, a *calendar year* in which February has 29 *calendar days* is called a **leap year**, while a *calendar year* in which February has only 28 *calendar days* is called a **non-leap year**.

> *Note* — Other *calendars* defined the phrases *leap year* and *non-leap year* differently.

For years with the "`B.C.`" *epoch name*, the *logical year* number is subtracted from one to get zero or a negative number, which is then used in place of the *logical year* in the preceding rules.

> *Example* — The year 5 BC was a *leap year* in the proleptic Gregorian *calendar*, meaning February had 29 *calendar days*. This is because subtracting 5 from 1 gives −4 which is exactly divisible by 4.

> *Note* — Although the Gregorian *calendar* was first introduced in 1582, ELF allows its use proleptically, including for *dates* BC. This is a partial departure from [GEDCOM 5.5.1] which only allows *incomplete dates* referencing only a *calendar year* to use the "`B.C.`" *epoch name*.

A *date* which uses a *calendar day* number which is greater than the number of *calendar days* in the specified year and month is not a *well-formed date*.

> *Example* — The *date* "`29 FEB 2018`" is not *well-formed* in the Gregorian *calendar* because February only had 28 days in 2018 due to 2018 not being exactly divisible by 4.

4.1.1 The `elf:DateExact` datatype

The `elf:DateExact` *datatype* is used to record the creation or modification *date* of various objects in the ELF data model using the Gregorian *calendar*. The *lexical space* of this *datatype* is the set of *strings* which match the following `GregDate` production and which are additionally *well-formed dates* in the Gregorian *calendar* defined in §4.1.

```
GregDate  ::= GregDay S GregMonth S GregYear

GregDay   ::= [0-9] [0-9]?
GregMonth ::= "JAN" | "FEB" | "MAR" | "ARP" | "MAY" | "JUN" | "JUL"
              | "AUG" | "SEP" | "OCT" | "NOV" | "DEC"
GregYear  ::= [0-9] [0-9] [0-9] [0-9]
```

> *Example* — The *string* "`24 DEC 2018`" matches the `GregDate` production and is a *well-formed date* in the Gregorian *calendar*. It is therefore in the *lexical space* of `elf:DateExact`. "`54 NOV 2018`" is not in the *lexical space* of `elf:DateExact` despite matching the `GregDate` production because it is not a *well-formed date* in the Gregorian *calendar* due to November only having 30 days.

> *Note* — This *datatype* is more restricted than a Gregorian *date* written in the generic *date* syntax due to the following additional constraints:
>
> — There MUST NOT be a *calendar escape*.
> — There *calendar day* and *calendar month* MUST be present.
> — The *calendar year* component MUST be four digits long.
> — The *epoch name* MUST NOT be present.
> — The *date modifiers* defined in §3.2 MUST NOT be used.
> — It MUST NOT be a *date period*.

Formally, the `elf:DateExact` *datatype* is a *structured non-language-tagged datatype* which has the following *properties*:

Datatype definition

Name	`https://terms.fhiso.org/elf/DateExact`
Type	`http://www.w3.org/2000/01/rdf-schema#Datatype`
Pattern	`[0-9]{1,2}[ \t\r\n](JAN|FEB|MAR|ARP|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DEC)[ \t\r\n][0-9]{4}`
Supertype	*No non-trivial supertypes*
Abstract	`false`

> *Note* — The *pattern* in the table above has been split on to three lines for convenience of presentation; it is, however, really one *pattern* and contains no *whitespace* or line breaks. Any functional difference between the `GregDate` production and the *pattern* specified above is unintentional.

> *Note* — The `elf:DateExact` *datatype* is not specified as a *subtype* of the `elf:DateValue` *datatype* because the latter is a *language-tagged datatype*.

4.2 The Julian calendar

> *Note* — The Julian *calendar* is the name given to the *calendar* introduced by Julius Caesar in 45 BC and subsequently amended by Augustus in about 8 BC to correct an error in the application of its *leap year* rule during its first three decades. The Augustan form of the Julian *calendar* is represented by the `@#DJULIAN@` *calendar escape*. Because the Gregorian *calendar* derived from the Julian *calendar*, many of the details here are the same as those in §4.1.

The Julian *calendar* has an *epoch* at the start of the *calendar day* 1 January 1 AD, as expressed in the Julian *calendar*, or 30 December 1 BC, as expressed in the proleptic Gregorian *calendar*. The Julian *calendar* has the same two *epoch names*, "`B.C.`" and "`A.D.`", as the Gregorian *calendar*, but defined relative to the Julian *epoch* instead of the Gregorian *epoch*. The "`A.D.`" *epoch name* is the *default epoch name* for the *calendar*, and its name SHOULD be omitted.

As for the Gregorian *calendar*, regardless of *epoch name*, the *logical year* SHALL be an integer greater than 0.

Dual years are only allowed when the *logical year* and the *historical year* differ by exactly one year. They are used to represent the differing conventions for the first day of the year.

> *Note* — This is a deviation from [GEDCOM 5.5.1] where *dual years* are not permitted with the Julian *calendar*.

> *Note* — *Logical years* always begin on 1 January, but *historical years* can begin on an arbitrary day of the year which can result in the *historical year* being ahead or behind the *logical year*. Because the use of *dual years* is not restricted to just years beginning on Lady

> Day, it is not in general possible to tell simply from the fact that a *dual year* was used what convention for the start of the year. This can usually be inferred from context, and the *logical year* allows an application to process the *date* without knowing this.

> *Example* — In France, the year was reckoned to begin on Easter Sunday until 1564 (or 1565 — the change did not happen uniformly). An application cannot tell when reading the *date* "`10 FEB 1521/22`" whether it has been written using the French convention of starting the year on Easter Sunday or the English convention of starting the year on Lady Day, but it is not normally necessary to know this as this *date* refers to the same *calendar day* regardless of when the *historical year* began.
>
> This is an example where the *historical year* alone could be ambiguous due to Easter being a movable feast. In France, there were two days which would have been written "1er jour d'avril, 1522", as Easter was early in the *logical year* 1522 and late in 1523. For both days described as 1 April 1522, the *historical year* is 1522, but only for the first is the *logical year* also 1522: for the second, it is 1523. This means the first *date* MUST be encoded in ELF as "`1 APR 1522`", while the second SHOULD be encoded as "`1 APR 1522/23`" (though it MAY alternatively be encoded as "`1 APR 1523`", without using a *dual year*).

Every *calendar year* in the Julian *calendar*, as reckoned by *logical year*, consists of the same 12 *calendar months* as the Gregorian calendar. Their *month names*, the usual English form of their name, and the number of *calendar days* in each *calendar month* is as given for the Gregorian calendar in the table in §4.1. The only difference between the *calendars* is the rule for determining the number of *calendar days* in February.

In the Julian *calendar*, the rules for determining the number of *calendar days* in February are as follows:

— If the *logical year* number is exactly divisible by 4, then February has 29 days.
— Otherwise, February has 28 days.

The terms *leap year* and *non-leap year* are defined as for the Gregorian *calendar*.

> *Example* — The year 1900 was a *leap year* in the Gregorian *calendar*, as 1900 is divisible by 100, but it was not a *leap year* in the Julian *calendar*. This means "`29 FEB 1900`" is a *well-formed date* in the Gregorian *calendar*, but not in the Julian *calendar*.

For years with the "`B.C.`" *epoch name*, the *logical year* number is subtracted from one to get zero or a negative number, which is then used in place of the *logical year* in the preceding rules.

> *Note* — The year 5 BC was a *leap year* according to ELF's definition of the Julian *calendar*, because subtracting 5 from 1 gives −4 which is divisible by 4. In fact, the year 5 BC was almost certainly not a *leap year* as the Augustan reform was still taking effect at this point. There is disagreement between historians on the exact details of the reform, but is generally accepted that by AD 8 the rule for *leap years* given above was being applied correctly. Any

> use of the Julian calendar (in its final Augustan form) before about AD 5 should therefore be regarded as proleptic.

A *date* which uses a *calendar day* number which is greater than the number of *calendar days* in the specified year and month is not a *well-formed date*.

4.3 The French Republican calendar

> *Note* — The French Republican *calendar* or French Revolutionary *calendar* are the names given to the new *calendar* adopted in 1794 by the French National Convention. It is represented in ELF by the `@#DFRENCH R@` *calendar escape*.

The French Republican *calendar* has an *epoch* at the start of the Gregorian *calendar day* 22 September 1792, the date the First French Republic was founded. This *date* is identified as 1 Vendémiaire I in the new *calendar*.

> *Note* — It is common to write French Republican *calendar years* using Roman numerals.

The *calendar* has a single anonymous *forwards epoch name*. It does not provide a *backwards epoch name* for referring to *dates* before the founding of the First Republic, nor are zero or negative *logical years* permitted.

> *Editorial note* — If it proves undesirable to have an anonymous *epoch name*, the usual phrase are "l'époque républicaine" and "l'ère de la republique". An *epoch name* of "`E.R.`" would therefore seem appropriate.

The *logical year* SHALL be an integer greater than 0 and SHOULD be no greater than 18. Applications MAY consider any *date* with a *logical year* greater than 18 to be not *well-formed*.

> *Note* — The placement of *leap years* in the French Revolutionary *calendar* was never defined satisfactorily due to an ambiguity in the legislation that had not been resolved when the *calendar* was abolished. The *calendar* was in actual use during years II to XIV, and the placement of *leap years* in the period between years I and XVIII is unambiguous. The *calendar* was used again very briefly during the Paris Commune in May 1871 (Floréal and Prairial LXXIX), and applications are encouraged to support this, but ELF does not require this as the number of sources of genealogical relevance from the Paris Commune is likely to be small.

Dual years MUST NOT be used in the French Republican *calendar*.

Every *calendar year* in the French Republican *calendar* consists of 12 *calendar months*, which are followed by 5 or 6 intercalary days or **jours complémentaires** which ELF treats as a thirteenth month. Their *month names* are given in the table below in order of their occurrence in the *calendar year*. The table also gives the usual form of their name in French, and the number of *calendar days* in each month. The *calendar days* in each *calendar month* are numbered sequentially starting with 1.

VEND	Vendémiaire	30 days
BRUM	Brumaire	30 days
FRIM	Frimaire	30 days
NIVO	Nivôse	30 days
PLUV	Pluviôse	30 days
VENT	Ventôse	30 days
GERM	Germinal	30 days
FLOR	Floréal	30 days
PRAI	Prairial	30 days
MESS	Messidor	30 days
THER	Thermidor (or Fervidor)	30 days
FRUC	Fructidor	30 days
COMP	Jours complémentaires	5 or 6 days — see below

> *Note* — The month of Thermidor was also called Fervidor, however ELF uses the *month name* THER regardless of which form the source uses. FERV MUST NOT be used.

In the French Republican *calendar*, a *calendar year* with 6 *jours complémentaires* is called a **leap year**, while a *calendar year* with only 5 *jours complémentaires* is called a **non-leap year**. If the *logical year* number is 3, 7, 11 or 15, the *calendar year* was a *leap year*; all other years with *logical year* numbers between 1 and 18, inclusive, were *non-leap years*.

> *Note* — This standard does not specify when *leap years* occurred after year XVIII because the legislation was ambiguous. The legislation required *leap years* to be arranged such that the autumnal equinox would fall on the first day of the year, 1 Vendémiaire, as observed from the Paris Observatory. Under this rule, the *leap year* after year XV would have been five years later, in year XX. However the legislation also said that *leap years* were every four years, in which case the next *leap year* would have been XIX. A solution was proposed by Charles-Gilbert Romme, one of the creators of the *calendar*, which would have placed a *leap year* in year XX, and then one every four years with similar rules as the Gregorian *calendar* applying every century. No decision had been made when Napoléon abolished the *calendar*, so no definitive statement can be made on which subsequent years were *leap years*, and this standard does not require any particular interpretation. Applications MAY do any of the above and remain *conformant*; they MAY also reject anything after year XVII as not being *well-formed*.

A *date* which uses a *calendar day* number which is greater than the number of *calendar days* in the specified year and month is not a *well-formed date*. This provision applies to *jours complémentaires* too.

4.4 The Hebrew calendar

> *Note* — The Hebrew *calendar* is the name given to the *calendar* used by Jewish peoples around the world which developed into its current form in the early ninth century. It is represented in ELF by the `@#DHEBREW@` *calendar escape*.

The Hebrew *calendar* has an *epoch* at the start of the *calendar day* referred to as 7 September 3761 BC in the proleptic Gregorian *calendar*. This *date* is identified as 1 Tishrei AM 1 in the Hebrew *calendar*.

> *Note* — The *date* of this *epoch* is more commonly quoted as 7 October 3761 BC, which is its *date* in the proleptic Julian *calendar*. This is the *date* of the start of the Hebrew *calendar year* traditionally believed to contain the Creation.

> *Editorial note* — A day in the Hebrew *calendar* is normally reckoned to begin at sunset, which means the *epoch* should properly be at sunset on 6 September 3761 BC in the proleptic Gregorian *calendar*, or 6 October 3761 BC in the proleptic Julian *calendar*. However, as noted in §2.1, the definition of a *calendar day* given in [ISO 8601] and adopted here says a *calendar day* starts at midnight.

The *calendar* has a single *forwards epoch name*, "`A.M.`", which is the *default epoch name* for the *calendar*. It SHOULD be omitted when serialising *dates* in ELF. The *calendar* does not provide a *backwards epoch name* for referring to *dates* before the Hebrew *epoch*, nor are zero or negative *logical years* permitted.

> *Note* — [GEDCOM 5.5.1] does allow the "`B.C.`" *epoch name* to be used with the Hebrew *calendar*, but it seems almost certain this is an error in that standard. ELF does not permit "`B.C.`" to be used with this *calendar*, and any Hebrew *dates* using of it will not be *well-formed date*.

It is NOT RECOMMENDED for *dual years* to be used with the Hebrew *calendar*.

> *Note* — This is not an outright prohibition on *dual years*, and alternative calculations of the year of Creation were used to number *calendar years* in early times. These could be used as *historical years*, typically differing from the *logical year* by no more than three years.

Every *calendar year* in the Hebrew *calendar* consists of either 12 or 13 *calendar months*. Their *month names* are given in the table below in order of their occurrence in the *calendar year*. The table also gives the usual form of their name in English and Hebrew, and the number of *calendar days* in each month. The *calendar days* in each *calendar month* are numbered sequentially starting with 1.

TSH	Tishrei	תשרי	30 days
CSH	Cheshvan	חשוון	29 days, or sometimes 30 days — see below
KSL	Kislev	כסלו	30 days, or sometimes 29 days — see below
TVT	Tevet	טבת	29 days
SHV	Shevat	שבט	30 days
ADR	Adar I	א אדר	30 days, if present — see below
ADS	Adar II	ב אדר	29 days
NSN	Nisan	ניסן	30 days
IYR	Iyar	אייר	29 days
SVN	Sivan	סיוון	30 days
TMZ	Tammuz	תמוז	29 days
AAV	Av	אב	30 days
ELL	Elul	אלול	29 days

> *Note* — The English names given above are transliterations of the Hebrew, and many variants of the transliterations can be found. Cheshvan is a shortened form of Marcheshvan (מרחשון) produced by dropping the first syllable. Adar I and Adar II are otherwise known as Adar Rishon (אדר ראשון) and Adar Sheni (אדר שני), respectively, using the Hebrew words for first and second. Adar I and II are also sometimes called Adar Aleph and Adar Bet (אדר א and אדר ב) after the first and second letters in the Hebrew alphabet.

> *Note* — For religious purposes, the *calendar year* is sometimes said to begin with the *calendar month* of Nisan. ELF uses the usual civil definition of the year beginning with the *calendar month* of Tishrei.

Calendar years containing 13 *calendar months* are called **leap years**, and fall in a 19 year cycle called the **Metonic cycle**. *Calendar years* which are not *leap years* are called **non-leap years**. To determine whether a year is a *leap year*, the *logical year* number is divided by 19 and the remainder taken to find the year's place in the *Metonic cycle*. If the remainder is 0, 3, 6, 8, 11, 14 or 17 then the year is a *leap year*; otherwise it is a *non-leap year*.

In a *non-leap year*, the *calendar month* of Adar I is omitted and Adar II is known simply as Adar (אדר). Nevertheless, the *month name* used for Adar in a *non-leap year* is "ADR", the same *month name* as Adar I in a *leap year*. "ADS" is not *well-formed* when used as a *month name* in a *non-leap year*, and it is RECOMMENDED that applications rewrite it to "ADR".

> *Note* — The intended handling of Adar in *non-leap years* is very poorly unspecified in [GEDCOM 5.5.1]. As Adar I is the extra month inserted into the *calendar*, it would make sense if Adar were represented as "ADS" in *non-leap years*, but in practice this seems not to be what current vendors do. Possibly because [GEDCOM 5.5.1] describes "ADR" as standing for Adar rather than Adar I or Adar Rishon, vendors have used "ADR" to represent Adar in *non-leap years* and Adar I in *leap years*. ELF standardises that behaviour.

The rule for deciding when Cheshvan is extended in length to have 30 *calendar days*, and when Kislev is reduced in length to have 29 *calendar days*, are somewhat complex and based on when in the week the Tishrei *molad* falls.

> *Note* — Although an application needs to know how many *calendar days* are in each *calendar month* in order to determine whether a *date* is a *well-formed date*, an applications can be *conformant* without being able to determine whether a *date* is *well-formed*. This would allow a *conformant* application to support the Hebrew *calendar* without full knowledge of how long each *calendar month* is. An application which cannot determine how many *calendar days* are in Cheshvan and Kislev in a particular year, MUST consider "30 CSH" and "30 KSL" to be *well-formed* regardless of year.

The **molad** is the calculated *instant* of the new moon, as seen from Jerusalem. The first *molad* occurred 5 hours and 204 *halakim* into Monday, 1 Tishrei AM 1. A **helek** (plural: **halakim**) is a traditional Hebrew subdivision of the hour equal to 3⅓ seconds: there are 1080 *halakim* in an hour. The week used by the Hebrew *calendar* is the standard seven day week.

> *Note* — This time is stated relative to the traditional Hebrew start of the day, which was at sunset the previous evening; it also uses unequal hours so that the time between sunset and sunrise is always 12 hours, regardless of the time of year. Applications do not need to be aware of this detail as all times are expressed using Hebrew time reckoning in this calculation. All they need to know is that there are 1080 *halakim* (also spelt "chalakim") in an hour, 24 hours in a *calendar day*, and 7 *calendar days* in a week.

In the Hebrew *calendar* calculations in this standard, *instants* within a week are expressed with as three integers separated with colons. These being the day number, the number of hours into the day, and the number of *halakim* into the hour. Days of the week are numbered with Saturday being 0.

> *Example* — The *instant* of the first *molad* can be written 2:5:204, meaning 5 hours and 204 *halakim* into a Monday.

The *molad* for any given *calendar month* and *calendar year* is calculated from an earlier *molad* whose time and day of the week are known by adding 4 weeks, 1 day, 12 hours, and 793 *halakim* per elapsed *calendar month* between *molads*.

> *Note* — This *duration* is commonly written 4:1:12:793, where the components are weeks, days, hours and *halakim*, though the number of weeks can be dropped for the purpose of

> determining the length of the *calendar year*. As 793 *halakim* does not equate to an integer number of seconds, doing the calculation in *halakim* rather than seconds avoids rounding errors if floating point arithmetic is used.

> *Note* — The computation of the *molad* is simplified by knowing that a complete 19 year *Metonic cycle* always contains exactly 235 *calendar months*.

> *Example* — The year AM 5779 began in September AD 2018, 5778 years after the first *molad*. 5778 divided by 19 is 304, with a remainder of 2, so from the first *molad* to the Tishrei *molad* for AM 5779, there were 304 complete *Metonic cycles* plus two extra years, which are both *non-leap years*, being years 1 and 2 in the *Metonic cycle*. This means 71 464 *calendar months* elapsed between the first *molad* and the Tishrei *molad* for AM 5779. Adding 71 464 lots of 4:1:12:793 to 2:5:204 gives 301482:2:14:316. The number of weeks can be discarded as it is irrelevant when determining the year length, leaving just 2:14:316. This says the Tushrei *molad* for the year AM 5779 happened on a Monday, being the second day after the Sabbath, at 08:17:33⅓, when converted to hours, minutes and seconds after midnight, rather than hours and *halakim* after sunset.

Once the Tishrei *molad* is known, the length of the *calendar year* can be determine using the following table. If the Tishrei *molad* is on or after the *instant* in the first column, and before the *instant* in the second column, then the length of the *calendar year* can be found in one of the last four columns, depending on the position of the year in the *Metonic cycle* as shown in the column headings.

Molad ≥	Molad <	1, 4, 9, 12 or 15	7 or 18	2, 5, 10, 13 or 16	0, 3, 6, 8, 11, 14 or 17
0:00:000	0:18:000	355	355	355	385
0:18:000	1:09:204	353	353	353	383
1:09:204	1:20:491	355	355	355	383
1:20:491	2:15:589	355	355	355	385
2:15:589	2:18:000	354	354	355	385
2:18:000	3:18:000	354	354	354	384
3:18:000	4:11:695	354	354	354	383
4:11:695	5:09:204	354	354	354	385
5:09:204	5:18:000	355	355	355	385
5:18:000	6:00:408	353	353	353	383
6:00:408	6:09:204	355	353	353	383
6:09:204	6:20:491	355	355	355	383
6:20:491	7:00:000	355	355	355	385

Cheshvan has 29 *calendar days* except when the *calendar year* is 355 or 385 *calendar days* long, in which case it has 30 *calendar days*. Kislev has 30 *calendar days* except when the *calendar year* is only 353 or 383 *calendar days* long, in which case it only has 29 *calendar days*.

> *Example* — The previous example calculated the Tishrei *molad* for AM 5779 to be at 2:14:316, which means the fourth row of the table applies. As 5779 divided by 19 is 304 with a remainder of 2, the year AM 5779 is the second year in the *Metonic cycle*, so the fifth column, whose heading includes "2", gives the year length as 355 *calendar days*. That means both Cheshvan and Kislev have 30 *calendar days*. "`30 CSH 5779`" and "`30 KSL 5779`" are therefore both *well-formed dates*.

> *Note* — This table is called the "Table of Four Gates". Its complexities are the result of the competing requirements that Yom Kippur does not fall on a day adjacent to the Sabbath (i.e. that 10 Tishrei not fall on a Friday or Sunday); that the new moon represented by the *molad* cannot be observed in daylight until it is six hours old and Tishrei cannot start until the new moon it deemed observable; that all months have 29 or 30 *calendar days*; and that only Cheshvan and Kislev can vary in length.

A *date* which uses a *calendar day* number which is greater than the number of *calendar days* in the specified year and month is not a *well-formed date*.

> *Editorial note* — The TSC decided to include a detailed specification of the Hebrew *calendar* in this standard rather than referencing a third party description largely because of the lack of a good, accessible description of *calendar* presented in a way that would make it easy to implement the *calendar* in ELF. Most sources describing the *calendar* do so in terms of the postponement rules which makes it easier to understand why the *calendar* is as it is, but does not make it straightforward to implement. It is hoped that by providing these details here, vendors will be encouraged to provide full implementations of the *calendar*.

5 The `elf:Time` datatype

ELF uses the `elf:Time` *datatype* to represent *times of day* using the 24-hour clock in *hours*, *minutes* and *seconds*, with these components separated by a colon (U+003A). The *hours* and *minutes* components are REQUIRED, and the *seconds* component SHOULD be provided. A fractional *seconds* component MAY be provided.

> *Example* — The value "`15:30:00`" is a valid *time of day*, represented using the `elf:Time` *datatype*. It represents half past three in the afternoon.

The `elf:Time` *datatype* is typically used in conjunction with a value of type `elf:DateExact` called its **associated date**.

> *Note* — The `elf:DateExact` *datatype* is limited to expressing *dates* in the Gregorian *calendar*, so the *associated date* will always be a Gregorian *date*.

> *Example —* In [ELF Data Model], *times of day* are found as `TIME` substructures of a `DATE` element which has a payload of *datatype* `elf:DateExact`.
>
> ```
> 2 DATE 10 DEC 2018
> 3 TIME 13:52:00
> ```
>
> In this example, "`10 DEC 2018`" is the *associated date* for the time "`13:52:00`".

The *lexical space* of the `elf:Time` *datatype* is the set of *strings* which match the following `Time` production, as well as the other constraints given here on the numerical value of each component. *Whitespace* is not permitted anywhere in an `elf:Time` value.

```
Time     ::=  HH ":" MM (":" SS)? TZD?

HH       ::= [0-9] [0-9]
MM       ::= [0-9] [0-9]
SS       ::= [0-9] [0-9] ("." [0-9]+)?
TZD      ::= "Z" | ("+" | "-") HH ":" MM
```

The `HH` production encodes the *hours* component of the *time of day*, zero-padded to two digits. It SHALL be a decimal integer between `00` and `24`, inclusive. Values outside this range are outside the *lexical space* of `elf:Time`. The *hours* component SHALL only be `24` if the *minutes* and *seconds* components are both zero or absent; any other uses of `24` as an *hours* component is outside the *lexical space* of the *datatype*.

> *Example —* The *strings* "`30:00`" and "`24:30`" are both outside the *lexical space* of this *datatype*. The former is invalid because the *hours* component of `30` is not between `00` to `24`; the latter is invalid because it has an *hours* component of `24` and a non-zero *minutes* component.

The *string* "`24:00:00`" is the **end-of-day instant** and denote the final *instant* of a *calendar day*. It is the same *instant* as the first *instant* of the following *calendar day* (which is denoted `00:00:00`). *Conformant* applications MUST accept *end-of-day instants* as valid input but MUST NOT create new instances of them. A *conformant* application MAY convert an *end-of-day instant* to `00:00:00`, but only if it has an *associated date* and that is simultaneously incremented by one day.

> *Example —* A *time of day* of "`24:00:00`" with an *associated date* of "`30 NOV 2018`" MAY be converted to a *time of day* of "`00:00:00`" with an *associated date* of "`1 DEC 2018`".

> *Note —* [GEDCOM 5.5.1] does not specify whether or not the *end-of-day instant* is legal, and existing applications are unlikely to produce it. It is supported by `elf:Time` for compatibility with [ISO 8601] and the `xsd:time` *datatype* defined in [XSD Pt2].

The `MM` production encodes the *minutes* component of the *time of day*, zero-padded to two digits. It SHALL be a decimal integer between `00` and `59`, inclusive. Values outside this range are outside the *lexical space* of `elf:Time`.

The SS production encodes the *seconds* component of the *time of day*, zero-padded to two digits and followed by an OPTIONAL fractional component. It SHALL be a decimal greater than or equal to 00 and strictly less than 61. Values outside this range are outside the *lexical space* of `elf:Time`.

Applications MUST preserve at least the first three decimal digits of a fractional *seconds* component, but MAY truncate or round the fractional part of the *seconds* component beyond that. Applications MAY add or remove trailing zeros on the frational part of the *seconds* component, and SHOULD add a *seconds* component of `:00` if none was given.

> *Note* — These provisions allows applications to process *times of day* using the standard data structures and facilities provided in many program languages, without preserving the original lexical form of the *time of day*.

> *Note* — A future version of this standard is likely to make the *seconds* component REQUIRED to make this *datatype* fully compatible with the `xsd:time` *datatype* defined in [XSD Pt2].

A *seconds* component greater than or equal to 60, and strictly less than 61 is called a **leap second** component. Any use of a *leap second* component is part of the *lexical space* of `elf:Time`, but applications SHALL only create new *leap second* to represent *leap seconds* inserted by the International Earth Rotation Service or its successor.

Conformant applications encountering a *leap seconds* component MAY convert it to an ordinary *seconds* component by subtracting one from its value. This SHOULD NOT be done if the application supports *leap seconds* and knows the specified *time of day* was a *leap second*.

> *Example* — The *string* "`12:56:60.800`" is in the *lexical space* of this *datatype* and has a *leap second* component of `60.800`. As *leap seconds* are always inserted at midnight UTC and there is no timezone in which this *time of day* is midnight UTC, this *time of day* could not arise in practice and therefore *conformant* applications MUST NOT generate such a *time of day*. *Conformant* applications MUST accept such *times of day* but MAY subtract one second to convert it to "`12:56:59.800`".

> *Note* — [GEDCOM 5.5.1] makes no mention of *leap seconds*, but existing applications are likely to generate such *times of day* if they happen to save a file during the inserted *leap second*. *Leap seconds* are also supported by [ISO 8601], but not in the `xsd:time` *datatype* defined in [XSD Pt2]. These rules allow compatibility with these standards and with existing use, while also allowing applications to ignore the *leap second*.

The *lexical space* of the `elf:Time` *datatype* allows the inclusion of a **time zone designator** matching the TZD production. *Conformant* applications MUST accept *time zone designators* on *dates* in input, but SHOULD ignore them and MAY remove them. *Conformant* applications MUST NOT include *time zone designators* on newly generated *times of day*.

> *Note* — Syntactic support for *time zone designators* is an extention to [GEDCOM 5.5.1], and is included in ELF for forwards compatibility. This standard does not specify the meaning conveyed by a *time zone designator*, but the syntax used is compatible with [ISO 8601] and the `xsd:time` *datatype* defined in [XSD Pt2], and a future version of this standard is likely to define it by reference to the latter standard. Application choosing not to ignore *time zone designators* are encouraged to follow [XSD Pt2].

Formally, the `elf:Time` *datatype* is a *structured non-language-tagged datatype* which has the following *properties*:

Datatype definition

Name	`https://terms.fhiso.org/elf/Time`
Type	`http://www.w3.org/2000/01/rdf-schema#Datatype`
Pattern	`(([01][0-9]|2[0-3]):[0-5][0-9](:([0-5][0-9]|60)(\.[0-9]+)?)?` `|24:00:00(\.0+)?)(Z|(\+|-)([01][0-9]|2[0-3]):[0-5][0-9])?`
Supertype	*No non-trivial supertypes*
Abstract	`false`

> *Note* — The *pattern* in the table above has been split on to two lines for convenience of presentation; it is, however, really one *pattern* and contains no *whitespace* or line breaks. really one single line. Any functional difference between the `Time` production and the *pattern* specified above is unintentional.

6 The `elf:Age` datatype

The **age** of a living individual is defined as the *duration* which has elapsed since the *instant* of their birth.

> *Note* — It is normal to stop counting *age* at the *instant* of an individual's death, therefore an *age* given for a posthumous event such as a burial is likely to be the *age* of the person when they died, i.e. the *duration* of the *time interval* from birth to death, rather than from birth to burial.

The `elf:Age` *datatype* is used to represent *ages* in ELF, which it does by recording number of *calendar years*, *calendar months* and *calendar days* to have elapsed during the *duration*.

The *lexical space* of this *datatype* is the set of *strings* which match the following `Age` production:

```
Age      ::= ( [<>] S? )? ( Duration | BareYear ) | AgeWord

Duration ::= [0-9]+ | [0-9]+ "y" ( S? [0-9]+ "m" )? ( S? [0-9]+ "d")?
             | [0-9]+ "m" ( S? [0-9]+ "d" )? | [0-9]+ "d"
BareYear ::= [0-9]+
AgeWord  ::= "CHILD" | "INFANT" | "STILLBORN"
```

> *Note* — These productions are case-sensitive, so the *string* "`17y`" matches the `Duration` production, while "`17Y`" does not.

> *Note* — Because the `elf:Age` *datatype* is currently only used in ELF in contexts where *whitespace normalisation* has been carried out, the `S` production will only ever match exactly one space *character*.

A *conformant* application serialising an *age* using this *datatype* SHOULD use a single space *character* (U+0020) wherever *whitespace* is permitted in the `Age` production.

> *Note* — [GEDCOM 5.5.1] under-specifies how *whitespace* is allowed in *ages*. The `Age` production is permissive in its treatment of *whitespace*, including allowing it to be omitted entirely. The preceding recommendation is intended to make *conformant* ELF applications maximally compliant with current GEDCOM implementations which typically use a single space *character*.

An *age* which uses the `elf:Age` *datatype* SHALL either consist of a quantitative *duration* conforming to the `Duration` or `BareYear` productions, or an qualitative *age* keyword conforming to the `AgeWord` production.

When an *age* is a quantitative *duration* matching the `Duration` production, it SHALL contain one, two or three integers, each followed by a "`y`", "`m`" or "`d`" suffix to denote a number of *calendar years*, *calendar months* or *calendar days*, respectively, the sum of which is the *age*.

> *Example* — The *age* "`15y 2m`" represents a *duration* of 15 years and 2 months.

If there is only a number of *calendar years* present, the `BareYear` production allows the "`y`" suffix to be omitted, though this is NOT RECOMMENDED.

> *Example* — The *age* "`42y`" only specifies a number of *calendar years* and so can be written "`42`" without the "`y`" suffix. The *age* "`15y 2m`" MUST NOT be written "`15 2m`" as this *duration* includes both a number of *calendar years* and *calendar months*.

> *Note* — Allowing the "`y`" suffix to be omitted as shown in the `Duration` production is a change from [GEDCOM 5.5.1] where it was REQUIRED. This has been relaxed because GEDCOM files fairly commonly contain lines like the following, despite them being illegal:
>
> ```
> 2 AGE 58
> ```

An *age* which is written including a (possibly zero) number of *calendar days* MAY be presumed to have a *precision* of a few *calendar days*; an *age* which includes a (possibly zero) number of *calendar months*, but no explicit number of *calendar days*, MAY be presumed to have a *precision* of a few *calendar months*; an *age* containing only a number of *calendar years* MAY be presumed to have a *precision* of a few *calendar years*.

> *Example* — The *age* "40y" MAY be assumed to have a *precision* of a few years. The actual *precision* will depend on context. In many situations the *precision* will be exactly one year, and so assuming the *age* has been given correctly, the individual was born at least 40 years ago but not as long as 41 years ago.
>
> At other times, a lower *precision* may apply. For example, on the 1841 census of Britain, adults were asked to round their *age* down to the previous multiple of five years. In such a context, an *age* of "40y" means the individual was born at least 40 years ago but not as long as 45 years ago. As ELF does not provide a means of stating a range of years in an *age*, nor of explicitly stating the *precision*, an *age* recorded as "40" on the 1841 census SHOULD be encoded in ELF as "40y".

The precise meaning of an *age* which is quantitative *duration* is dependent on context, cultural considerations and the *calendar* in use.

> *Example* — The previous example gave an example of how the interpretation of "40y" depended on context, due to different *precisions*.

> *Example* — An elderly person reckoning their age using the Islamic calendar could consider their age in years to be several years greater than if they were reckoning their age using the Gregorian calendar because the Islamic *calendar year* is around 11 *calendar days* shorter than the Gregorian *calendar year*. ELF does not provide a means of specifying which *calendar* was when recording an *age*.

> *Note* — A future version of ELF may add facilities to specify which *calendar* is being used for *ages* and also to allow an explicit *precision* to be stated, for example to say that an *age* has been given within a five-year *age* bracket.

Ages SHOULD generally be rounded down when expressed using just a number of *calendar years*, or using a number *calendar years* and *calendar months*, but an application SHOULD NOT assume this is necessarily the case without additional information or context.

> *Example* — In many parts of East Asia, *ages* are rounded up rather than down, so a newborn infant is considered to be one year old, and becomes two years old on the first anniversary of their birth.

A *conformant* application MAY remove any leading zeros preceding a non-zero digit, change how *whitespace* is used in an *age*, or append any omitted "y" suffix, but MUST NOT otherwise alter the *age*. In particular, applications MUST NOT insert or remove zero components, except as allowed below, nor convert between days, months and years.

> *Example* — "2y 0m", "2y" and "24m" are all distinct *ages* and applications MUST NOT rewrite one to another. However a *conformant* application MAY convert "2" to "2y", adding the "y"

> suffix; MAY convert "`02y`" to "`2y`", removing leading zero; and MAY convert "`2y0m`" to "`2y 0m`", a simple change in *whitespace*.

In addition, a *conformant* application MAY insert or remove components which are the number zero followed by "`y`", "`m`" or "`d`" suffix, providing doing so does not alter the presumed *precision* of the *age*.

> *Example* — The *ages* "`2y 0m`" and "`2y`" are presumed to have different *precisions*: the former has a *precision* of a few months, while the latter has a *precision* of a few years. Converting one to the other by inserting or removing a "`0m`" component would change the *precision* of the *age*, and of this reason a *conformant* application MUST NOT do it.
>
> However it is only the least significant component present (representing the smallest unit of duration) which affects the *precision*, and other zero components MAY be inserted or removed. For example, "`2m`" and "`0y 2m`" are equivalent in ELF, and applications MAY convert either one to the other.

Ages which are quantitative *durations* MAY be prefixed with with a "<" or a ">". These are interpreted as meaning the individual is at most the specified *age*, or is at least the specified *age*, respectively.

> *Note* — This means the "<" and ">" token are actually interpreted as ≤ and ≥ operators, though in practice the *precision* of *ages* in ELF is such that difference between ≤ and <, or between ≥ and > does not matter.
>
> In common usage, a person may be said to over 21 as soon as they've had their 21st birthday, at which point they would normally round their age down to 21. ELF allows for this usage, and "`> 21y`" may be used to refer to someone whose *age* is exactly 21. This may seem to be a deviation from [GEDCOM 5.5.1] which defines them as meaning less than or greater than the specified *age*, respectively, but because of the uncertainty in the intended *precision* of an *age*, there is little practical difference. The case for interpreting "<" as less than or equal to is weaker, but there are examples in current GEDCOM files where "<" has been used on *ages* inferred from an *age* given a few months later.

> *Example* — If a source gives an individual's *age* as 52 in March, their *age* might be inferred to be "`< 52y`" the preceding January. In practice, they might be 51 or 52, assuming the source is *accurate*. However, it is normally best not to infer *ages* and only to record *ages* when they are given in sources.

Instead of a quantitative *duration*, an *age* MAY be expressed qualitatively using an **age word** which matches the `AgeWord` production. This standard defines three *age words*: `CHILD`, `INFANT` and `STILLBORN`. These SHOULD be used when a source describes an individual as a child, an infant, or stillborn, respectively, or using other words which are broadly equivalent. They SHOULD NOT be used when a source describes an individual using a quantitative *age*.

> *Note* — The words “child”, “infant” and “stillborn” can be considered societal roles which are to some extent culturally dependent, rather than well-defined *age* brackets. A modern source might describe a 15-year-old as a child, while in mediæval times, a person of this age would be working and unlikely to be described as a child. This standard does not therefore put firm limits on the *ages* meant by these terms. This is a deviation from [GEDCOM 5.5.1] which states that a child is less than 8 years old, an infant is less than 1 year old, and a stillborn individual is approximately 0 days old. Nevertheless, these ages may be useful in deciding whether a foreign word or phrase can reasonably be translated as one of these word.

The *age word* `STILLBORN` is not only a statement of *age* but also conveys the fact that the individual died just prior to, at, or immediately after the time of birth. This *age word* MUST NOT be used to refer to infants whose *age* is about 0 days unless they died around the time of birth.

> *Editorial note* — Having exactly three *age words* seems unsatisfactory as it fails to cope with other descriptions that might occur in a source, such as “elderly”. The TSC have discussed whether *age words* should be deprecated, or whether they should be developed into a more general and extensible *age word* mechanism, but have reached no conclusion so far.

Formally, the `elf:Age` *datatype* is a *structured non-language-tagged datatype* which has the following *properties*:

Datatype definition

Name	`https://terms.fhiso.org/elf/Age`
Type	`http://www.w3.org/2000/01/rdf-schema#Datatype`

Pattern	`([<>][ \t\r\n]*)?([0-9]+|[0-9]+y([ \t\r\n]*[0-9]+m)?`
	`([ \t\r\n]*[0-9]+d)?|[0-9]+m([ \t\r\n]*[0-9]+d)?|[0-9]+d)`
	`|CHILD|INFANT|STILLBORN`

Supertype	*No non-trivial supertypes*
Abstract	`false`

> *Note* — The *pattern* in the table above has been split on to three lines for convenience of presentation; it is, however, really one *pattern* and contains no *whitespace* or line breaks. Any functional difference between the `Age` production and the *pattern* specified above is unintentional.

> *Editorial note* — Depending on the details of how the ELF data model is specified, it may make sense to split this into two *datatypes*: the `elf:Age` *datatype* which would be restricted to just quantitative *ages* and would therefore serve as a general *duration* datatype, and a separate `elf:AgeWord` *datatype*. The various ELF structures having an `elf:Age` *payload* would then be changed to have *payloads* which were a *union of datatypes*. This introduces several technical complications, including the need make *datatype correction* work with

> multiple *default datatypes*. In principle this works fine when the *default datatypes* have disjoint *lexical spaces*, as they would here, but it would need careful specification.

7 References

7.1 Normative references

[Basic Concepts]
FHISO (Family History Information Standards Organisation). *Basic Concepts for Genealogical Standards*. First public draft. (See https://fhiso.org/TR/basic-concepts.)

[RFC 2119]
IETF (Internet Engineering Task Force). *RFC 2119: Key words for use in RFCs to Indicate Requirement Levels.* Scott Bradner, 1997. (See http://tools.ietf.org/html/rfc2119.)

[XML]
W3C (World Wide Web Consortium). *Extensible Markup Language (XML) 1.1*, 2nd edition. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, and John Cowan eds., 2006. W3C Recommendation. (See https://www.w3.org/TR/xml11/.)

7.2 Other references

[ELF Data Model]
FHISO (Family History Information Standards Organisation). *Extended Legacy Format (ELF): Data Model*. Exploratory draft.

[ELF Serialisation]
FHISO (Family History Information Standards Organisation). *Extended Legacy Format (ELF): Serialisation Format*. Exploratory draft.

[GEDCOM 5.3]
The Church of Jesus Christ of Latter-day Saints. *The GEDCOM Standard*, draft release 5.3. 4 Nov 1993.

[GEDCOM 5.5]
The Church of Jesus Christ of Latter-day Saints. *The GEDCOM Standard*, release 5.5. 2 Jan 1996, as amended by the errata sheet dated 10 Jan 1996.

[GEDCOM 5.5.1]
The Church of Jesus Christ of Latter-day Saints. *The GEDCOM Standard*, draft release 5.5.1. 2 Oct 1999.

[GEDCOM X Dates]
Intellectual Reserve Inc. *The GEDCOM X Date Format*. Stable draft, accessed December 2018. See http://gedcomx.org/.

[ISO 8601]

ISO (International Organization for Standardization). *ISO 8601:2004. Data elements and interchange formats — Information interchange — Representation of dates and times*. 2004.

[ISO 8601-2]

ISO (International Organization for Standardization). *ISO 8601-2:2009. Data elements and interchange formats — Information interchange — Part 2: Extensions*. Draft, 15 Feb 2016.

[FHISO Patterns]

FHISO (Family History Information Standards Organisation). *The Pattern Datatype*. First public draft. (See https://fhiso.org/TR/patterns.)

[XSD Pt2]

W3C (World Wide Web Consortium). *W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*. David Peterson, Shudi Gao (高 殊 镝), Ashok Malhotra, C. M. Sperberg-McQueen and Henry S. Thompson, ed., 2012. W3C Recommendation. (See https://www.w3.org/TR/xmlschema11-2/.)

[Triples Discovery]

FHISO (Family History Information Standards Organisation). *Simple Triples Discovery Mechanism*. First public draft. (See https://fhiso.org/TR/triples-discovery.)