# The FamilySearch GEDCOM Specification

7.0.17

Prepared by the

Family History Department
The Church of Jesus Christ of Latter-day Saints

10 February 2026

**Suggestions and Correspondence:**

**Family History Department**
15 East South Temple Street
Salt Lake City, UT 84150 USA

GEDCOM@FamilySearch.org

Copyright 1984–2026 Intellectual Reserve, Inc. All rights reserved. A service provided by The Church of Jesus Christ of Latter-day Saints.

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at

> http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

In accordance with the Apache 2.0 license that governs this work, any other work that is based on or derived from this work must include a readable copy of the following NOTICE. For more information, please refer to the full copy of the Apache 2.0 license.

> NOTICE:
>
> This work comprises, is based on, or is derived from the FAMILYSEARCH GEDCOM™ Specification, © 1984-2026 Intellectual Reserve, Inc. All rights reserved.
>
> “FAMILYSEARCH GEDCOM™” and “FAMILYSEARCH®” are trademarks of Intellectual Reserve, Inc. and may not be used except as allowed by the Apache 2.0 license that governs this work or as expressly authorized in writing and in advance by Intellectual Reserve, Inc.

# Contents


- Introduction 6
  - Purpose and Content of *The FamilySearch GEDCOM Specification* 6
  - Purpose for Version 7.x 7
  - A Guide to Version Numbers 8
  - URIs and Prefix Notation 8
- 1 Hierarchical container format 10
  - 1.1 Characters 10
  - 1.2 Structures 11
  - 1.3 Lines 13
  - 1.4 The Header and Trailer 17
  - 1.5 Extensions 18
    - 1.5.1 Extension Tags 21
    - 1.5.2 Requirements and Recommendations 24
    - 1.5.3 Extension versus Standard 24
  - 1.6 Removing data 25
- 2 Data types 27
  - 2.1 Text 27
  - 2.2 Integer 27
  - 2.3 Enumeration 27
  - 2.4 Date 28
  - 2.5 Time 31
  - 2.6 Age 31
  - 2.7 List 32
  - 2.8 Personal Name 33
  - 2.9 Language 34
  - 2.10 Media Type 34
  - 2.11 Special 34
  - 2.12 File Path 35
  - 2.13 URI 36
  - 2.14 Tag Definition 36
  - 2.15 Latitude 36
  - 2.16 Longitude 37
- 3 Genealogical structures 38
  - 3.1 A Metasyntax for Structure Organization 38
  - 3.2 Structure Organization 40
    - 3.2.1 Document 40
    - 3.2.2 Records 43
    - 3.2.3 Substructures 50

- 3.3 Structure Meaning 66
  - 3.3.1 Events 66
  - 3.3.2 Attributes 70
  - 3.3.3 Latter-day Saint Ordinances 72
  - 3.3.4 Structure types 73
- 3.4 Enumeration Values 102
  - `g7:enumset-ADOP` 102
  - `g7:enumset-EVEN` 102
  - `g7:enumset-EVENATTR` 103
  - `g7:enumset-MEDI` 103
  - `g7:enumset-PEDI` 104
  - `g7:enumset-QUAY` 104
  - `g7:enumset-RESN` 105
  - `g7:enumset-ROLE` 105
  - `g7:enumset-SEX` 106
  - `g7:enumset-FAMC-STAT` 107
  - `g7:enumset-ord-STAT` 107
  - `g7:enumset-NAME-TYPE` 109
- 4 The FamilySearch GEDZIP file format 110
- 5 Contributors 112
- 6 Appendix A: Known Calendars and Dates 114
  - 6.1 Known Calendars 114
    - `GREGORIAN` 114
    - `JULIAN` 115
    - `FRENCH_R` 115
    - `HEBREW` 116
  - 6.2 Dual dates 117
  - 6.3 Calendars in date ranges and date periods 118

# Introduction

FamilySearch GEDCOM 7.0 was released in 2021 as the latest version of the GEDCOM format for the transmission and storage of genealogical information. GEDCOM was developed by the Family History Department of The Church of Jesus Christ of Latter-day Saints to provide a flexible, uniform format for exchanging computerized genealogical data. Its first purpose is to foster sharing genealogical information and to develop a wide range of inter-operable software products to assist genealogists, historians, and other researchers. Its second purpose is as a long-term storage format for preserving genealogical information in an open, standard format that will be accessible and understood by future genealogists and the systems they use.

"GEDCOM" is an acronym for **GE**nealogical **D**ata **COM**munication and is traditionally pronounced "ˈdʒɛdkɑm."

# Purpose and Content of *The FamilySearch GEDCOM Specification*

*The FamilySearch GEDCOM Specification* is a technical document written for computer programmers, system developers, and technology-aware users. This document describes a document and file format as follows:

- A hierarchical container format (see Chapter 1 (p.10))
- A set of data types (see Chapter 2 (p.27))
- A set of genealogical structures (see Chapter 3 (p.38))
- The FamilySearch GEDZIP file format (see Chapter 4 (p.110))

Chapter 1 describes a hierarchical container format. This container format is a general-purpose data representation language for representing any kind of structured information in a sequential medium, similar to XML, JSON, YAML, or SDLang. Chapter 1 discusses the syntax and identification of structured information in general, but it does not deal with the semantic content of any particular kind of data.

Chapter 2 describes several data types used to represent genealogical information, such as a date format that permits dating in multiple calendar systems.

Chapter 3 describes a set of nested genealogical structures for representing historical claims, such as individuals, families, and events; sourcing information, such as sources, repositories, and citations; and research metadata, such as information about researchers and rights.

A set of structures conforming to the first 3 chapters of *The FamilySearch GEDCOM Specification* is called a FamilySearch GEDCOM dataset. A string of octets encoding a dataset is called a data stream.

Chapter 4 describes a file format for bundling a dataset with a set of media files or other supporting documents.

> *Note* — Prior to 7.0:
>
> - The container format was called “the GEDCOM data format.”
> - The data types were unnamed and described in various places throughout the document.
> - The genealogical structures were known as “the Lineage-Linked GEDCOM Form.”

In addition to this specification itself, additional supplimental material can be found at https://gedcom.io, including:

- Links to example and supporting software
- Frequently asked questions for users and developers
- A guide to migrating from 5.5.1 to 7.0
- Ways to engage in the community

# Purpose for Version 7.x

There have been multiple prior releases of this specification, with somewhat idiosyncratic version numbering. The first public comment draft was released in 1984. The previous major version was 5.5.1 which was released in draft status in November 1999 and re-released as a standard in October 2019.

Version 7.0 has a number of goals, including

- Clarify ambiguities in the specification.
- Simplify implementations by removing special-case handling.
- Modernize character encoding, length restrictions, and specification wording.
- Introduce semantic versioning (see https://semver.org/).
- Add better multimedia handling, negative assertions, and notes with markup.
- Add support for common extensions to 5.5.1.
- Provide tools for better interoperability of extensions.

Version 7.0 introduces several breaking changes with version 5.5.1; 5.5.1 files are, in general, not valid 7.0 files and *vice versa*. These breaking changes were necessary to remove complicated constructs left over from earlier versions. For a complete list of changes, see the accompanying changelog.

# A Guide to Version Numbers

Starting with version 7.0.0, version numbers use semantic versioning. The 3 numbers are titled *major.minor.patch*.

A new *major* version may make arbitrary changes to the specification. Distinct major versions are not in general either forward or backward compatible with one another.

A new *minor* version will preserve the validity of data from all previous minor versions. It may make additional data valid, for example by adding new structure types, allowing current structures in new contexts, or adding new enumerated values or calendars. A minor release will not change the semantic meaning of data from previous minor releases, so for example a 7.0 document is also a valid 7.1 document and represents the same information in both.

A new *patch* version is a clarified or improved specification for the same data and introduces no changes in the data itself. Any software that correctly implements *X.Y.Z* also correctly implements *X.Y.W*. If there is an ambiguity or contradiction in the specification, it will be resolved in a patch version unless it is known that implementations interpreted the spec differently and that clarifying the intended meaning would cause incompatibilities between those implementations.

It is recommended that implementations accept all data at their own or a lesser minor version, regardless of the patch version. It is also recommended that they import data from subsequent minor versions by treating any unexpected structures, enumerations, or calendars as if they were extensions (p.18).

# URIs and Prefix Notation

This document defines Uniform Resource Identifiers (URIs) to unambiguously identify various concepts, including structure types, data types, calendars, enumerated values, and so on. In a few places, existing URIs defined by other bodies are used, following the best practice that a new URI should not be introduced for a concept for which a URI is known.

Rather than write out URIs in full, we use prefix notation: any URI beginning with 1 of the following short prefixes followed by a colon is shorthand for a URI beginning with the corresponding URI prefix

| Short Prefix | URI Prefix |
|---|---|
| `g7` | `https://gedcom.io/terms/v7/` |
| `xsd` | `http://www.w3.org/2001/XMLSchema#` |
| `dcat` | `http://www.w3.org/ns/dcat#` |

> *Example* — When the specification says `xsd:string`, it means `http://www.w3.org/2001/XMLSchema#string`. This is a specification shorthand only; the string "`xsd:string`" is not the URI defining this concept, `http://www.w3.org/2001/XMLSchema#string` is.

# 1. Hierarchical container format

## 1.1. Characters

Each data stream is a sequence of octets or bytes. The octets encode a sequence of characters according to the UTF-8 character encoding as described in §10.2 of ISO/IEC 10646:2020 and in RFC 3629.

> *Note* — Previous versions allowed multiple character encodings, defaulting to ANSEL. 7.0 only uses the UTF-8 character encoding.

A file containing a FamilySearch GEDCOM data stream should use the filename extension `.ged`.

The first character in each data stream should be U+FEFF, the byte-order mark. If present, this initial character has no meaning within this specification but serves to indicate to other systems that the file uses the UTF-8 character encoding.

Certain characters must not appear anywhere within a data stream:

- The C0 control characters other than tab and line endings (U+0000–U+001F except U+0009, U+000A and U+000D)
- The DEL character (U+007F)
- Surrogates (U+D800–U+DFFF)
- Invalid code points (U+FFFE and U+FFFF)

Implementations should be aware that bytes per character and characters per glyph are both variable when using UTF-8. Use of Unicode-aware processing and display libraries is recommended.

Character-level grammars are specified in this document using Augmented Backus-Naur Form (ABNF) as defined in STD 68 and modified in RFC 7405. We use the term “production” to refer to an ABNF rule, supported by any other rules it references.

> *Note* — The following is a brief summary of the parts of ABNF, as defined by STD 68 and RFC 7405, that are used in this document:
>
> - A rule consists of a rulename, an equals sign `=`, and 1 or more alternative matches.
> - Alternatives are separated by slashes `/`.
> - The first line of a rule must not be indented; the second and subsequent lines of a rule must be indented.
> - Comments are introduced with a semi-colon `;`.
> - Unicode codepoints are given in hexadecimal preceded by `%x`. Ranges of allowed codepoints are given with a hyphen `-`.
> - Double quotes delimit literal strings. Literal strings are case-insensitive unless they are preceded by `%s`.
> - Parentheses `()` group elements. Brackets `[]` mark optional content. Preceding a group or element by `*` means any number may be included. Preceding a group or element by `1*` means 1 or more may be included.

The banned characters can be expressed in ABNF as production `banned` (p.10):

```
        banned = %x00-08 / %x0B-0C / %x0E-1F ; C0 other than LF CR and Tab
               / %x7F                        ; DEL
               / %x80-9F                     ; C1
               / %xD800-DFFF                 ; Surrogates
               / %xFFFE-FFFF                 ; invalid
        ; All other rules assume the absence of any banned characters
```

All other ABNF expressions in this document assume the absence of any characters matching production `banned`.

This document additionally makes use of the following named character sets in ABNF:

```
        ; DIGIT     = %x30-39   ; 0 through 9 -- defined in RFC 5234 section B.1
        nonzero     = %x31-39   ; 1 through 9
        ucletter    = %x41-5A   ; A through Z
        underscore  = %x5F      ; _
        atsign      = %x40      ; @
```

## 1.2. Structures

A **structure** consists of a structure type, an optional **payload**, and a collection of sub-structures. The payload is a value expressed as a string using 1 of several data types, as described in Chapter 2 (p.27).

Every structure is either a **record**, meaning it is not contained in any other structure's collection of substructures, or it is a **substructure** of exactly 1 other structure. The other structure is called its **superstructure**. Each substructure either refines the meaning of its superstructure, provides metadata about its superstructure, or introduces new data that is closely related to its superstructure.

Each **structure type** is identified by a URI and defines several properties of any structure with that type, including

- The meaning of structures of this type.
- The payload type of the structure's payload, which shall be one of
  - no payload, or
  - a pointer to a record with a specific structure type, or
  - a data type (p.27); if an enumeration (p.27) or list of enumerations (p.32), also a set of permitted enumeration values.
- Which structure types may appear as substructures of the structure and with what **cardinality** they may appear. Cardinality is specified by two flags:
  - whether a substructure of this type is required or not; and
  - whether multiple substructures of this type are permitted or not.

The collection of substructures is partially ordered. Substructures with the same structure type are in a fixed order, but substructures with different structure types may be reordered. The order of substructures of a single type indicates user preference, with the first substructure being the most-preferred value, unless a different meaning is explicitly indicated in the structure's definition.

A structure must have either a non-empty payload or at least 1 substructure. Empty payloads and missing payloads are considered equivalent. The remainder of this document uses "payload" as shorthand for "non-empty payload".

> *Note* — Unlike structures, pseudo-structures needn't have either payloads or substructures. `TRLR` (p.99) never has either, and `CONT` (p.76) doesn't when payloads contain empty lines.

A structure is a representation of data about its **subject**. Examples include the entity, event, claim, or activity that the structure describes.

Datasets also contain 3 types of pseudo-structures:

- The header resembles a record, comes first in each document, and contains metadata about the entire document in its substructures. See The Header (p.17) for more details.
- The trailer resembles a record, comes last in each document, and cannot contain substructures.

- A line continuation resembles a substructure, comes before any other substructures, is used to encode multi-line payloads, and cannot contain substructures.

Previous versions limited the number of characters that could appear in a structure, record, and payload. Those restrictions were removed in 7.0.

## 1.3. Lines

A **line** is a string representation of (part of) a *structure*. A line consists of a level, optional cross-reference identifier, tag, optional line value, and line terminator. It matches the production `Line`:

```
Line    = Level D [Xref D] Tag [D LineVal] EOL

Level   = "0" / nonzero *DIGIT
D       = %x20                              ; space
Xref    = atsign 1*tagchar atsign           ; but not "@VOID@"
Tag     = stdTag / extTag
LineVal = pointer / lineStr
EOL     = %x0D [%x0A] / %x0A                ; CR-LF, CR, or LF

stdTag  = ucletter *tagchar
extTag  = underscore 1*tagchar
tagchar = ucletter / DIGIT / underscore

pointer = voidPtr / Xref
voidPtr = %s"@VOID@"

nonAt   = %x09 / %x20-3F / %x41-10FFFF      ; non-EOL, non-@
nonEOL  = %x09 / %x20-10FFFF                ; non-EOL
lineStr = (nonAt / atsign atsign) *nonEOL ; leading @ doubled
```

The **level** matches production `Level` (p.13) and is used to encode substructure relationships. Any line with level 0 encodes a record or a record-like pseudo-structure. Any line with level $x > 0$ encodes a substructure of the structure encoded by the nearest preceding line with level $x - 1$.

> *Note* — Previous versions allowed spaces and blank lines to precede the level of a line. That permission was removed from 7.0 to simplify parsing.

The **cross-reference identifier** matches production `Xref` (but not `voidPtr`) and indicates that this is a structure to which pointer-type payloads may point. Each cross-reference identifier must be unique within a given data document. Cross-reference identifiers are not retained between data streams and should not be made visible to the user to avoid them referring to transient data within notes or other durable data.

Each record to which other structures point must have a cross-reference identifier. A record to which no structures point may have a cross-reference identifier, but does not need to have one. A substructure or pseudo-structure must not have a cross-reference identifier.

The **tag** matches production `Tag` and encodes the structure's type. Tags that match the production `stdTag` are defined in this document. Tags that match `extTag` are defined according to Extensions (p.18).

The same tag may be used to represent multiple structure types. The structure type of each structure is identified by its tag and the type of its superstructure. The mapping between (superstructure type, tag) pairs and structure types is given elsewhere in this document (for standard structure types and tags) or the [schema] and extension authors' documentation (for extension structure types and tags).

> *Example* — The tag `ADOP` is used in this document to represent two structure types. Which one is meant can be identified by the superstructure type as follows:
>
> | Superstructure type | Structure type identified by tag `ADOP` |
> |---|---|
> | `g7:record-INDI` (p. 84) | `g7:ADOP` |
> | `g7:ADOP-FAMC` (p.82) | `g7:FAMC-ADOP` |
>
> An extension-defined substructure (p.18) could also be used to place either of these structure types in extension superstructures.
>
> The `ADOP` tag is also used in the set of enumerated values permitted by the `g7:DATA-EVEN` (p.80), `g7:SOUR-EVEN` (p.80), and `g7:NO` (p.89) structure types.

The **line value** matches production `LineVal` (p.13) and encodes the structure's payload. Line value content is sufficient to distinguish between pointers and line strings. Pointers are encoded as the cross-reference identifier of the pointed-to structure. Each non-pointer payload may be encoded in 1 or more line strings (line continuations encode multi-line payloads in several line strings). The exact encoding of non-pointer payloads is dependent on the data type of the payload, as determined by the structure type. The data type of non-pointer payloads cannot be fully determined by line value content alone.

Note that production `LineVal` does not match the empty string. Because empty payloads and missing payloads are considered equivalent, both a structure with no payload and a structure with the empty string as its payload are encoded with no `LineVal` and no space after the `Tag`.

> *Example* — The payload of a `MARR` structure has type `[Y|<NULL>]`, which is optional but if present cannot be the empty string. The payload of a `EVEN` structure has type `Text` (p.27), which is not optional but can be the empty string. The `Line` encoding a no-payload `MARR` (p.86) is "`1 MARR`" and the `Line` encoding an empty-payload `EVEN` is "`1 EVEN`"; both `Line`s have no `LineVal` and no trailing space.

If a line value matches production `Xref`, the same value must occur as the cross-reference identifier of a structure within the document. The special `voidPtr` production is provided to encode null pointers.

If the first character of the string stored in a line string is U+0040 (`@`), the line string must escape that character by doubling that `@`.

> *Note* — Previous versions required doubling all `@` in a line value, but such doubling was not widely implemented in practice. `@` is only doubled in this version if it is the first character of a line string.

> *Example* — A structure with tag `NOTE` (p.89), level 1, and a 2-line payload where the first line is "`me@example.com is my email`" and the second line is "`@me and @I are my social media handles`" would be encoded as
>
> ```
> 1 NOTE me@example.com is my email
> 2 CONT @@me and @I are my social media handles
> ```

> *Note* — Line values that match neither `Xref` nor `lineStr` are prohibited. They have been used in previous versions (for example, a line value beginning `@#D` was a date in versions 4.0 through 5.5.1) and may be used again in a future version if an appropriate need arises.

The components of a line are each separated by a single **delimiter** matching production `D`. A delimiter is always a single space character (U+0020). Using multiple delimiters between components of a line is prohibited. Thus if the tag is followed by 2 spaces, the first space is a delimiter and the second space is part of the line value.

All characters in a payload must be preserved in the corresponding line value, including preserving any leading or trailing spaces.

Each line is ended by a **line terminator** matching production `EOL`. A line terminator may be a carriage return U+000D, line feed U+000A, or a carriage return followed by a line feed. The same line terminator should be used on every line of a given document.

Line values cannot contain internal line terminators, but some payloads can. If a payload contains a line terminator, the payload is split on the line terminators into several payloads. The first of these split payloads is encoded as the line value of the structure's line, and each subsequent split payload is encoded as the line value of a **line continuation** pseudo-structure placed immediately following, and with one greater level than, the structure's line. The tag of a line continuation pseudo-structure is `CONT` (p.76). The order of the line continuation pseudo-structures matches the order of the lines of text in the payload.

> *Note* — Versions prior to 7.0 had another `CONT`-like tag, `CONC`, which split line values without introducing a line break. `CONC` does not appear in version 7. To support multi-version GEDCOM parsers, the `CONC` tag is reserved and will not appear as the tag of a structure type.

Line continuation pseudo-structures are not considered to be structures. While they match production `Line` and their level and position makes them appear to be substructures of the structure, they are actually a continuation of the encoding of the structure's payload and are not part of a structure's collection of substructures. They must appear immediately following the line whose payload they are encoding and before any other line.

Because line terminators in payloads are encoded using line continuations, it is not possible to distinguish between U+000D and U+000A in payloads.

> *Note* — Previous versions limited the number of characters that could appear in a tag, cross-reference identifier, and line-value. Those restrictions were removed in version 7.0. The `CONC` pseudo-structure, which allowed line values to have a shorter length restriction than payloads, was also removed.

*Example* — The following are examples of valid but unrelated lines:

- level 0, cross-reference identifier `@I1234@`, tag `INDI`, no line value.

```
0 @I1234@ INDI
```

- level 1, no cross-reference identifier, tag `CHIL` (p.76), pointer line value pointing to the structure with cross-reference identifier "`@I1234@`".

```
1 CHIL @I1234@
```

- level 1, no cross-reference identifier, tag `NOTE` (p.89), and line value + continuation pseudo-structure to encode a 4-line payload string: "`This is a note field that`", "`  spans four lines.`", "", and "`(the third line was blank)`". Note that leading and trailing spaces are preserved.

```
1 NOTE This is a note field that
2 CONT   spans four lines.
2 CONT
2 CONT (the third line was blank)
```

## 1.4. The Header and Trailer

Every dataset must begin with a header pseudo-structure and end with a trailer pseudo-structure.

The trailer pseudo-structure has level `0`, tag `TRLR` (p.99) and no line value or substructures. The trailer has no semantic meaning; it is present only to mark the end of the dataset.

The header pseudo-structure has level `0`, tag `HEAD` (p.83), and no line value. The substructures of the header pseudo-structure provide metadata about the entire dataset. Some of those substructures are defined here; others are defined in Chapter 3 (p.38) or by extensions.

Every header must contain a substructure with a known tag that identifies the specification to which the dataset complies. For FamilySearch GEDCOM 7.0, this is the `GEDC` (p.83) structure described in Chapter 3 (p.83).

A header should contain an extension schema structure with tag `SCHMA` (p.94) as described in Extensions (p.18).

## 1.5. Extensions

A **standard structure** is a structure whose type, tag, meaning, superstructure, and cardinality within the superstructure are described in this document. This includes records such as `INDI` and substructures such as `INDI.NAME` (p.88).

The recommended way to go beyond the set of standard structure types in this specification or to expand their usage is to submit a feature request on the FamilySearch GEDCOM development page so that the ramifications of the proposed addition and its interplay with other proposals may be discussed and the addition may be included in a subsequent version of this specification.

This specification also provides multiple ways for extension authors to go beyond the specification without submitting a feature request, which are described in the remainder of this section.

Extensions can introduce new structure types, new enumeration values, new calendars with their associated months, and new data types. They can also extend existing structures with new permitted substructure types and extend existing enumeration-type payloads with new permitted values. Extensions cannot change existing meanings, cardinalities, or calendars.

A **tagged extension structure** is a structure whose tag matches production `extTag` (p.13). Tagged extension structures may appear as records or substructures of any other structure. Their meaning is defined by their tag, as is discussed more fully in the section Extension Tags (p.21).

Any substructure of a tagged extension structure that uses a tag matching `stdTag` is an **extension-defined substructure**. Substructures of an extension-defined substructure that uses a tag matching `stdTag` are also extension-defined substructures. The meaning and use of each extension-defined substructure is defined by the tagged extension structure it occurs within, not by its tag alone nor by this specification.

*Example* — In the following

```
0 @P1@ _LOC
1 NAME Βυζάντιον
2 DATE FROM 667 BCE TO 324
1 _POP 15149358
2 DATE 31 DEC 2020
0 @I1@ INDI
1 BIRT
2 _LOC @P1@
```

- Both uses of `_LOC` are tagged extension structures, as is `_POP`.
- `_LOC.NAME` and `_LOC.NAME.DATE` are both extension-defined substructures. Their meaning is defined by the specification defining `_LOC`, but since no standard definition of `NAME` (p.88) permits `DATE` (p.78) as a substructure, such use is deprecated.
- `_POP.DATE` is an extension-defined substructure. Its meaning is defined by the specification defining `_POP`.
- Even though both `DATE`s appear to have `g7:type-DATE` payloads, we can't know that is the intended data type without consulting the defining specifications of `_LOC` and `_POP`, respectively. The first might be a `g7:type-DATE#period` and the second a `g7:type-DATE#exact`, for example.

Extension-defined substructures should match the structure type, payload, and substructure collection of at least one standard type with the same tag, though it can add more substructures to the substructure collection. This specification deprecates using a `stdTag` (p.13) with a definition that does not match any standard type with that tag.

*Example* — An extension-defined substructure with tag "`DATE`" should provide a date or date period relevant to its superstructure, as do all `DATE`-tagged structures in this specification. Extensions should not use "`DATE`" to tag a structure describing anything else (even something that might reasonably be abbreviated "date", such as someone an individual dated).

As a special case, a tagged extension structure can be defined to have a standard structure type. These are called **relocated standard structures** and can only appear with superstructures that are not documented as a superstructure of that structure type in this specification. The extension-defined substructures of a relocated standard structure are the substructure types documented in this specification for that structure type, including usual limitations on cardinality, payloads, substructures, etc.

> *Example* — Suppose `_DATE` is defined to mean a `g7:DATE` (using a documented extension tag (p.21)). Then in the following
>
> ```
> 0 @I1@ INDI
> 1 NAME John /Doe/
> 2 _DATE FROM 6 APR 1917 TO 11 NOV 1918
> 3 PHRASE During America's involvement in the Great War
> 1 BIRT
> 2 PLAC Queens, New York, New York, USA
> ```
>
> - `_DATE` is a relocated standard structure with type `g7:DATE`, with the usual payload type and meaning of a `g7:DATE`.
> - `PHRASE` is the structure type expected with that tag as a substructure of `g7:DATE`: namely, `g7:PHRASE` (p.90).
> - `_DATE` can not be used as a substructure of `BIRT` (p.75) because `BIRT` has a documented `g7:DATE` substructure with tag `DATE`.
> - `BIRT` can not be used as a substructure of `_DATE` or `_DATE.PHRASE` because neither structure type has a documented substructure with tag `BIRT`.

Because all substructures have meanings defined relative to their superstructures and no records do, standard records cannot be relocated and relocated standard structures cannot be used as records.

> *Example* — The `g7:PLAC` (p.91) substructure documents where the event described in its superstructure occurred. If an application wants a record to describe the place itself it should create a new URI for that extension; reusing `g7:PLAC` for a record with no superstructure is not appropriate.

All other non-standard structures are prohibited. Examples of prohibited structures include, but are not limited to,

- a record or substructure of a standard structure using a tag matching production `stdTag` (p.13) that is not defined in this document;
- any substructure with cardinality `{0:1}` appearing more than once;
- a standard substructure appearing as a record or vice-versa;
- a standard structure whose payload does not match the requirements of this document.

> *Note* — In some cases, an extension may need to allow multiple structures where this document allows only 1. The recommended way to do this is to create an extension tag and URI and serve a page describing how the semantics of the structure have been extended to allow multiple instances.
>
> > *Example* — Suppose I have multiple sources that give different ages of the wife at a wedding; however, this specification allows only 1 `MARR` (p. 69).`WIFE.AGE`. An extension could not include multiple `MARR.WIFE` nor `MARR.WIFE.AGE`, but could define a new extension `_AGE`, give it a URL, and provide the following definition of this extension structure type at that URL:
> >
> > > Alternate age: an age attested by some source, but not accepted by the researcher as the actual age of the individual. If the age is accepted by the researcher, the standard tag `AGE` (p.74) should be used instead.
> >
> > This alternate age extension structure could be used as follows:
>
> ```
> 1 MARR
> 2 WIFE
> 3 AGE 27y
> 3 _AGE 22y
> ```

Enumerated values may be extended with new values that match production `extTag` (p. 13). Enumerations may not use standard values from other enumeration sets.

> *Example* — The following is not allowed because `PARENT` is defined as a value for `ROLE` (p.94), not for `RESN` (p.93)
>
> ```
> 0 @BAD@ INDI
> 1 RESN PARENT
> 1 NOTE The above enumeration value is not allowed
> ```

Dates may be extended provided they use a calendar that matches production `extTag`. Dates with extension calendars may also use extension months and epochs.

### 1.5.1. Extension Tags

Each use of the `extTag` production is called an extension tag, including when used as a tag, calendar, month, epoch, or enumerated value. Each `extTag` is either a *documented extension tag* or an *undocumented extension tag*. It is recommended that documented extension tags be used instead of undocumented extension tags wherever possible.

A **documented extension tag** is a tag that is mapped to a URI using the schema structure. The schema structure is a substructure of the header with tag `SCHMA` (p.94). It should appear within the document before any extension tags. The schema's substructures are tag definitions.

A tag definition is a structure with tag `TAG` (p.96). Its payload is an Tag Definition (p.36), which includes both an extension tag and a URI, and defines that extension tag to be an abbreviation for that URI within the current document.

> *Example* — The following header
>
> ```
> 0 HEAD
> 1 SCHMA
> 2 TAG _SKYPEID http://xmlns.com/foaf/0.1/skypeID
> 2 TAG _MEMBER http://xmlns.com/foaf/0.1/member
> ```
>
> defines the following tags
>
> | Tag | Means |
> |---|---|
> | `_SKYPEID` | `http://xmlns.com/foaf/0.1/skypeID` |
> | `_MEMBER` | `http://xmlns.com/foaf/0.1/member` |
>
> Note that at the time of writing, the FOAF URIs used in this example are not URLs.

The meaning of a documented extension tag is identified by its URI, not its tag. As such each documented extension tag needs its own URI: it is its URI, not its tag, that defines its meaning. Documented extension tags can be changed freely by modifying the schema, though it is recommended that documented extension tags not be changed. However, a tag change may be necessary if a product picks the same tags for URIs that another product uses for different URIs. A given schema should map only one tag to each URI.

> *Example* — The following 2 document fragments are semantically equivalent and a system importing one may export it as the other without change of meaning.
>
> ```
> 0 HEAD
> 1 SCHMA
> 2 TAG _SKYPEID http://xmlns.com/foaf/0.1/skypeID
> 0 @I0@ INDI
> 1 _SKYPEID example.person
> ```
>
> ```
> 0 HEAD
> 1 SCHMA
> 2 TAG _SI http://xmlns.com/foaf/0.1/skypeID
> 0 @I0@ INDI
> 1 _SI example.person
> ```

It is recommended that the URIs used for documented extension tags be URLs that can be used to access documentation for the meaning of the tag.

> *Note* — The W3C has an interest group note that discusses several ways of achieving this URI/URL mapping, including how a single webpage can describe multiple tags using either HTTP redirects (which requires some server setup) or what they call “Hash URIs” (which require no setup).
>
> That interest group note also explains why it might be desirable to have a separate URIs for a concept and the document describing that concept. Because of the structure of the schema, that separation is less important for FamilySearch GEDCOM 7 than it is for the semantic web, but it remains good advice where feasible.

The schema structure may contain the same tag more than once with different URIs. Reusing tags in this way must not be done unless the concepts identified by those URIs cannot appear in the same place in a dataset, and should not be done unless the URIs identify closely related concepts.

> *Example* — Consider three extensions:
>
> - `https://example.com/LocationRecord`, a record that describes a location.
> - `https://example.com/LocationPointer`, a substructure of most events that points to a `https://example.com/LocationRecord`.
> - `https://example.com/inLocoParentis`, a substructure of some events indicating a non-parent entity that filled the legal role of a parent for that event.
>
> Given this, we have the following:
>
> - `https://example.com/LocationPointer` and `https://example.com/inLocoParentis` must not be given the same tag because they can appear in the same place in a dataset.
> - `https://example.com/LocationRecord` and `https://example.com/inLocoParentis` may be given the same tag, but should not be given the same tag because they identify unrelated concepts.
> - `https://example.com/LocationRecord` and `https://example.com/LocationStructure` may be given the same tag.
>
> One way to satisfy these constraints and recommendations is with the following schema:

```
1 SCHMA
2 TAG _LOC https://example.com/LocationRecord
2 TAG _LOC https://example.com/LocationPointer
2 TAG _ILP https://example.com/inLocoParentis
```

An extension tag that is not given a URI in the schema structure is called an **undocumented extension tag**. The meaning of an undocumented extension tag is identified by its superstructure type and its tag.

### 1.5.2. Requirements and Recommendations

- It is recommended that applications not use undocumented extension tags.
- It is required that each tag definition's extension tag be unique within the document.
- It is recommended that each documented extension tag's URI be unique within the document.
- It is recommended that extension creators use URLs as their URIs and serve a YAML file describing the meaning of an extension at its URL, such as a file in the GEDCOM structure registry. The FamilySearch GEDCOM YAML file format defines how the YAML file provides machine-readable documentation and metadata about extensions. For example, the `superstructures` key in the YAML file can be used to disambiguate which meaning applies within a given context, such as in the `_LOC` example above where the meaning is context-dependent.

### 1.5.3. Extension versus Standard

Standard structures take priority over extensions. Data contained in extension tags will not be interpreted by other systems correctly unless the other system supports that particular extension. In particular, those supporting extensions should keep in mind the following:

- If a standard structure is present that contradicts an extension that is present, the standard structure has priority and the extension should be updated to align with it.

  > *Example* — If a document has an extension `_ISODATE` in ISO 8601 format that disagrees with a `DATE` (p.78) in the `DateValue` (p.28) format, the `DATE` shall be taken as more correct and the `_ISODATE` updated to reflect that.

- If a standard structure can be extracted as a subset of the semantics of an extension, the standard tag must be generated along with the extension and kept in sync with it by systems understanding the extension.

  > *Example* — If a document has an extension `_LOC` providing a detailed hierarchical place representation with historical names, boundaries, and the like, it must also generate the corresponding `PLAC` (p.91) structures with the subset of that information which `PLAC` can represent.

- If an extension can be extracted as a subset of the semantics of a standard structure, or if the extension and standard structure only sometimes align, then the standard structure should be included if and only if the semantics align in this case.

  > *Example* — If a document has an extension `_PARTNER` that generalizes `HUSB` (p.83) and `WIFE` (p.101) and some `ASSO` (p.75) `ROLE` (p.94)s, then it should pair the extension with those standard structures if and only if it knows which one applies.

  > *Example* — If a document has an extension `_HOUSEHOLD` that is the same as `FAM` in some situations but not in others, then it should keep the `_HOUSEHOLD` and `FAM` in sync if and only if they align.

- Six standard structure types are exceptions to these rules: `NOTE` (p.89), `SNOTE` (p.95), `INDI.EVEN` (p.80), `FAM.EVEN` (p.80), `INDI.FACT`, and `FAM.FACT`. Each of these allows human-readable text to describe information that cannot be captured in more-specific structures. As such, all other structures express information that could be described using 1 or more of those structure types. Extensions do not need to duplicate their information using any of those structures.

  > *Example* — If a document has an extension `_MEMBER` that indicates membership in clubs, boards, and other groups, it is not required to duplicate that information in an `INDI.FACT` because `INDI.FACT` is 1 of the 6 special structure types listed above.

  > *Example* — If a document has an extension `_WEIGHT` that describes the weight of a person, it must duplicate that information in an `INDI.DSCR` because `INDI.DSCR` is not 1 of the 6 generic structure types.

## 1.6. Removing data

There may be situations where data needs to be removed from a dataset, such as when a user requests its deletion or marks it as confidential and not for export.

In general, removed data should result in removed structures.

Pointers to a removed structure should be replaced with `voidPtr` (p.13)s.

If removal of a structure makes the superstructure invalid because the superstructure required the substructure, the structure should instead be retained and have its payload changed to a `voidPtr` if a pointer, or to a data type-appropriate empty value if a non-pointer.

If removing a structure leaves its superstructure with no payload and no substructures, the superstructure should also be removed.

A structure can also be removed if it provides no new information. For example,

```
0 @I1@ INDI
1 NAME John /Doe/
1 NAME John /Doe/
1 FAMC @F1@
1 FAMC @F1@
0 @F1@ FAM
1 CHIL @I1@
1 CHIL @I1@
```

provides no information beyond the simpler form:

```
0 @I1@ INDI
1 NAME John /Doe/
```

# 2. Data types

Every line value (with any continuation pseudo-structures) is a string. However, those strings can encode 1 of several conceptual data types.

## 2.1. Text

A free-text string is text in a human language. Conceptually, it may be either a user-generated string or a source-generated string. Programmatically, both are treated as unconstrained sequences of characters with an associated language.

```
anychar = %x09-10FFFF ; but not banned, as with all ABNF rules
Text    = *anychar
```

The URI for the `Text` data type is `xsd:string`.

## 2.2. Integer

An integer is a non-empty sequence of ASCII decimal digits and represents a non-negative integer in base-10. Leading zeros have no semantic meaning and should be omitted.

```
Integer = 1*DIGIT
```

Negative integers are not supported by this specification.

The URI for the `Integer` data type is `xsd:nonNegativeInteger`.

## 2.3. Enumeration

An enumeration is a selection from a set of options. They are represented as a string matching the same production as a tag, with the additional permission that standard enumerations may be integers.

```
stdEnum = stdTag / Integer
Enum    = stdEnum / extTag
```

Each structure type with an enumeration payload also defines specific payload values it permits. These permitted payloads match production `stdEnum` and should each have a defined URI. Payload values that match production `extTag` (p.13) are always permitted in structures with an enumeration payload and have their URI defined by the schema.

Each enumeration value has a distinct meaning as identified by its corresponding URI.

The URI of a given tag in an enumeration payload is determined by the tag itself and by the structure type of the structure it is in the payload of.

> *Example* — The tag `HUSB` (p.83) is used in this document to represent two enumeration values. Which one is meant can be identified by the structure type it appears in as follows:
>
> | Containing structure type | Enumeration value identified by tag `HUSB` |
> |---|---|
> | `g7:FAMC-ADOP` (p.73) | `g7:enum-ADOP-HUSB` |
> | `g7:ROLE` (p.94) | `g7:enum-HUSB` |
>
> An extension (p.18) could also place either of these enumeration values in an extension structure type; the extension authors should document which one they permit.
>
> The `HUSB` tag is also used to identify two different structure types, `g7:FAM-HUSB` (p.83) and `g7:HUSB`.

The URI for the `Enum` (p.27) data type is `g7:type-Enum`.

## 2.4. Date

The date formats defined in this specification include the ability to store approximate dates, date periods, and dates expressed in different calendars.

Technically, there are 3 distinct date data types:

- `DateValue` is a generic type that can express many kinds of dates.
- `DateExact` is used for timestamps and other fully-known dates.
- `DatePeriod` is used to express time intervals that span multiple days.

```
DateValue   = [ date / DatePeriod / dateRange / dateApprox ]
DateExact   = day D month D year  ; in Gregorian calendar
DatePeriod  = [ %s"TO" D date ]
            / %s"FROM" D date [ D %s"TO" D date ]
            ; note both DateValue and DatePeriod can be the empty string

date        = [calendar D] [[day D] month D] year [D epoch]
dateRange   = %s"BET" D date D %s"AND" D date
            / %s"AFT" D date
            / %s"BEF" D date
dateApprox  = (%s"ABT" / %s"CAL" / %s"EST") D date

dateRestrict = %s"FROM" / %s"TO" / %s"BET" / %s"AND" / %s"BEF"
             / %s"AFT" / %s"ABT" / %s"CAL" / %s"EST"

calendar = %s"GREGORIAN" / %s"JULIAN" / %s"FRENCH_R" / %s"HEBREW"
         / extTag

day     = Integer
year    = Integer
month   = stdTag / extTag  ; constrained by calendar
epoch   = %s"BCE" / extTag ; constrained by calendar
```

In addition to the constraints above:

- The allowable `day` (p.28)s, `month`s, `year`s, and `epoch`s are determined by the `calendar`. All known calendars restrict `day` to be between 1 and a month-specific maximum. The largest known maximum is 36, and most months in most calendars have a lower maximum.
- No calendar names, months, or epochs match `dateRestrict`.
- Extension calendars (those with `extTag` for their `calendar`) must use `extTag`, not `stdTag`, for months.

It is recommended that calendars avoid using a single tag to refer to both a month and an epoch.

An absent `calendar` is equivalent to the calendar `GREGORIAN` (p.114).

The grammar above allows for `date`s to be preceded by various words. The meaning of these words is given as follows:

| Production | Meaning |
|---|---|
| `FROM` *x* | Lasted for multiple days, beginning on *x*. |
| `TO` *x* | Lasted for multiple days, ending on *x*. |
| `BET` *x* | Exact date unknown, but no earlier than *x*. |

| Production | Meaning |
|---|---|
| `AND` *x* | Exact date unknown, but no later than *x*. |
| `BEF` *x* | Exact date unknown, but no later than *x*. |
| `AFT` *x* | Exact date unknown, but no earlier than *x*. |
| `ABT` *x* | Exact date unknown, but near *x*. |
| `CAL` *x* | *x* is calculated from other data. |
| `EST` *x* | Exact date unknown, but near *x*; and *x* is calculated from other data. |

Known calendars and tips for handling dual dating and extension calendars are given in Appendix A: Calendars and Dates (p.114).

`DateValue` and `DatePeriod` payloads may also be the empty string if no suitable form is known but a substructure (such as a `PHRASE` (p.90) or `TIME` (p.96)) is desired.

> *Note* — Versions 5.3 through 5.5.1 allowed phrases inside `DateValue` payloads. Date phrases were moved to the `PHRASE` substructure in version 7.0. A current limitation, however, is that a phrase in the `PHRASE` substructure cannot specify a language, so if a non-default language is needed to correctly interpret the phrase two options exist:
>
> - `PHRASE` can be used with a documented extension tag for the language, as discussed in `g7:LANG` (p.84).
> - `<<EVENT_DETAIL>>.SOUR.DATA.TEXT` can be used instead along with a `LANG` substructure; this loses the connection with the date, but includes the language with a standard tag.

> *Note* — As defined by the grammar above, every date must have a year. If no year is known, the entire date may be omitted.
>
> > *Example* — The following is an appropriate way to handle a missing year
> >
> > ```
> > 2 DATE
> > 3 PHRASE 5 January (year unknown)
> > ```

The URI for the `DateValue` (p.28) data type is `g7:type-Date`.

The URI for the `DateExact` data type is `g7:type-Date#exact`.

The URI for the `DatePeriod` data type is `g7:type-Date#period`.

## 2.5. Time

Time is represented on a 24-hour clock (for example, 23:00 rather than 11:00 PM). It may be represented either in event-local time or in Coordinated Universal Time (UTC). UTC is indicated by including a `Z` (U+005A) after the time value; event-local time is indicated by its absence. When a time is used together with a `DateExact`, it is recommended that UTC time be used rather than event-local time.

```
Time     =  hour ":" minute [":" second ["." fraction]] [%s"Z"]

hour     = DIGIT / ("0" / "1") DIGIT / "2" ("0" / "1" / "2" / "3")
minute   = ("0" / "1" / "2" / "3" / "4" / "5") DIGIT
second   = ("0" / "1" / "2" / "3" / "4" / "5") DIGIT
fraction = 1*DIGIT
```

> *Note* — The above grammar prohibits end-of-day instant `24:00:00` and leap-seconds. It allows both `02:50` and `2:50` as the same time.

The URI for the `Time` (p.31) data type is `g7:type-Time`.

## 2.6. Age

Ages are represented by counts of years, months, weeks, and days.

```
Age         = [[ageBound D] ageDuration]

ageBound    = "<" / ">"
ageDuration = years [D months] [D weeks] [D days]
            / months [D weeks] [D days]
            / weeks [D days]
            / days

years   = Integer %x79    ; 35y
months  = Integer %x6D    ; 11m
weeks   = Integer %x77    ; 8w
days    = Integer %x64    ; 21d
```

Where

| Production | Meaning |
|---|---|
| `<` | The real age was less than the provided age |
| `>` | The real age was greater than the provided age |

| Production | Meaning |
|---|---|
| `years` | a number of years |
| `months` | a number of months |
| `weeks` | a number of weeks |
| `days` | a number of days |

Non-integer numbers should be rounded down to an integer. Thus, if someone has lived for 363.5 days, their age might be written as `363d`, `51w 6d`, `51w`, `0y`, etc.

Because numbers are rounded down, `>` effectively includes its endpoint; that is, the age `> 8d` includes people who have lived 8 days + a few seconds.

Different cultures count ages differently. Some increment years on the anniversary of birth and others at particular seasons. Some round to the nearest year, others round years down, others round years up. Because users may be unaware of these traditions or may fail to convert them to the round-down convention, errors in age of up to a year are common.

> *Note* — Because age payloads are intended to allow recording the age as it was recorded in records that could contain errors, odd ages such as `8w 30d`, `1y 400d`, `1y 30m`, etc. are permitted. Some applications might convert these to more standard forms; if so, it is recommended that they use a `PHRASE` (p.90) substructure to hold the original form.

Age payloads may also be omitted entirely if no suitable form is known but a substructure (such as a `PHRASE`) is desired.

> *Note* — Versions 5.5 and 5.5.1 allowed a few specific phrases inside `Age` (p.31) payloads. Age phrases were moved to the `PHRASE` substructure in 7.0.

The URI for the `Age` data type is `g7:type-Age`.

## 2.7. List

A list is a meta-syntax representing a sequence of values with another data type. Two list data types are used in this document: List:Text and List:Enum. Lists are serialized in a comma-separated form, delimited by a comma (U+002C `,`) and any number of spaces (U+0020) between each item. It is recommended that a comma-space pair (U+002C U+0020) be used as the delimiter.

```
list      = listItem *(listDelim listItem)
listItem  = [ nocommasp / nocommasp *nocomma nocommasp ]
listDelim = *D "," *D
nocomma   = %x09-2B / %x2D-10FFFF
nocommasp = %x09-1D / %x21-2B / %x2D-10FFFF

List-Text = list
List-Enum = Enum *(listDelim Enum)
```

If valid for the underlying type, empty strings may be included in a list by having no characters between delimiters.

> *Example* — A `List:Text` with value "`,  ,  one,  more,`" has 5 `Text` (p.27)-type values: 2 empty strings, the string "`one`", the string "`more`", and 1 more empty string.

There is no escaping mechanism to allow lists of entries that begin or end with spaces or that contain comma characters.

The URI for the `List:Text` data type is `g7:type-List#Text`.

The URI for the `List:Enum` data type is `g7:type-List#Enum`.

## 2.8. Personal Name

A personal name is mostly free-text. It should be the name as written in the culture of the individual and should not contain line breaks, repeated spaces, or characters not part of the written form of a name (except for U+002F as explained below).

```
PersonalName = nameStr
             / [nameStr] "/" [nameStr] "/" [nameStr]

nameChar     = %x20-2E / %x30-10FFFF  ; any but '/' and '\t'
nameStr      = 1*nameChar
```

The character U+002F (`/`, slash or solidus) has special meaning in a personal name, being used to delimit the portion of the name that most closely matches the concept of a surname, family name, or the like. This specification does not provide any standard way of representing names that contain U+002F.

The URI for the `PersonalName` data type is `g7:type-Name`.

## 2.9. Language

The language data type represents a human language or family of related languages, as defined in BCP 47. It consists of a sequence of language subtags separated by hyphens, where language subtags are registered by the IANA.

The ABNF grammar for language tags is given in BCP 47, section 2.1, production `Language-Tag`.

The URI for the `Language` data type is `xsd:Language`.

## 2.10. Media Type

The media type data type represents the encoding of information in bytes or characters, as defined in RFC 2045 and registered by the IANA.

The official grammar for media type is given in RFC 2045, section 5.1, which defines the syntax of registered values and extension values.

```
MediaType = type "/" subtype parameters
```

where:

- `type` and `subtype` are defined in RFC 2045 section 5.1, and registered values (i.e., those not beginning with "x-") are further constrained by the definitions in RFC 6838, section 4.2. A registry of media types is maintained publicly by the IANA.
- `parameters` is defined in RFC 9110, section 5.6.6. Note that the `parameters` definition in GEDCOM matches that used by HTTP headers which permit whitespace around the ";" delimiter, whereas email headers in RFC 2045 do not.

The URI for the `MediaType` (p.34) data type is `dcat:mediaType`.

## 2.11. Special

The special data type is a string conforming to a case-specific standard or constraints. The constraints on each special data type instance are either unique to that structure type or are not simply expressed. For example, the payload of an `IDNO` (p.84) structure may obey different rules for each possible `TYPE` (p.99) substructure.

Each special data type is distinct. The URI for the generic data type subsuming all `Special` data types is `xsd:string` (the same as the `Text` (p.27) data type).

```
Special = Text
```

# 2.12. File Path

The file path data type describes where an digital file is located in a machine-readable way. Syntactically, the payload is a “valid URL string” as defined by the WHATWG URL specification. That is, it can be an absolute or relative URL, optionally with a fragment string, and can contain non-ASCII characters that are permitted in a valid URL string. It can also include percent-encoded bytes.

Version 7.0 only supports the following URLs:

- A URL with scheme `ftp`, `http`, or `https` refers to a **web-accessible file**.
- A URL with scheme `file` refers to either a **local file** or a **non-local file**, as defined by RFC 8089. Local file URLs must not be used in FamilySearch GEDZIP (p.110) and should be avoided in datasets that are expected to be shared on the web or with unknown parties, but may be appropriate for close collaboration between parties with known similar file structures.
- A URL with all of the following:
  - no scheme
  - not beginning with `/` (U+002F)
  - not containing any path segments equal to `..` (U+002E U+002E)
  - not containing a reverse solidus character (U+005C `\`) or banned (p.10) character, either directly or in escaped form
  - no query or fragment

  refers to a **local file**. If the dataset is part of a GEDZIP file (p.110), the URL of the local file is a zip archive filename; otherwise, the URL of a local file is resolved with *base* equal to the directory containing the dataset.

  It is recommended that local files use the directory prefix `media/`, but doing so is not required.

  For compatibility with GEDZIP (p.110) and related formats, it is recommended that the following file paths not be used:

  - `gedcom.ged`
  - `MANIFEST.MF`
  - any URL beginning `META-INF/`

Additional URLs may be supported in future versions of this specification.

The URI for the `FilePath` data type is `g7:type-FilePath`.

## 2.13. URI

The URI data type is used to provide agent-controlled durable identifiers for technically-precise content. URIs are not generally intended to be user-facing nor for storing URIs that are found in historical documents; rather, they are used as machine-readable identifiers with formally-defined meaning.

The payload is a "URI Reference" as defined in RFC 3986 section 4.1 with ABNF production `URI-reference`. The URI Reference is a more restrictive syntax than the URL Strings permitted by the File Path (p.35) data type, faciltiating easier automated equality tests between URIs.

Relative URIs should be avoided in datasets that are expected to be shared on the web or with unknown parties, but may be appropriate for close collaboration between parties with a shared base URI.

The URI for the `URI` data type is `xsd:anyURI`.

## 2.14. Tag Definition

A tag definition consists of an extension tag, a space, and a URI. It defines that the extension tag is used to refer to the concept identified by that URI. See Extension Tags (p.21) for more details.

The URIs in Tag Definitions have the same requirements and recommendations as those defined by the URI (p.36) data type.

```
TagDef = extTag D URI-reference
```

The URI for the `TagDef` data type is `g7:type-TagDef`.

## 2.15. Latitude

A latitudinal coordinate. The payload is either `N` (for a coordinate north of the equator) or `S` (for a coordinate south of the equator) followed by a decimal number of degrees. Minutes and seconds are not used and should be converted to fractional degrees prior to encoding. The number of degrees is limited by definition to be between 0 (the equator) and 90 (the north or south pole).

```
Latitude = ("N" / "S") upto90 [ "." 1*digit]
upto90   = "90" / [upto8] digit
upto8    = "0" / "1" / "2" / "3" / "4" / "5" / "6" / "7" / "8"
```

> *Example* — 18 degrees, 9 minutes, and 3.4 seconds North would be formatted as `N18.150944`.

The URI for the `Latitude` data type is `g7:type-Latitude`.

## 2.16. Longitude

A longitudinal coordinate. The payload is either `E` (for a coordinate east of the prime meridian) or `W` (for a coordinate west of the prime meridian) followed by a decimal number of degrees. Minutes and seconds are not used and should be converted to fractional degrees prior to encoding. The number of degrees is limited by definition to be between 0 (the prime meridian) and 180 (the 180th meridian).

```
Longitude = ("N" / "S") upto180 [ "." 1*digit]
upto180   = "180" / "1" upto7 digit / [["0"] digit] digit
upto7     = "0" / "1" / "2" / "3" / "4" / "5" / "6" / "7"
```

> *Example* — 168 degrees, 9 minutes, and 3.4 seconds East would be formatted as `E168.150944`.

The URI for the `Longitude` data type is `g7:type-Longitude`.

# 3. Genealogical structures

This chapter describes a set of structure types for exchanging family-based lineage-linked genealogical information. Lineage-linked data pertains to individuals linked in family relationships across multiple generations.

The genealogical structures defined in this chapter are based on the general framework of the container format and data types defined in Chapters 1 and 2.

Historically, these genealogical structures were used as the only form approved for exchanging data with Ancestral File, TempleReady and other Family History resource files. Those systems were all replaced between 1999 and 2019, and GEDCOM-X was introduced as the new syntax for communication with their replacements. FamilySearch GEDCOM 7.0 and GEDCOM-X have similar expressive power, but as of 2021 GEDCOM is more common for exchanging single-researcher files between applications and GEDCOM-X is more common for transferring bulk data and communication directly between applications.

The basic description of the genealogical structures' organization is presented in the following 3 major sections:

- "Structure Organization (p.40)" describes records and other nested structures.
- "Structure Meaning (p.66)" provides a definition of each structure by its tag.
- "Enumeration Values (p.102)" provides a definition of each enumeration value by its containing structure.

## 3.1. A Metasyntax for Structure Organization

The structures, with their payloads and substructures, are represented using a custom metasyntax. The intent of this metasyntax is to resemble the line encoding of allowable structures. In the metasyntax:

- Options are placed between brackets `[` and `]` and have choices separated by pipes `|`.
- Named sets of rules are indicated with a name followed by `:=`.
- Level markers are used to indicate substructure relationships.
  - `0` means "must be a record".
  - `n` means "level inherited from rule instantiation".
  - `+1`, `+2`, etc., indicate nesting within nearest preceding structure with lesser level.

- Four cardinality markers are used: `{0:1}`, `{1:1}`, `{0:M}`, and `{1:M}`.
  - `{0:` means "optional" – the structure may be omitted
  - `{1:` means "required" – at least 1 must appear
  - `:M}` means "any number" – 1 or more structures may appear. Unless otherwise specified, the first is the most-preferred value. If an application needs to display just 1 of several `NAME` (p.88)s, `BIRT` (p.75)s, etc, they should show the first such structure unless more specific selection criteria are available.
  - `:1}` means "singular" – at most 1 may appear; a second must not be present.

  Systems interested in violating the cardinality rules should instead create extension structures (p.18) with different cardinality.
- Rule instantiation is indicated by the rule name in double angle-brackets (such as `<<`rule name`>>`) and a cardinality marker.

  The cardinality markers of rule instantiations and their referenced line templates are combined such that the resulting cardinality is required only if both combined cardinalities are required and singular only if both combined cardinalities are singular.

  > *Example —* The definition of the `FAM` record has the line
  >
  > ```
  > +1 <<CREATION_DATE>>                          {0:1}
  > ```
  >
  > and the `CREATION_DATE` (p.52) rule begins
  >
  > ```
  > n CREA                                        {1:1}  g7:CREA
  > ```
  >
  > Thus, a `FAM` record has an optional singular `CREA` (p.77) substructure (such as cardinality `{0:1}`).

- Line templates have several parts:
  - An optional cross-reference template `@XREF:`tag`@`, meaning this structure may be pointed to by other structures.

    Structures that are not pointed to by other structures need not have a cross-reference identifier (p.13) even if their line template has a cross-reference template.
  - The standard tag for this structure.
  - An optional payload descriptor; if present this is 1 of the following:
    - `@<XREF:`tag`>@` means a pointer to a structure with this cross-reference template; `@VOID@` is also permitted.
    - `<`data type`>` means a non-pointer payload, as described in Data types (p.27). If the data type allows the empty string, the payload may be omitted.
    - `[`text`|<NULL>]` means the payload is optional but if present must be the given text.

If there is a payload descriptor, a payload that matches the payload is required of the described structure unless the descriptor says the payload is optional.

If there is no payload descriptor, the described structure must not have a payload.

- A cardinality marker.
- The URI of this structure type.

  Pseudo-structures do not have a URI.

- Within the metasyntax, the order in which substructures are presented within a structure and the order in which choices are presented within an option set are not significant unless otherwise specified in the text next to the metasyntax block.

The context of a structure's superstructure may be necessary in addition to the structure's standard tag to fully determine its structure type. To refer to a structure in the context of its superstructure, tags are written with intervening periods. For example, `GEDC.VERS` (p.101) refers to a structure with tag `VERS` (p.101) and a superstructure with tag `GEDC` (p.83).

# 3.2. Structure Organization

## 3.2.1. Document

### Dataset :=

```
0 <<HEADER>>                                {1:1}
0 <<RECORD>>                                {0:M}
0 TRLR                                      {1:1}  g7:TRLR
```

The order of these is significant: the `HEADER` (p.42) must come first and `TRLR` (p.99) must be last, with any `RECORD`s in between.

**RECORD** :=

```
[
n <<FAMILY_RECORD>>                    {1:1}
|
n <<INDIVIDUAL_RECORD>>                {1:1}
|
n <<MULTIMEDIA_RECORD>>                {1:1}
|
n <<REPOSITORY_RECORD>>                {1:1}
|
n <<SHARED_NOTE_RECORD>>               {1:1}
|
n <<SOURCE_RECORD>>                    {1:1}
|
n <<SUBMITTER_RECORD>>                 {1:1}
]
```

**HEADER** :=

```
n HEAD                                  {1:1}  g7:HEAD
  +1 GEDC                               {1:1}  g7:GEDC
     +2 VERS <Special>                  {1:1}  g7:GEDC-VERS
  +1 SCHMA                              {0:1}  g7:SCHMA
     +2 TAG <TagDef>                    {0:M}  g7:TAG
  +1 SOUR <Special>                     {0:1}  g7:HEAD-SOUR
     +2 VERS <Special>                  {0:1}  g7:VERS
     +2 NAME <Text>                     {0:1}  g7:NAME
     +2 CORP <Text>                     {0:1}  g7:CORP
        +3 <<ADDRESS_STRUCTURE>>        {0:1}
        +3 PHON <Special>               {0:M}  g7:PHON
        +3 EMAIL <Special>              {0:M}  g7:EMAIL
        +3 FAX <Special>                {0:M}  g7:FAX
        +3 WWW <Special>                {0:M}  g7:WWW
     +2 DATA <Text>                     {0:1}  g7:HEAD-SOUR-DATA
        +3 DATE <DateExact>             {0:1}  g7:DATE-exact
           +4 TIME <Time>               {0:1}  g7:TIME
        +3 COPR <Text>                  {0:1}  g7:COPR
  +1 DEST <Special>                     {0:1}  g7:DEST
  +1 DATE <DateExact>                   {0:1}  g7:HEAD-DATE
     +2 TIME <Time>                     {0:1}  g7:TIME
  +1 SUBM @<XREF:SUBM>@                 {0:1}  g7:SUBM
  +1 COPR <Text>                        {0:1}  g7:COPR
  +1 LANG <Language>                    {0:1}  g7:HEAD-LANG
  +1 PLAC                               {0:1}  g7:HEAD-PLAC
     +2 FORM <List:Text>                {1:1}  g7:HEAD-PLAC-FORM
  +1 <<NOTE_STRUCTURE>>                 {0:1}
```

The header pseudo-structure provides metadata about the entire dataset. A few sub-structures of note:

- `GEDC` (p.83) identifies the specification that this document conforms to. It is recommended that `GEDC` be the first substructure of the header.
- `SCHMA` (p.94) gives the meaning of extension tags; see Extensions (p.18) for more details.
- `SOUR` (p.95) describes the originating software.
  - `CORP` (p.77) describes the corporation creating the software.
  - `HEAD.SOUR.DATA` (p.78) describes a larger database, electronic data source, or digital repository this data is extracted from.
- `LANG` (p.84) and `PLAC` (p.91) give a default value for the rest of the document.

> *Deprecation Note* — `HEAD.SOUR.DATA` is now deprecated and applications should use `HEAD.SOUR.NAME` instead.

### 3.2.2. Records

**`FAMILY_RECORD` :=**

```
n @XREF:FAM@ FAM                                {1:1}  g7:record-FAM
  +1 RESN <List:Enum>                           {0:1}  g7:RESN
  +1 <<FAMILY_ATTRIBUTE_STRUCTURE>>             {0:M}
  +1 <<FAMILY_EVENT_STRUCTURE>>                 {0:M}
  +1 <<NON_EVENT_STRUCTURE>>                    {0:M}
  +1 HUSB @<XREF:INDI>@                         {0:1}  g7:FAM-HUSB
     +2 PHRASE <Text>                           {0:1}  g7:PHRASE
  +1 WIFE @<XREF:INDI>@                         {0:1}  g7:FAM-WIFE
     +2 PHRASE <Text>                           {0:1}  g7:PHRASE
  +1 CHIL @<XREF:INDI>@                         {0:M}  g7:CHIL
     +2 PHRASE <Text>                           {0:1}  g7:PHRASE
  +1 <<ASSOCIATION_STRUCTURE>>                  {0:M}
  +1 SUBM @<XREF:SUBM>@                         {0:M}  g7:SUBM
  +1 <<LDS_SPOUSE_SEALING>>                     {0:M}
  +1 <<IDENTIFIER_STRUCTURE>>                   {0:M}
  +1 <<NOTE_STRUCTURE>>                         {0:M}
  +1 <<SOURCE_CITATION>>                        {0:M}
  +1 <<MULTIMEDIA_LINK>>                        {0:M}
  +1 <<CHANGE_DATE>>                            {0:1}
  +1 <<CREATION_DATE>>                          {0:1}
```

The `FAM` record was originally structured to represent families where a male `HUSB` (p.83) (husband or father) and female `WIFE` (p.101) (wife or mother) produce `CHIL` (p.76) (children). The `FAM` record may also be used for cultural parallels to this, including nuclear families, marriage, cohabitation, fostering, adoption, and so on, regardless of the gender of the partners. Sex, gender, titles, and roles of partners should not be inferred based on the partner that the `HUSB` or `WIFE` structure points to.

The individuals pointed to by the `HUSB` and `WIFE` are collectively referred to as “partners”, “parents” or “spouses”.

Some displays may be unable to display more than 2 partners. Displays may use `HUSB` and `WIFE` as layout hints, for example, by consistently displaying the `HUSB` on the same side of the `WIFE` in a tree view. Family structures with more than 2 partners may either use several `FAM` records or use `ASSOCIATION_STRUCTURE` (p.51)s to indicate additional partners. `ASSO` (p.75) should not be used for relationships that can be expressed using `HUSB`, `WIFE`, or `CHIL` instead.

> *Note* — The FAM record will be revised in a future version to more fully express the diversity of human family relationships.

The order of the CHIL (children) pointers within a FAM (family) structure should be chronological by birth; this is an exception to the usual "most preferred value first" rule. A CHIL with a voidPtr (p.13) indicates a placeholder for an unknown child in this birth order.

If a FAM record uses HUSB or WIFE to point to an INDI record, the INDI record must use FAMS (p.82) to point to the FAM record. If a FAM record uses CHIL to point to an INDI record, the INDI record must use a FAMC (p.82) to point to the FAM record.

An INDI record should not have multiple FAMS substructures pointing to the same FAM.

A FAM record should not have multiple CHIL substructures pointing to the same INDI; doing so implies a nonsensical birth order. An INDI record may have multiple FAMC substructures pointing to the same FAM, but doing so is not recommended.

Source citations and notes related to the start of a specific child relationship should be placed under the child's BIRT (p.75), CHR (p.76), or ADOP (p.73) event, rather than under the FAM record.

If an INDI that can be reached from a FAM by following CHIL and FAMS pointers can also be reachable by following HUSB, WIFE, and FAMC pointers, then that implies that a person is their own ancestor/descendant. In most cases that would be an error, though it is theoretically possible that such a situation could occur with non-biological relationships (marriages, adoptions, etc.).

### `INDIVIDUAL_RECORD` :=

```
      n @XREF:INDI@ INDI                          {1:1}  g7:record-INDI
        +1 RESN <List:Enum>                       {0:1}  g7:RESN
        +1 <<PERSONAL_NAME_STRUCTURE>>            {0:M}
        +1 SEX <Enum>                             {0:1}  g7:SEX
        +1 <<INDIVIDUAL_ATTRIBUTE_STRUCTURE>>     {0:M}
        +1 <<INDIVIDUAL_EVENT_STRUCTURE>>         {0:M}
        +1 <<NON_EVENT_STRUCTURE>>                {0:M}
        +1 <<LDS_INDIVIDUAL_ORDINANCE>>           {0:M}
        +1 FAMC @<XREF:FAM>@                      {0:M}  g7:INDI-FAMC
           +2 PEDI <Enum>                         {0:1}  g7:PEDI
              +3 PHRASE <Text>                    {0:1}  g7:PHRASE
           +2 STAT <Enum>                         {0:1}  g7:FAMC-STAT
              +3 PHRASE <Text>                    {0:1}  g7:PHRASE
           +2 <<NOTE_STRUCTURE>>                  {0:M}
        +1 FAMS @<XREF:FAM>@                      {0:M}  g7:FAMS
           +2 <<NOTE_STRUCTURE>>                  {0:M}
        +1 SUBM @<XREF:SUBM>@                     {0:M}  g7:SUBM
        +1 <<ASSOCIATION_STRUCTURE>>              {0:M}
        +1 ALIA @<XREF:INDI>@                     {0:M}  g7:ALIA
           +2 PHRASE <Text>                       {0:1}  g7:PHRASE
        +1 ANCI @<XREF:SUBM>@                     {0:M}  g7:ANCI
        +1 DESI @<XREF:SUBM>@                     {0:M}  g7:DESI
        +1 <<IDENTIFIER_STRUCTURE>>               {0:M}
        +1 <<NOTE_STRUCTURE>>                     {0:M}
        +1 <<SOURCE_CITATION>>                    {0:M}
        +1 <<MULTIMEDIA_LINK>>                    {0:M}
        +1 <<CHANGE_DATE>>                        {0:1}
        +1 <<CREATION_DATE>>                      {0:1}
```

The individual record is a compilation of facts or hypothesized facts about an individual. These facts may come from multiple sources. Source citations and notes allow documentation of the source where each of the facts were discovered.

A single individual may have facts distributed across multiple individual records, connected by `ALIA` (p.74) (alias, in the computing sense not the pseudonym sense) pointers. See `ALIA` for more details.

Individual records are linked to Family records by use of bi-directional pointers. Details about those links are stored as substructures of the pointers in the individual record. Source citations and notes related to the start of the individual's relationship to parents should be placed under the individual's `BIRT`, `CHR` (p.76), or `ADOP` (p.73) event, rather than directly under the `INDI` record, since the former permits explicitly identifying the family record whereas the latter does not.

Other associations or relationships are represented by the ASSO (p.75) (association) tag. The person's relation or associate is the person being pointed to. The association or relationship is stated by the value on the subordinate ROLE (p.94) line. ASSO should not be used for relationships that can be expressed using FAMS (p.82) or FAMC (p.82) instead.

> *Example* — The following example refers to 2 individuals, @I1@ and @I2@, where @I2@ is a godparent of @I1@:
>
> ```
> 0 @I1@ INDI
> 1 ASSO @I2@
> 2 ROLE GODP
> ```

Events stored as facts within an INDI record may also have FAMC or ASSO tags to indicate families and individuals that participated in those events. For example, a FAMC pointer subordinate to an adoption event indicates a relationship to family by adoption; biological parents can be shown by a FAMC pointer subordinate to the birth event; the eulogist at a funeral can be shown by an ASSO pointer subordinate to the burial event; and so on. A subordinate FAMC pointer is allowed to refer to a family where the individual does not appear as a child.

If a FAM that can be reached from a INDI by following FAMS (p.82) and CHIL (p.76) pointers can also be reachable by following FAMC, HUSB (p.83), and WIFE (p.101) pointers, then that implies that a person is their own ancestor/descendant. In most cases that would be an error, though it is theoretically possible that such a situation could occur with non-biological relationships (marriages, adoptions, etc.).

### MULTIMEDIA_RECORD :=

```
n @XREF:OBJE@ OBJE                          {1:1}  g7:record-OBJE
  +1 RESN <List:Enum>                       {0:1}  g7:RESN
  +1 FILE <FilePath>                        {1:M}  g7:FILE
     +2 FORM <MediaType>                    {1:1}  g7:FORM
        +3 MEDI <Enum>                      {0:1}  g7:MEDI
           +4 PHRASE <Text>                 {0:1}  g7:PHRASE
     +2 TITL <Text>                         {0:1}  g7:TITL
     +2 TRAN <FilePath>                     {0:M}  g7:FILE-TRAN
        +3 FORM <MediaType>                 {1:1}  g7:FORM
  +1 <<IDENTIFIER_STRUCTURE>>               {0:M}
  +1 <<NOTE_STRUCTURE>>                     {0:M}
  +1 <<SOURCE_CITATION>>                    {0:M}
  +1 <<CHANGE_DATE>>                        {0:1}
  +1 <<CREATION_DATE>>                      {0:1}
```

The multimedia record refers to 1 or more external digital files, and may provide some additional information about the files and the media they encode.

The file reference can occur more than once to group multiple files together. Grouped files should each pertain to the same context. For example, a sound clip and a photo both of the same event might be grouped in a single `OBJE` (p.89).

The change and creation dates should be for the `OBJE` record itself, not the underlying files.

A `MULTIMEDIA_RECORD` (p.46) may contain a pointer to a `SOURCE_RECORD` (p.49) and vice versa. Applications must not create datasets where these mutual pointers form a cycle. Applications should also ensure they can handle invalid files with such cycles in a safe manner.

### `REPOSITORY_RECORD` :=

```
      n @XREF:REPO@ REPO                          {1:1}  g7:record-REPO
        +1 NAME <Text>                            {1:1}  g7:NAME
        +1 <<ADDRESS_STRUCTURE>>                  {0:1}
        +1 PHON <Special>                         {0:M}  g7:PHON
        +1 EMAIL <Special>                        {0:M}  g7:EMAIL
        +1 FAX <Special>                          {0:M}  g7:FAX
        +1 WWW <Special>                          {0:M}  g7:WWW
        +1 <<NOTE_STRUCTURE>>                     {0:M}
        +1 <<IDENTIFIER_STRUCTURE>>               {0:M}
        +1 <<CHANGE_DATE>>                        {0:1}
        +1 <<CREATION_DATE>>                      {0:1}
```

The repository record provides information about an institution or person that has a collection of sources. Informal repositories include the owner of an unpublished work or of a rare published source, or a keeper of personal collections. An example would be the owner of a family Bible containing unpublished family genealogical entries.

Layered repositories, such as an archive containing copies of a subset of records from another archive or archives that have moved or been bought by other archives, are not modeled in this version of the specification. It is expected they will be added in a later version. Until such time, it is recommended that the repository record store current contact information, if known.

### SHARED_NOTE_RECORD :=

```
n @XREF:SNOTE@ SNOTE <Text>                {1:1}  g7:record-SNOTE
  +1 MIME <MediaType>                      {0:1}  g7:MIME
  +1 LANG <Language>                       {0:1}  g7:LANG
  +1 TRAN <Text>                           {0:M}  g7:NOTE-TRAN
     +2 MIME <MediaType>                   {0:1}  g7:MIME
     +2 LANG <Language>                    {0:1}  g7:LANG
  +1 <<SOURCE_CITATION>>                   {0:M}
  +1 <<IDENTIFIER_STRUCTURE>>              {0:M}
  +1 <<CHANGE_DATE>>                       {0:1}
  +1 <<CREATION_DATE>>                     {0:1}
```

A catch-all location for information that does not fully fit within other structures. It may include research notes, additional context, alternative interpretations, reasoning, and so forth.

A shared note record may be pointed to by multiple other structures. Shared notes should only be used if editing the note in one place should edit it in all other places or if the note itself requires an IDENTIFIER_STRUCTURE (p.55). If each instance of the note may be edited separately and no identifier is needed, a NOTE (p.89) should be used instead.

Each SNOTE.TRAN (p.98) must have either a MIME (p.87) or LANG (p.84) substructure or both.

> *Example* — The origin of a name might be a reasonable shared note, while the reason a particular person was given that name may make more sense as a non-shared note.
>
> ```
> 0 @GORDON@ SNOTE "Gordon" is a traditional Scottish surname.
> 1 CONT It became a given name in honor of Charles George Gordon.
> 0 @I1@ INDI
> 1 NAME Gordon /Jones/
> 2 NOTE Named after the astronaut Gordon Cooper
> 2 SNOTE @GORDON@
> ```

> *Note* — The ability to have multiple structures share a single note using pointers was introduced in version 5.0 in 1991. However, as of 2021 relatively few applications have a user interface that presents shared notes as such to users. It is recommended that SNOTE (p.95) be avoided when NOTE will suffice.

A SHARED_NOTE_RECORD (p.48) may contain a pointer to a SOURCE_RECORD and vice versa. Applications must not create datasets where these mutual pointers form a cycle. Applications should also ensure they can handle invalid files with such cycles in a safe manner.

### SOURCE_RECORD :=

```
n @XREF:SOUR@ SOUR                          {1:1}  g7:record-SOUR
  +1 DATA                                   {0:1}  g7:DATA
     +2 EVEN <List:Enum>                    {0:M}  g7:DATA-EVEN
        +3 DATE <DatePeriod>                {0:1}  g7:DATA-EVEN-DATE
           +4 PHRASE <Text>                 {0:1}  g7:PHRASE
        +3 <<PLACE_STRUCTURE>>              {0:1}
     +2 AGNC <Text>                         {0:1}  g7:AGNC
     +2 <<NOTE_STRUCTURE>>                  {0:M}
  +1 AUTH <Text>                            {0:1}  g7:AUTH
  +1 TITL <Text>                            {0:1}  g7:TITL
  +1 ABBR <Text>                            {0:1}  g7:ABBR
  +1 PUBL <Text>                            {0:1}  g7:PUBL
  +1 TEXT <Text>                            {0:1}  g7:TEXT
     +2 MIME <MediaType>                    {0:1}  g7:MIME
     +2 LANG <Language>                     {0:1}  g7:LANG
  +1 <<SOURCE_REPOSITORY_CITATION>>         {0:M}
  +1 <<IDENTIFIER_STRUCTURE>>               {0:M}
  +1 <<NOTE_STRUCTURE>>                     {0:M}
  +1 <<MULTIMEDIA_LINK>>                    {0:M}
  +1 <<CHANGE_DATE>>                        {0:1}
  +1 <<CREATION_DATE>>                      {0:1}
```

A source record describes an entire source. A source may also point to REPO (p.93)s to describe repositories or archives where the source document may be found. The part of a source relevant to a specific fact, such as a specific page or entry, is indicated in a SOURCE_CITATION (p.65) that points to the source record.

> *Note* — This sourcing model is known to be insufficient for some use cases and may be refined in a future version of this specification.

A SOURCE_RECORD (p.49) may contain a pointer to a SHARED_NOTE_RECORD (p.48) and vice versa. Applications must not create datasets where these mutual pointers form a cycle. Applications should also ensure they can handle invalid files with such cycles in a safe manner.

A SOURCE_RECORD may contain a pointer to a MULTIMEDIA_RECORD (p.46) and vice versa. Applications must not create datasets where these mutual pointers form a cycle. Applications should also ensure they can handle invalid files with such cycles in a safe manner.

**SUBMITTER_RECORD** :=

```
n @XREF:SUBM@ SUBM                          {1:1}  g7:record-SUBM
  +1 NAME <Text>                            {1:1}  g7:NAME
  +1 <<ADDRESS_STRUCTURE>>                  {0:1}
  +1 PHON <Special>                         {0:M}  g7:PHON
  +1 EMAIL <Special>                        {0:M}  g7:EMAIL
  +1 FAX <Special>                          {0:M}  g7:FAX
  +1 WWW <Special>                          {0:M}  g7:WWW
  +1 <<MULTIMEDIA_LINK>>                    {0:M}
  +1 LANG <Language>                        {0:M}  g7:SUBM-LANG
  +1 <<IDENTIFIER_STRUCTURE>>               {0:M}
  +1 <<NOTE_STRUCTURE>>                     {0:M}
  +1 <<CHANGE_DATE>>                        {0:1}
  +1 <<CREATION_DATE>>                      {0:1}
```

The submitter record identifies an individual or organization that contributed information contained in the dataset. All records in the document are assumed to be contributed by the submitter referenced in the HEAD (p.83), unless a SUBM (p.96) structure inside a specific record points at a different submitter record.

### 3.2.3. Substructures

**ADDRESS_STRUCTURE** :=

```
n ADDR <Special>                            {1:1}  g7:ADDR
  +1 ADR1 <Special>                         {0:1}  g7:ADR1
  +1 ADR2 <Special>                         {0:1}  g7:ADR2
  +1 ADR3 <Special>                         {0:1}  g7:ADR3
  +1 CITY <Special>                         {0:1}  g7:CITY
  +1 STAE <Special>                         {0:1}  g7:STAE
  +1 POST <Special>                         {0:1}  g7:POST
  +1 CTRY <Special>                         {0:1}  g7:CTRY
```

A specific building, plot, or location. The payload is the full formatted address as it would appear on a mailing label, including appropriate line breaks (encoded using CONT (p.76) tags). The expected order of address components varies by region; the address should be organized as expected by the addressed region.

Optionally, additional substructures such as STAE (p.95) and CTRY (p.77) are provided to be used by systems that have structured their addresses for indexing and sorting. If the substructures and ADDR (p.73) payload disagree, the ADDR payload shall be taken as correct. Because the regionally-correct order and formatting of address components cannot be determined from the substructures alone, the ADDR payload is required, even if its content appears to be redundant with the substructures.

> *Deprecation Note* — ADR1 (p.73) and ADR2 (p.73) were introduced in version 5.5 (1996) and ADR3 (p.73) in version 5.5.1 (1999), defined as "The first/second/third line of an address." Some applications interpreted ADR1 as "the first line of the *street* address", but most took the spec as-written and treated it as a straight copy of a line of text already available in the ADDR payload.
>
> Duplicating information bloats files and introduces the potential for self-contradiction. ADR1, ADR2, and ADR3 should not be added to new files.

### ASSOCIATION_STRUCTURE :=

```
n ASSO @<XREF:INDI>@                    {1:1}  g7:ASSO
  +1 PHRASE <Text>                      {0:1}  g7:PHRASE
  +1 ROLE <Enum>                        {1:1}  g7:ROLE
     +2 PHRASE <Text>                   {0:1}  g7:PHRASE
  +1 <<NOTE_STRUCTURE>>                 {0:M}
  +1 <<SOURCE_CITATION>>                {0:M}
```

An individual associated with the subject of the superstructure. The nature of the association is indicated in the ROLE (p.94) substructure.

A voidPtr (p.13) and PHRASE (p.90) can be used to describe associations to people not referenced by any INDI record.

> *Example* — The following indicates that "Mr Stockdale" was the individual's teacher and that individual @I2@ was the clergy officiating at their baptism.
>
> ```
> 0 @I1@ INDI
> 1 ASSO @VOID@
> 2 PHRASE Mr Stockdale
> 2 ROLE OTHER
> 3 PHRASE Teacher
> 1 BAPM
> 2 DATE 1930
> 2 ASSO @I2@
> 3 ROLE CLERGY
> ```

### CHANGE_DATE :=

```
n CHAN                                  {1:1}  g7:CHAN
  +1 DATE <DateExact>                   {1:1}  g7:DATE-exact
     +2 TIME <Time>                     {0:1}  g7:TIME
  +1 <<NOTE_STRUCTURE>>                 {0:M}
```

The date of the most recent modification of the superstructure, optionally with notes about that modification.

The `NOTE` (p.89) substructure may describe previous changes as well as the most recent, although only the most recent change is described by the `DATE` substructure.

### `CREATION_DATE` :=

```
n CREA                                  {1:1}  g7:CREA
  +1 DATE <DateExact>                   {1:1}  g7:DATE-exact
     +2 TIME <Time>                     {0:1}  g7:TIME
```

The date of the initial creation of the superstructure. Because this refers to the initial creation, it should not be modified after the structure is created.

### `DATE_VALUE` :=

```
n DATE <DateValue>                      {1:1}  g7:DATE
  +1 TIME <Time>                        {0:1}  g7:TIME
  +1 PHRASE <Text>                      {0:1}  g7:PHRASE
```

A date, optionally with a time and/or a phrase. If there is a `TIME` (p.96), it asserts that the event happened at a specific time on a single day. `TIME` should not be used with `DatePeriod` (p.28) but may be used with other date types.

> *Note* — There is currently no provision for approximate times or time phrases. Time phrases are expected to be added in version 7.1.

### `EVENT_DETAIL` :=

```
n <<DATE_VALUE>>                        {0:1}
n <<PLACE_STRUCTURE>>                   {0:1}
n <<ADDRESS_STRUCTURE>>                 {0:1}
n PHON <Special>                        {0:M}  g7:PHON
n EMAIL <Special>                       {0:M}  g7:EMAIL
n FAX <Special>                         {0:M}  g7:FAX
n WWW <Special>                         {0:M}  g7:WWW
n AGNC <Text>                           {0:1}  g7:AGNC
n RELI <Text>                           {0:1}  g7:RELI
n CAUS <Text>                           {0:1}  g7:CAUS
n RESN <List:Enum>                      {0:1}  g7:RESN
n SDATE <DateValue>                     {0:1}  g7:SDATE
  +1 TIME <Time>                        {0:1}  g7:TIME
  +1 PHRASE <Text>                      {0:1}  g7:PHRASE
n <<ASSOCIATION_STRUCTURE>>             {0:M}
n <<NOTE_STRUCTURE>>                    {0:M}
n <<SOURCE_CITATION>>                   {0:M}
n <<MULTIMEDIA_LINK>>                   {0:M}
n UID <Special>                         {0:M}  g7:UID
```

Substructures that may be shared by most individual and family events and attributes.

Note that many of these substructures are limited to 1 per event. Conflicting event information should be represented by placing them in separate event structures (with appropriate source citations) rather than by placing them under the same enclosing event.

### `FAMILY_ATTRIBUTE_STRUCTURE` :=

```
[
n NCHI <Integer>                          {1:1}  g7:FAM-NCHI
  +1 TYPE <Text>                          {0:1}  g7:TYPE
  +1 <<FAMILY_EVENT_DETAIL>>              {0:1}
|
n RESI <Text>                             {1:1}  g7:FAM-RESI
  +1 TYPE <Text>                          {0:1}  g7:TYPE
  +1 <<FAMILY_EVENT_DETAIL>>              {0:1}
|
n FACT <Text>                             {1:1}  g7:FAM-FACT
  +1 TYPE <Text>                          {1:1}  g7:TYPE
  +1 <<FAMILY_EVENT_DETAIL>>              {0:1}
]
```

Family attributes; see Family Attributes (p.71) for descriptions of each family attribute type.

> *Note* — Family attribute structures vary as follows:
>
> - `FAM.NCHI` (p.88) has an Integer (p.27) payload; others have Text (p.27) payloads
> - `FAM.FACT` (p.81) requires `TYPE` (p.99); it's optional for others

### `FAMILY_EVENT_DETAIL` :=

```
n HUSB                                    {0:1}  g7:HUSB
  +1 AGE <Age>                            {1:1}  g7:AGE
     +2 PHRASE <Text>                     {0:1}  g7:PHRASE
n WIFE                                    {0:1}  g7:WIFE
  +1 AGE <Age>                            {1:1}  g7:AGE
     +2 PHRASE <Text>                     {0:1}  g7:PHRASE
n <<EVENT_DETAIL>>                        {0:1}
```

Substructures shared by most family events and attributes.

### `FAMILY_EVENT_STRUCTURE` :=

```
[
n ANUL [Y|<NULL>]                         {1:1}  g7:ANUL
  +1 TYPE <Text>                          {0:1}  g7:TYPE
```

```
    +1 <<FAMILY_EVENT_DETAIL>>                {0:1}
  |
  n CENS [Y|<NULL>]                           {1:1}  g7:FAM-CENS
    +1 TYPE <Text>                            {0:1}  g7:TYPE
    +1 <<FAMILY_EVENT_DETAIL>>                {0:1}
  |
  n DIV [Y|<NULL>]                            {1:1}  g7:DIV
    +1 TYPE <Text>                            {0:1}  g7:TYPE
    +1 <<FAMILY_EVENT_DETAIL>>                {0:1}
  |
  n DIVF [Y|<NULL>]                           {1:1}  g7:DIVF
    +1 TYPE <Text>                            {0:1}  g7:TYPE
    +1 <<FAMILY_EVENT_DETAIL>>                {0:1}
  |
  n ENGA [Y|<NULL>]                           {1:1}  g7:ENGA
    +1 TYPE <Text>                            {0:1}  g7:TYPE
    +1 <<FAMILY_EVENT_DETAIL>>                {0:1}
  |
  n MARB [Y|<NULL>]                           {1:1}  g7:MARB
    +1 TYPE <Text>                            {0:1}  g7:TYPE
    +1 <<FAMILY_EVENT_DETAIL>>                {0:1}
  |
  n MARC [Y|<NULL>]                           {1:1}  g7:MARC
    +1 TYPE <Text>                            {0:1}  g7:TYPE
    +1 <<FAMILY_EVENT_DETAIL>>                {0:1}
  |
  n MARL [Y|<NULL>]                           {1:1}  g7:MARL
    +1 TYPE <Text>                            {0:1}  g7:TYPE
    +1 <<FAMILY_EVENT_DETAIL>>                {0:1}
  |
  n MARR [Y|<NULL>]                           {1:1}  g7:MARR
    +1 TYPE <Text>                            {0:1}  g7:TYPE
    +1 <<FAMILY_EVENT_DETAIL>>                {0:1}
  |
  n MARS [Y|<NULL>]                           {1:1}  g7:MARS
    +1 TYPE <Text>                            {0:1}  g7:TYPE
    +1 <<FAMILY_EVENT_DETAIL>>                {0:1}
  |
  n EVEN <Text>                               {1:1}  g7:FAM-EVEN
    +1 TYPE <Text>                            {1:1}  g7:TYPE
    +1 <<FAMILY_EVENT_DETAIL>>                {0:1}
  ]
```

Family events; see Family Events (p.69) for descriptions of each family event type.

An event structure may be used to discuss an event even if the event is not known to have occurred. See Events (p.66) for a discussion of how `DATE` (p.78), `PLAC` (p.91), and the optional `Y` payload indicate whether the structure is asserting the event occurred. See the `NON_EVENT_STRUCTURE` (p.61) for how to state an event did not occur.

> *Note* — Family event structures vary as follows:
>
> - `FAM.EVEN` (p.80) has a Text (p.27) payload; others may have a `Y` payload
> - `FAM.EVEN` requires `TYPE` (p.99); it's optional for others

### `IDENTIFIER_STRUCTURE` :=

```
[
n REFN <Special>                          {1:1}  g7:REFN
  +1 TYPE <Text>                          {0:1}  g7:TYPE
|
n UID <Special>                           {1:1}  g7:UID
|
n EXID <Special>                          {1:1}  g7:EXID
  +1 TYPE <URI>                           {0:1}  g7:EXID-TYPE
]
```

> *Deprecation Note* — Having an `EXID` (p.80) without an `EXID.TYPE` (p.100) substructure is deprecated. The meaning of an `EXID` depends on its `EXID.TYPE`. The cardinality of `EXID.TYPE` will be changed to `{1:1}` in version 8.0.

Each of these provides an identifier for a structure or its subject, and each is different in purpose:

- `REFN` (p.92) is a user-generated identifier for a structure.
- `UID` (p.100) is a globally-unique identifier for a structure.
- `EXID` is an identifier maintained by an external authority that applies to the subject of the structure.

### `INDIVIDUAL_ATTRIBUTE_STRUCTURE` :=

```
[
n CAST <Text>                             {1:1}  g7:CAST
  +1 TYPE <Text>                          {0:1}  g7:TYPE
  +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
|
n DSCR <Text>                             {1:1}  g7:DSCR
  +1 TYPE <Text>                          {0:1}  g7:TYPE
  +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
|
```

```
      n EDUC <Text>                              {1:1}  g7:EDUC
        +1 TYPE <Text>                           {0:1}  g7:TYPE
        +1 <<INDIVIDUAL_EVENT_DETAIL>>           {0:1}
      |
      n IDNO <Special>                           {1:1}  g7:IDNO
        +1 TYPE <Text>                           {1:1}  g7:TYPE
        +1 <<INDIVIDUAL_EVENT_DETAIL>>           {0:1}
      |
      n NATI <Text>                              {1:1}  g7:NATI
        +1 TYPE <Text>                           {0:1}  g7:TYPE
        +1 <<INDIVIDUAL_EVENT_DETAIL>>           {0:1}
      |
      n NCHI <Integer>                           {1:1}  g7:INDI-NCHI
        +1 TYPE <Text>                           {0:1}  g7:TYPE
        +1 <<INDIVIDUAL_EVENT_DETAIL>>           {0:1}
      |
      n NMR <Integer>                            {1:1}  g7:NMR
        +1 TYPE <Text>                           {0:1}  g7:TYPE
        +1 <<INDIVIDUAL_EVENT_DETAIL>>           {0:1}
      |
      n OCCU <Text>                              {1:1}  g7:OCCU
        +1 TYPE <Text>                           {0:1}  g7:TYPE
        +1 <<INDIVIDUAL_EVENT_DETAIL>>           {0:1}
      |
      n PROP <Text>                              {1:1}  g7:PROP
        +1 TYPE <Text>                           {0:1}  g7:TYPE
        +1 <<INDIVIDUAL_EVENT_DETAIL>>           {0:1}
      |
      n RELI <Text>                              {1:1}  g7:INDI-RELI
        +1 TYPE <Text>                           {0:1}  g7:TYPE
        +1 <<INDIVIDUAL_EVENT_DETAIL>>           {0:1}
      |
      n RESI <Text>                              {1:1}  g7:INDI-RESI
        +1 TYPE <Text>                           {0:1}  g7:TYPE
        +1 <<INDIVIDUAL_EVENT_DETAIL>>           {0:1}
      |
      n SSN <Special>                            {1:1}  g7:SSN
        +1 TYPE <Text>                           {0:1}  g7:TYPE
        +1 <<INDIVIDUAL_EVENT_DETAIL>>           {0:1}
      |
      n TITL <Text>                              {1:1}  g7:INDI-TITL
        +1 TYPE <Text>                           {0:1}  g7:TYPE
        +1 <<INDIVIDUAL_EVENT_DETAIL>>           {0:1}
      |
      n FACT <Text>                              {1:1}  g7:INDI-FACT
```

```
      +1 TYPE <Text>                          {1:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
    ]
```

Individual attributes; see Individual Attributes (p.70) for descriptions of each individual attribute type..

> *Note* — Individual attribute structures vary as follows:
>
> - `INDI.NCHI` (p.88) and `NMR` (p.88) have Integer (p.27) payloads; `IDNO` (p.84) and `SSN` (p.95) have Special (p.34) payloads; others have Text (p.27) payloads
> - `INDI.FACT` (p.81) and `IDNO` require `TYPE` (p.99); it's optional for others

### `INDIVIDUAL_EVENT_DETAIL` :=

```
    n <<EVENT_DETAIL>>                        {1:1}
    n AGE <Age>                               {0:1}  g7:AGE
      +1 PHRASE <Text>                        {0:1}  g7:PHRASE
```

Substructures shared by most individual events and attributes.

### `INDIVIDUAL_EVENT_STRUCTURE` :=

```
    [
    n ADOP [Y|<NULL>]                         {1:1}  g7:ADOP
      +1 TYPE <Text>                          {0:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
      +1 FAMC @<XREF:FAM>@                    {0:1}  g7:ADOP-FAMC
         +2 ADOP <Enum>                       {0:1}  g7:FAMC-ADOP
            +3 PHRASE <Text>                  {0:1}  g7:PHRASE
    |
    n BAPM [Y|<NULL>]                         {1:1}  g7:BAPM
      +1 TYPE <Text>                          {0:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
    |
    n BARM [Y|<NULL>]                         {1:1}  g7:BARM
      +1 TYPE <Text>                          {0:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
    |
    n BASM [Y|<NULL>]                         {1:1}  g7:BASM
      +1 TYPE <Text>                          {0:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
    |
    n BIRT [Y|<NULL>]                         {1:1}  g7:BIRT
      +1 TYPE <Text>                          {0:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
```

```
      +1 FAMC @<XREF:FAM>@                    {0:1}  g7:FAMC
    |
    n BLES [Y|<NULL>]                         {1:1}  g7:BLES
      +1 TYPE <Text>                          {0:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
    |
    n BURI [Y|<NULL>]                         {1:1}  g7:BURI
      +1 TYPE <Text>                          {0:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
    |
    n CENS [Y|<NULL>]                         {1:1}  g7:INDI-CENS
      +1 TYPE <Text>                          {0:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
    |
    n CHR [Y|<NULL>]                          {1:1}  g7:CHR
      +1 TYPE <Text>                          {0:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
      +1 FAMC @<XREF:FAM>@                    {0:1}  g7:FAMC
    |
    n CHRA [Y|<NULL>]                         {1:1}  g7:CHRA
      +1 TYPE <Text>                          {0:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
    |
    n CONF [Y|<NULL>]                         {1:1}  g7:CONF
      +1 TYPE <Text>                          {0:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
    |
    n CREM [Y|<NULL>]                         {1:1}  g7:CREM
      +1 TYPE <Text>                          {0:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
    |
    n DEAT [Y|<NULL>]                         {1:1}  g7:DEAT
      +1 TYPE <Text>                          {0:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
    |
    n EMIG [Y|<NULL>]                         {1:1}  g7:EMIG
      +1 TYPE <Text>                          {0:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
    |
    n FCOM [Y|<NULL>]                         {1:1}  g7:FCOM
      +1 TYPE <Text>                          {0:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
    |
    n GRAD [Y|<NULL>]                         {1:1}  g7:GRAD
      +1 TYPE <Text>                          {0:1}  g7:TYPE
```

```
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
    |
    n IMMI [Y|<NULL>]                         {1:1}  g7:IMMI
      +1 TYPE <Text>                          {0:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
    |
    n NATU [Y|<NULL>]                         {1:1}  g7:NATU
      +1 TYPE <Text>                          {0:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
    |
    n ORDN [Y|<NULL>]                         {1:1}  g7:ORDN
      +1 TYPE <Text>                          {0:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
    |
    n PROB [Y|<NULL>]                         {1:1}  g7:PROB
      +1 TYPE <Text>                          {0:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
    |
    n RETI [Y|<NULL>]                         {1:1}  g7:RETI
      +1 TYPE <Text>                          {0:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
    |
    n WILL [Y|<NULL>]                         {1:1}  g7:WILL
      +1 TYPE <Text>                          {0:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
    |
    n EVEN <Text>                             {1:1}  g7:INDI-EVEN
      +1 TYPE <Text>                          {1:1}  g7:TYPE
      +1 <<INDIVIDUAL_EVENT_DETAIL>>          {0:1}
    ]
```

Individual events; see Individual Events (p.67) for descriptions of each individual event type.

An event structure may be used to discuss an event even if the event is not known to have occurred. See Events (p.66) for a discussion of how `DATE` (p.78), `PLAC` (p.91), and the optional `Y` payload indicate whether the structure is asserting the event occurred. See the `NON_EVENT_STRUCTURE` (p.61) for how to state an event did not occur.

> *Note* — Individual event structures vary as follows:
>
> - `INDI.EVEN` (p.80) has a Text (p.27) payload; others may have a `Y` payload
> - `INDI.EVEN` requires `TYPE` (p.99); it's optional for others
> - `BIRT` (p.75) and `CHR` (p.76) may have a `FAMC` (p.82) with no substructures; `ADOP` (p.73) may have a `FAMC` with an optional `ADOP` substructure; others may not have a `FAMC` substructure. The `FAMC` substructure can itself have substructures for source citations and notes related to the child's relationship to parents, and is the recommended place to store such information.

### `LDS_INDIVIDUAL_ORDINANCE` :=

```
[
n BAPL                                  {1:1}  g7:BAPL
  +1 <<LDS_ORDINANCE_DETAIL>>           {0:1}
|
n CONL                                  {1:1}  g7:CONL
  +1 <<LDS_ORDINANCE_DETAIL>>           {0:1}
|
n ENDL                                  {1:1}  g7:ENDL
  +1 <<LDS_ORDINANCE_DETAIL>>           {0:1}
|
n INIL                                  {1:1}  g7:INIL
  +1 <<LDS_ORDINANCE_DETAIL>>           {0:1}
|
n SLGC                                  {1:1}  g7:SLGC
  +1 <<LDS_ORDINANCE_DETAIL>>           {0:1}
  +1 FAMC @<XREF:FAM>@                  {1:1}  g7:FAMC
]
```

Ordinances performed by members of The Church of Jesus Christ of Latter-day Saints; see Latter-day Saint Ordinances (p.72) for descriptions of each ordinance type.

### `LDS_ORDINANCE_DETAIL` :=

```
n <<DATE_VALUE>>                        {0:1}
n TEMP <Text>                           {0:1}  g7:TEMP
n <<PLACE_STRUCTURE>>                   {0:1}
n STAT <Enum>                           {0:1}  g7:ord-STAT
  +1 DATE <DateExact>                   {1:1}  g7:DATE-exact
     +2 TIME <Time>                     {0:1}  g7:TIME
n <<NOTE_STRUCTURE>>                    {0:M}
n <<SOURCE_CITATION>>                   {0:M}
```

Dates for these ordinances should be in the default (`GREGORIAN` (p.114)) calendar and be 1830 or later. These ordinances can be performed posthumously by proxy, and the date may reflect that posthumous date.

### `LDS_SPOUSE_SEALING` :=

```
n SLGS                                  {1:1}  g7:SLGS
  +1 <<LDS_ORDINANCE_DETAIL>>           {0:1}
```

Ordinances performed by members of The Church of Jesus Christ of Latter-day Saints; see Latter-day Saint Ordinances (p.72) for descriptions of each ordinance type.

### `MULTIMEDIA_LINK` :=

```
n OBJE @<XREF:OBJE>@                    {1:1}  g7:OBJE
  +1 CROP                               {0:1}  g7:CROP
     +2 TOP <Integer>                   {0:1}  g7:TOP
     +2 LEFT <Integer>                  {0:1}  g7:LEFT
     +2 HEIGHT <Integer>                {0:1}  g7:HEIGHT
     +2 WIDTH <Integer>                 {0:1}  g7:WIDTH
  +1 TITL <Text>                        {0:1}  g7:TITL
```

Links the superstructure to the `MULTIMEDIA_RECORD` (p.46) with the given pointer.

The optional `CROP` (p.77) substructure indicates that a subregion of an image represents or applies to the superstructure.

The optional `TITL` (p.96) substructure supersedes any `OBJE.FILE.TITL` substructures included in the `MULTIMEDIA_RECORD`.

### `NON_EVENT_STRUCTURE` :=

```
n NO <Enum>                             {1:1}  g7:NO
  +1 DATE <DatePeriod>                  {0:1}  g7:NO-DATE
     +2 PHRASE <Text>                   {0:1}  g7:PHRASE
  +1 <<NOTE_STRUCTURE>>                 {0:M}
  +1 <<SOURCE_CITATION>>                {0:M}
```

Indicates that a specific type of event, given in the payload, did not happen within a given date period (or never happened if there is no `DATE` (p.78) substructure).

Substructures may provide discussion about the non-occurrence of the event but must not limit the meaning of what did not occur. No substructure other than `DATE` may restrict the breadth of that negative assertion.

> *Example* — `1 NO MARR` means “no marriage occurred”

*Example —*

```
1 NO MARR
2 DATE TO 24 MAR 1880
```

means “no marriage had occurred as of March 24th, 1880”

### `NOTE_STRUCTURE` :=

```
[
n NOTE <Text>                                {1:1}  g7:NOTE
  +1 MIME <MediaType>                        {0:1}  g7:MIME
  +1 LANG <Language>                         {0:1}  g7:LANG
  +1 TRAN <Text>                             {0:M}  g7:NOTE-TRAN
     +2 MIME <MediaType>                     {0:1}  g7:MIME
     +2 LANG <Language>                      {0:1}  g7:LANG
  +1 <<SOURCE_CITATION>>                     {0:M}
|
n SNOTE @<XREF:SNOTE>@                       {1:1}  g7:SNOTE
]
```

A catch-all location for information that does not fully fit within other structures. It may include research notes, additional context, alternative interpretations, reasoning, and so forth.

Each `NOTE.TRAN` (p.98) must have either a `MIME` (p.87) or `LANG` (p.84) substructure, and may have both.

See `SHARED_NOTE_RECORD` (p.48) for advice on choosing between `NOTE` (p.89) and `SNOTE` (p.95).

A `NOTE_STRUCTURE` (p.62) can contain a `SOURCE_CITATION` (p.65), which in turn can contain a `NOTE_STRUCTURE`, allowing potentially unbounded nesting of structures. Because each dataset is finite, this nesting is also guaranteed to be finite.

### `PERSONAL_NAME_PIECES` :=

```
n NPFX <Text>                                {0:M}  g7:NPFX
n GIVN <Text>                                {0:M}  g7:GIVN
n NICK <Text>                                {0:M}  g7:NICK
n SPFX <Text>                                {0:M}  g7:SPFX
n SURN <Text>                                {0:M}  g7:SURN
n NSFX <Text>                                {0:M}  g7:NSFX
```

Optional isolated name parts; see `PERSONAL_NAME_STRUCTURE` for more details.

*Example* — "Lt. Cmndr. Joseph Allen jr." might be presented as

```
1 NAME Lt. Cmndr. Joseph /Allen/ jr.
2 NPFX Lt. Cmndr.
2 GIVN Joseph
2 SURN Allen
2 NSFX jr.
```

This specification does not define how the meaning of multiple parts with the same tag differs from the meaning of a single part with a concatenated larger payload. However, some applications allow the user to chose whether to combine or split name parts, meaning the tag quantity should be treated as expressing at least a user preference. Even when multiple `SURN` (p.96) tags are used, the `PersonalName` (p.33) data type identifies a single surname substring between its slashes.

### `PERSONAL_NAME_STRUCTURE` :=

```
n NAME <PersonalName>                          {1:1}  g7:INDI-NAME
  +1 TYPE <Enum>                               {0:1}  g7:NAME-TYPE
     +2 PHRASE <Text>                          {0:1}  g7:PHRASE
  +1 <<PERSONAL_NAME_PIECES>>                  {0:1}
  +1 TRAN <PersonalName>                       {0:M}  g7:NAME-TRAN
     +2 LANG <Language>                        {1:1}  g7:LANG
     +2 <<PERSONAL_NAME_PIECES>>               {0:1}
  +1 <<NOTE_STRUCTURE>>                        {0:M}
  +1 <<SOURCE_CITATION>>                       {0:M}
```

Names of individuals are represented in the manner the name is normally spoken, with the family name, surname, or nearest cultural parallel thereunto separated by slashes (U+002F `/`). Based on the dynamic nature or unknown compositions of naming conventions, it is difficult to provide a more detailed name piece structure to handle every case. The `PERSONAL_NAME_PIECES` (p.62) are provided optionally for systems that cannot operate effectively with less structured information. The Personal Name payload shall be seen as the primary name representation, with name pieces as optional auxiliary information; in particular it is recommended that all name parts in `PERSONAL_NAME_PIECES` appear within the `PersonalName` payload in some form, possibly adjusted for gender-specific suffixes or the like. It is permitted for the payload to contain information not present in any name piece substructure.

The name may be translated or transliterated into different languages or scripts using the `TRAN` (p.97) substructure. It is recommended, but not required, that if the name pieces are used, the same pieces are used in each translation and transliteration.

A `TYPE` (p.99) is used to specify the particular variation that this name is. For example; it could indicate that this name is a name taken at immigration or that it could be an 'also known as' name. See `g7:enumset-NAME-TYPE` (p.109) for more details.

> *Note* — Alternative approaches to representing names are being considered for future versions of this specification.

### `PLACE_STRUCTURE` :=

```
n PLAC <List:Text>                          {1:1}  g7:PLAC
  +1 FORM <List:Text>                       {0:1}  g7:PLAC-FORM
  +1 LANG <Language>                        {0:1}  g7:LANG
  +1 TRAN <List:Text>                       {0:M}  g7:PLAC-TRAN
     +2 LANG <Language>                     {1:1}  g7:LANG
  +1 MAP                                    {0:1}  g7:MAP
     +2 LATI <Latitude>                     {1:1}  g7:LATI
     +2 LONG <Longitude>                    {1:1}  g7:LONG
  +1 EXID <Special>                         {0:M}  g7:EXID
     +2 TYPE <URI>                          {0:1}  g7:EXID-TYPE
  +1 <<NOTE_STRUCTURE>>                     {0:M}
```

> *Deprecation Note* — Having an `EXID` (p.80) without an `EXID.TYPE` (p.100) substructure is deprecated. The meaning of an `EXID` depends on its `EXID.TYPE`. The cardinality of `EXID.TYPE` will be changed to `{1:1}` in version 8.0.

A place, which can be represented in several ways:

- The payload contains a comma-separated list of region names, ordered from smallest to largest. The specific meaning of each element is given by the `FORM` (p.82) substructure, or in the `HEAD.PLAC.FORM` (p.82) if there is no `FORM` substructure. If neither `FORM` exists, the meaning of the elements are not defined in this specification beyond being names of jurisdictions of some kind, ordered from smallest to largest.

  > *Note* — Some applications and users have defaulted to assuming a `FORM` of "City, County, State, Country", and some applications even ignore any `FORM` substructures and treat payloads with a smaller number of elements as if they had additional blank elements at the end.

  Elements should be left blank if they are unknown, do not apply to the location, or are too specific for the region in question.

  > *Example* — A record describing births throughout Oneida county could be recorded as
  >
  > ```
  > 0 @S1@ SOUR
  > 1 DATA
  > 2 EVEN BIRT
  > 3 PLAC , Oneida, Idaho, USA
  > 4 FORM City, County, State, Country
  > ```

- The payload may be translated or transliterated into different languages or scripts using the `TRAN` (p.97) substructure. It should use the same `FORM` as the payload.
- Global coordinates may be presented in the `MAP` (p.86) substructure

> *Note* — This specification does not support places where a region name contains a comma. An alternative system for representing locations is likely to be added in a later version.

### `SOURCE_CITATION` :=

```
n SOUR @<XREF:SOUR>@                    {1:1}  g7:SOUR
  +1 PAGE <Text>                        {0:1}  g7:PAGE
  +1 DATA                               {0:1}  g7:SOUR-DATA
     +2 <<DATE_VALUE>>                  {0:1}
     +2 TEXT <Text>                     {0:M}  g7:TEXT
        +3 MIME <MediaType>             {0:1}  g7:MIME
        +3 LANG <Language>              {0:1}  g7:LANG
  +1 EVEN <Enum>                        {0:1}  g7:SOUR-EVEN
     +2 PHRASE <Text>                   {0:1}  g7:PHRASE
     +2 ROLE <Enum>                     {0:1}  g7:ROLE
        +3 PHRASE <Text>                {0:1}  g7:PHRASE
  +1 QUAY <Enum>                        {0:1}  g7:QUAY
  +1 <<MULTIMEDIA_LINK>>                {0:M}
  +1 <<NOTE_STRUCTURE>>                 {0:M}
```

A citation indicating that the pointed-to source record supports the claims made in the superstructure. Substructures provide additional information about how that source applies to the subject of the citation's superstructure:

- `PAGE` (p.89): where in the source the relevant material can be found.
- `DATA` (p.77): the relevant data from the source.
- `EVEN`: what event the relevant material was recording.
- `QUAY` (p.92): an estimation of the reliability of the source in regard to these claims.
- `MULTIMEDIA_LINK` (p.61): digital copies of the cited part of the source

It is recommended that every `SOURCE_CITATION` (p.65) point to a `SOURCE_RECORD` (p.49). However, a `voidPtr` (p.13) can be used with the citation text in a `PAGE` substructure. The `PAGE` is defined to express a "specific location within the information referenced;" with a `voidPtr` there is no information referenced, so the `PAGE` may describe the entire source.

A `SOURCE_CITATION` can contain a `NOTE_STRUCTURE` (p.62), which in turn can contain a `SOURCE_CITATION`, allowing potentially unbounded nesting of structures. Because each dataset is finite, this nesting is also guaranteed to be finite.

**SOURCE_REPOSITORY_CITATION** :=

```
n REPO @<XREF:REPO>@                          {1:1}  g7:REPO
  +1 <<NOTE_STRUCTURE>>                       {0:M}
  +1 CALN <Special>                           {0:M}  g7:CALN
     +2 MEDI <Enum>                           {0:1}  g7:MEDI
        +3 PHRASE <Text>                      {0:1}  g7:PHRASE
```

This structure is used within a source record to point to a name and address record of the holder of the source document. Formal and informal repository name and addresses are stored in the REPOSITORY_RECORD (p.47). More formal repositories, such as the Family History Library, should show a call number of the source at that repository. The call number of that source should be recorded using a CALN (p.75) substructure.

# 3.3. Structure Meaning

## 3.3.1. Events

As a general rule, events are things that happen on a specific date. Use the dateRange (p.28) form to indicate that an event took place at some time between 2 dates. In most cases, a DatePeriod is inappropriate for an event; if the subject of your recording occurred over a period of time, then it is probably not an event, but rather an attribute.

Event structures can be used to record notes about an event without asserting the event actually occurred. An event structure asserts the event did occur if any of the following are true:

- There is a DATE (p.78) substructure

> *Example —*
>
> ```
> 1 DEAT
> 2 DATE 2 OCT 1937
> ```

> *Note* — Version 5.4 (1995) introduced the "event did occur" meaning of event.`DATE`, so it is now well-established in applications and files. However, it is common for users to enter a date range with no end without intending to indicate that the event occurred. For example, pre 7.0 files sometimes used
>
> ```
> 1 NATU
> 2 DATE AFT 1800
> ```
>
> to mean what 7.0 encodes as
>
> ```
> 1 NOT NATU
> 2 DATE TO 1800
> ```
>
> without intending to imply that `NATU` (p.88) ever did actually occur. Because this is a "sometimes used" rather than a "formally means" situation, it is likely that data using 5.x "after meaning not before" *de facto* pattern will be transferred as-is into 7.0 and persist in files for the foreseeable future.

- There is a `PLAC` (p.91) substructure

> *Example* —
>
> ```
> 1 DEAT
> 2 PLAC Cove, Cache, Utah
> ```

- The event has a payload. A special payload `Y` can be used with some event types to indicate that the event is known to have occurred without providing any additional information about it.

> *Example* —
>
> ```
> 1 DEAT Y
> ```

If none of the above are true, the structure should be seen as a place for inconclusive research notes about the possibility of the event. An assertion that an event did not occur should be encoded using the `NO` (p.89) structure.

#### 3.3.1.1. Individual Events

| Tag | Name<br>URI | Description |
|---|---|---|
| `ADOP` | adoption<br>`g7:ADOP` (p.73) | Creation of a legally approved child-parent relationship that does not exist biologically. |
| `BAPM` | baptism<br>`g7:BAPM` (p.75) | Baptism, performed in infancy or later. (See also `BAPL` (p.72) and `CHR`.) |

| Tag | Name<br>URI | Description |
|---|---|---|
| BARM | Bar Mitzvah<br>g7:BARM (p.75) | The ceremonial event held when a Jewish boy reaches age 13. |
| BASM | Bas Mitzvah<br>g7:BASM (p.75) | The ceremonial event held when a Jewish girl reaches age 13, also known as “Bat Mitzvah.” |
| BIRT | birth<br>g7:BIRT (p.75) | Entering into life. |
| BLES | blessing<br>g7:BLES (p.75) | Bestowing divine care or intercession. Sometimes given in connection with a naming ceremony. |
| BURI | depositing remains<br>g7:BURI (p.75) | Depositing the mortal remains of a deceased person. |
| CENS | census<br>g7:INDI-CENS (p.76) | Periodic count of the population for a designated locality, such as a national or state census. |
| CHR | christening<br>g7:CHR (p.76) | Baptism or naming events for a child. |
| CHRA | adult christening<br>g7:CHRA (p.76) | Baptism or naming events for an adult person. |
| CONF | confirmation<br>g7:CONF (p.76) | Conferring full church membership. |
| CREM | cremation<br>g7:CREM (p.77) | The act of reducing a dead body to ashes by fire. |
| DEAT | death<br>g7:DEAT (p.78) | Mortal life terminates. |
| EMIG | emigration<br>g7:EMIG (p.79) | Leaving one’s homeland with the intent of residing elsewhere. |
| FCOM | first communion<br>g7:FCOM (p.82) | The first act of sharing in the Lord’s supper as part of church worship. |
| GRAD | graduation<br>g7:GRAD (p.83) | Awarding educational diplomas or degrees to individuals. |
| IMMI | immigration<br>g7:IMMI (p.84) | Entering into a new locality with the intent of residing there. |
| NATU | naturalization<br>g7:NATU (p.88) | Obtaining citizenship. |

| Tag | Name<br>URI | Description |
|---|---|---|
| ORDN | ordination<br>g7:ORDN (p.89) | Receiving authority to act in religious matters. |
| PROB | probate<br>g7:PROB (p.92) | Judicial determination of the validity of a will. It may indicate several related court activities over several dates. |
| RETI | retirement<br>g7:RETI (p.93) | Exiting an occupational relationship with an employer after a qualifying time period. |
| WILL | will<br>g7:WILL (p.102) | A legal document treated as an event, by which a person disposes of his or her estate. It takes effect after death. The event date is the date the will was signed while the person was alive. (See also PROB) |

In addition, INDI.EVEN (p.80) is a structure for a generic individual event. It must have a TYPE (p.99) substructure to define what kind of event is being provided.

#### 3.3.1.2. Family Events

| Tag | Name<br>URI | Description |
|---|---|---|
| ANUL | annulment<br>g7:ANUL (p.74) | Declaring a marriage void from the beginning (never existed). |
| CENS | census<br>g7:FAM-CENS (p.76) | Periodic count of the population for a designated locality, such as a national or state census. |
| DIV | divorce<br>g7:DIV (p.79) | Dissolving a marriage through civil action. |
| DIVF | divorce filed<br>g7:DIVF (p.79) | Filing for a divorce by a spouse. |
| ENGA | engagement<br>g7:ENGA (p.79) | Recording or announcing an agreement between 2 people to become married. |
| MARB | marriage bann<br>g7:MARB (p.86) | Official public notice given that 2 people intend to marry. |
| MARC | marriage contract<br>g7:MARC (p.86) | Recording a formal agreement of marriage, including the prenuptial agreement in which marriage partners reach agreement about the property rights of 1 or both, securing property to their children. |

| Tag | Name<br>URI | Description |
|---|---|---|
| MARL | marriage license<br>g7:MARL (p.86) | Obtaining a legal license to marry. |
| MARR | marriage<br>g7:MARR (p.86) | A legal, common-law, or customary event such as a wedding or marriage ceremony that joins 2 partners to create or extend a family unit. |
| MARS | marriage settlement<br>g7:MARS (p.86) | Creating an agreement between 2 people contemplating marriage, at which time they agree to release or modify property rights that would otherwise arise from the marriage. |

In addition, FAM.EVEN (p.80) is a structure for a generic family event. It must have a TYPE substructure to define what kind of event is being provided.

## 3.3.2. Attributes

Unlike events, the presence of an attribute is sufficient to assert the attribute applied to the individual, regardless of the attribute's substructures and payload.

### 3.3.2.1. Individual Attributes

| Tag | Name<br>URI | Description |
|---|---|---|
| CAST | caste<br>g7:CAST (p.76) | The name of an individual's rank or status in society which is sometimes based on racial or religious differences, or differences in wealth, inherited rank, profession, or occupation. |
| DSCR | physical description<br>g7:DSCR (p.79) | The physical characteristics of a person. |
| EDUC | education<br>g7:EDUC (p.79) | Indicator of a level of education attained. |
| IDNO | identifying number<br>g7:IDNO (p.84) | A number or other string assigned to identify a person within some significant external system. It must have a TYPE substructure to define what kind of identification number is being provided. |
| NATI | nationality<br>g7:NATI (p.88) | An individual's national heritage or origin, or other folk, house, kindred, lineage, or tribal interest. |

| Tag | Name<br>URI | Description |
|---|---|---|
| NCHI | number of children<br>g7:INDI-NCHI (p.88) | The number of children that this person is known to be the parent of (all marriages). |
| NMR | number of marriages<br>g7:NMR (p.88) | The number of times this person has participated in a family as a spouse or parent. |
| OCCU | occupation<br>g7:OCCU (p.89) | The type of work or profession of an individual. |
| PROP | property<br>g7:PROP (p.92) | Pertaining to possessions such as real estate or other property of interest. |
| RELI | religion<br>g7:INDI-RELI (p.92) | A religious denomination to which a person is affiliated or for which a record applies. |
| RESI | residence<br>g7:INDI-RESI (p.93) | An address or place of residence where an individual resided. |
| SSN | social security number<br>g7:SSN (p.95) | A number assigned by the United States Social Security Administration, used for tax identification purposes. It is a type of IDNO. |
| TITL | title<br>g7:INDI-TITL (p.97) | A formal designation used by an individual in connection with positions of royalty or other social status, such as Grand Duke. |

In addition, INDI.FACT (p.81) is a structure for a generic individual attribute. It must have a TYPE (p.99) substructure to define what kind of attribute is being provided.

#### 3.3.2.2. Family Attributes

| Tag | Name<br>URI | Description |
|---|---|---|
| NCHI | number of children<br>g7:FAM-NCHI (p.88) | The number of children that belong to this family. |
| RESI | residence<br>g7:FAM-RESI (p.93) | An address or place of residence where a family resided. |

In addition, FAM.FACT (p.81) is a structure for a generic family attribute. It must have a TYPE substructure to define what kind of attribute is being provided.

### 3.3.3. Latter-day Saint Ordinances

The structures describing ordinances performed by The Church of Jesus Christ of Latter-day Saints are unlike regular events in that they might either be performed during life or by proxy on the behalf of a deceased individual.

Proxy ordinances on behalf of deceased persons were once requested and officially recorded using an earlier version of GEDCOM. This is no longer the case, but when it was the case the following principles held:

- `PLAC` (p.91) was used only for ordinances that were performed by the recipient in life.
- `TEMP` (p.96) was used with all `ENDL` (p.79), `SLGC` (p.95), and `SLGS` (p.95), but only with posthumous proxy `BAPL` (p.75) and `CONL` (p.76).

| Tag | Name<br>URI | Description |
|---|---|---|
| `BAPL` | baptism<br>`g7:BAPL` | The event of baptism performed at age 8 or later by priesthood authority of The Church of Jesus Christ of Latter-day Saints. (See also `BAPM` (p.67)) |
| `CONL` | confirmation<br>`g7:CONL` | The religious event by which a person receives membership in The Church of Jesus Christ of Latter-day Saints. (See also `CONF` (p.67)) |
| `INIL` | initiatory<br>`g7:INIL` (p. 84) | A religious event where an initiatory ordinance for an individual was performed by priesthood authority in a temple of The Church of Jesus Christ of Latter-day Saints. |
| `ENDL` | endowment<br>`g7:ENDL` | A religious event where an endowment ordinance for an individual was performed by priesthood authority in a temple of The Church of Jesus Christ of Latter-day Saints. |
| `SLGC` | sealing child<br>`g7:SLGC` | A religious event pertaining to the sealing of a child to his or her parents in a temple ceremony of The Church of Jesus Christ of Latter-day Saints. |
| `SLGS` | sealing spouse<br>`g7:SLGS` | A religious event pertaining to the sealing of a husband and wife in a temple ceremony of The Church of Jesus Christ of Latter-day Saints. (See also `MARR` (p.69)) |

### 3.3.4. Structure types

Structure types are listed in this section alphabetically by tag. When the same tag is used for different structure types in different contexts, they may be distinguished by their URI.

#### `ABBR` (Abbreviation) `<g7:ABBR>`

A short name of a title, description, or name used for sorting, filing, and retrieving records.

#### `ADDR` (Address) `<g7:ADDR>`

The location of, or most relevant to, the subject of the superstructure. See `ADDRESS_STRUCTURE` (p.50) for more details.

#### `ADOP` (Adoption) `<g7:ADOP>`

An Individual Event (p.67). See also `INDIVIDUAL_EVENT_STRUCTURE` (p.57).

#### `ADOP` (Adoption) `<g7:FAMC-ADOP>`

An enumerated value from set `g7:enumset-ADOP` (p.102) indicating which parent(s) in the family adopted this individual.

#### `ADR1` (Address Line 1) `<g7:ADR1>`

The first line of the address, used for indexing. This structure's payload should be a single line of text equal to the first line of the corresponding `ADDR`. See `ADDRESS_STRUCTURE` for more details.

> *Deprecation Note* — `ADR1` should not be added to new files; see `ADDRESS_STRUCTURE` for more details.

#### `ADR2` (Address Line 2) `<g7:ADR2>`

The second line of the address, used for indexing. This structure's payload should be a single line of text equal to the second line of the corresponding `ADDR`. See `ADDRESS_STRUCTURE` for more details.

> *Deprecation Note* — `ADR2` should not be added to new files; see `ADDRESS_STRUCTURE` for more details.

#### `ADR3` (Address Line 3) `<g7:ADR3>`

The third line of the address, used for indexing. This structure's payload should be a single line of text equal to the third line of the corresponding `ADDR`. See `ADDRESS_STRUCTURE` for more details.

> *Deprecation Note* — `ADR3` should not be added to new files; see `ADDRESS_STRUCTURE` for more details.

### `AGE` (Age at event) `<g7:AGE>`

The age of the individual at the time an event occurred. It is recommended that this be an age from a cited source document.

### `AGNC` (Responsible agency) `<g7:AGNC>`

The organization, institution, corporation, person, or other entity that has responsibility for the associated context. Examples are an employer of a person of an associated occupation, or a church that administered rites or events, or an organization responsible for creating or archiving records.

### `ALIA` (Alias) `<g7:ALIA>`

A single individual may have facts distributed across multiple individual records, connected by `ALIA` pointers (named after "alias" in the computing sense, not the pseudonym sense).

An `ALIA` pointer should not point to the superstructure of the `ALIA`.

> *Note* — This specification does not define how to connect `INDI` records with `ALIA`. Some systems organize `ALIA` pointers to create a tree structure, with the root `INDI` record containing the composite view of all facts in the leaf `INDI` records. Others distribute events and attributes between `INDI` records mutually linked by symmetric pairs of `ALIA` pointers.
>
> `ALIA` is known to be used for different purposes by different users. Some users use `ALIA` for uncertain connections, tentatively linking records prior to confirming identities and merging them into a single record; other users create one `INDI` for each single-source view of an individual, linked together with `ALIA` and never merged into a single record; other uses of `ALIA` may also exist. Applications should avoid assuming a particular usage was intended without user confirmation.
>
> A future version of this specification may adjust the definition of `ALIA`.

### `ANCI` (Ancestor interest) `<g7:ANCI>`

Indicates an interest in additional research for ancestors of this individual. (See also `DESI` (p.79)).

### `ANUL` (Annulment) `<g7:ANUL>`

A Family Event (p.69). See also `FAMILY_EVENT_STRUCTURE` (p.53).

### `ASSO` (Associates) `<g7:ASSO>`

A pointer to an associated individual. See `ASSOCIATION_STRUCTURE` (p.51) for more details.

### `AUTH` (Author) `<g7:AUTH>`

The person, agency, or entity who created the record. For a published work, this could be the author, compiler, transcriber, abstractor, or editor. For an unpublished source, this may be an individual, a government agency, church organization, or private organization.

### `BAPL` (Baptism, Latter-Day Saint) `<g7:BAPL>`

A Latter-Day Saint Ordinance (p.72). See also `LDS_INDIVIDUAL_ORDINANCE` (p.60).

### `BAPM` (Baptism) `<g7:BAPM>`

An Individual Event (p.67). See also `INDIVIDUAL_EVENT_STRUCTURE` (p.57).

### `BARM` (Bar Mitzvah) `<g7:BARM>`

An Individual Event (p.67). See also `INDIVIDUAL_EVENT_STRUCTURE`.

### `BASM` (Bas Mitzvah) `<g7:BASM>`

An Individual Event (p.67). See also `INDIVIDUAL_EVENT_STRUCTURE`.

### `BIRT` (Birth) `<g7:BIRT>`

An Individual Event (p.67). See also `INDIVIDUAL_EVENT_STRUCTURE`.

### `BLES` (Blessing) `<g7:BLES>`

An Individual Event (p.67). See also `INDIVIDUAL_EVENT_STRUCTURE`.

### `BURI` (Depositing remains) `<g7:BURI>`

An Individual Event (p.67). See also `INDIVIDUAL_EVENT_STRUCTURE`.

Although defined as any depositing of remains since it was introduced in the first version of GEDCOM, this tag is a shortened form of the English word “burial” and has been interpreted to mean “depositing of remains by burial” by some applications and users. In the absence of a clarifying `TYPE` (p.99) substructure it is likely, but not guaranteed, that a `BURI` structure refers to a burial rather than another form of depositing remains.

### `CALN` (Call number) `<g7:CALN>`

An identification or reference description used to file and retrieve items from the holdings of a repository. Despite the word “number” in the name, may contain any character, not just digits.

### `CAST` (Caste) `<g7:CAST>`

An Individual Attribute (p.70). See also `INDIVIDUAL_ATTRIBUTE_STRUCTURE` (p.55).

### `CAUS` (Cause) `<g7:CAUS>`

The reasons which precipitated an event. It is often used subordinate to a death event to show cause of death, such as might be listed on a death certificate.

### `CENS` (Census) `<g7:FAM-CENS>`

An Family Event (p.69).

### `CENS` (Census) `<g7:INDI-CENS>`

An Individual Event (p.67). See also `INDIVIDUAL_EVENT_STRUCTURE` (p.57).

### `CHAN` (Change) `<g7:CHAN>`

The most recent change to the superstructure. This is metadata about the structure itself, not data about its subject. See `CHANGE_DATE` (p.51) for more details.

### `CHIL` (Child) `<g7:CHIL>`

The child in a family, whether biological, adopted, foster, sealed, or other relationship.

### `CHRA` (Christening, adult) `<g7:CHRA>`

An Individual Event (p.67). See also `INDIVIDUAL_EVENT_STRUCTURE`.

### `CHR` (Christening) `<g7:CHR>`

An Individual Event (p.67). See also `INDIVIDUAL_EVENT_STRUCTURE`.

### `CITY` (City) `<g7:CITY>`

The name of the city used in the address. See `ADDRESS_STRUCTURE` (p.50) for more details.

### `CONF` (Confirmation) `<g7:CONF>`

An Individual Event (p.67). See also `INDIVIDUAL_EVENT_STRUCTURE`.

### `CONL` (Confirmation, Latter-Day Saint) `<g7:CONL>`

A Latter-Day Saint Ordinance (p.72). See also `LDS_INDIVIDUAL_ORDINANCE` (p.60).

### `CONT` (Continued) `<g7:CONT>`

A pseudo-structure to indicate a line break. The `CONT` tag is generated during serialization and is never present in parsed datasets. See Lines (p.13) for more details.

### COPR (Copyright) <g7:COPR>

A copyright statement, as appropriate for the copyright laws applicable to this data.

### CORP (Corporate name) <g7:CORP>

The name of the business, corporation, or person that produced or commissioned the product.

### CREA (Creation) <g7:CREA>

The initial creation of the superstructure. This is metadata about the structure itself, not data about its subject. See CREATION_DATE (p.52) for more details.

### CREM (Cremation) <g7:CREM>

An Individual Event (p.67). See also INDIVIDUAL_EVENT_STRUCTURE (p.57).

### CROP (Crop) <g7:CROP>

A subregion of an image to display. It is only valid when the superstructure links to a MULTIMEDIA_RECORD (p.46) with at least 1 FILE (p.82) substructure that refers to an external file with a defined pixel unit.

LEFT (p.86) and TOP (p.97) indicate the top-left corner of the region to display. WIDTH (p.101) and HEIGHT (p.83) indicate how many pixels wide and tall the region to display is. If omitted, LEFT and TOP each default to 0; WIDTH defaults to the image width minus LEFT; and HEIGHT defaults to the image height minus TOP.

If the superstructure links to a MULTIMEDIA_RECORD that includes multiple FILE substructures, the CROP applies to the first FILE to which it can apply, namely the first external file with a defined pixel unit.

It is recommended that CROP be used only with a single-FILE MULTIMEDIA_RECORD.

The following are errors:

- LEFT or LEFT + WIDTH exceed the image width.
- TOP or TOP + HEIGHT exceed the image height.
- CROP applied to a non-image or image without a defined pixel unit.

### CTRY (Country) <g7:CTRY>

The name of the country that pertains to the associated address. See ADDRESS_STRUCTURE (p.50) for more details.

### DATA (Data) <g7:DATA>

A structure with no payload used to distinguish a description of something from metadata about it. For example, SOUR (p.95) and its other substructures describe a source itself, while SOUR.DATA describes the content of the source.

### `DATA` (Data) `<g7:SOUR-DATA>`

See `g7:DATA`.

### `DATA` (Data) `<g7:HEAD-SOUR-DATA>`

The database, electronic data source, or digital repository from which this dataset was exported. The payload is the name of the database, electronic data source, or digital repository, with substructures providing additional details about it (not about the export).

### `DATE` (Date) `<g7:DATE>`

The principal date of the subject of the superstructure. The payload is a `DateValue` (p. 28).

When the superstructure is an event, the principal date indicates when the event took place.

When the superstructure is an attribute, the principal date indicates when the attribute was observed, asserted, or applied. A date period might put bounds on the attributes applicability, but other date forms assume that the attribute may have also applied on other dates too.

When the superstructure is a `g7:SOUR-DATA` (p.78), the principal date indicates when the data was entered into the source; or, for a source like a website that changes over time, a date on which the source contained the data.

See `DATE_VALUE` (p.52) for more details.

### `DATE` (Date) `<g7:DATE-exact>`

The principal date of the subject of the superstructure. The payload is a `DateExact`.

### `DATE` (Date) `<g7:HEAD-DATE>`

The `DateExact` that this document was created.

### `DATE` (Date) `<g7:NO-DATE>`

The `DatePeriod` during which the event did not occur or the attribute did not apply.

### `DATE` (Date) `<g7:DATA-EVEN-DATE>`

The `DatePeriod` covered by the entire source; the period during which this source recorded events.

### `DEAT` (Death) `<g7:DEAT>`

An Individual Event (p.67). See also `INDIVIDUAL_EVENT_STRUCTURE` (p.57).

### `DESI` (Descendant Interest) `<g7:DESI>`

Indicates an interest in research to identify additional descendants of this individual. See also `ANCI` (p.74).

### `DEST` (Destination) `<g7:DEST>`

An identifier for the system expected to receive this document. See `HEAD.SOUR` (p.95) for guidance on choosing identifiers.

### `DIVF` (Divorce filing) `<g7:DIVF>`

A Family Event (p.69). See also `FAMILY_EVENT_STRUCTURE` (p.53).

### `DIV` (Divorce) `<g7:DIV>`

A Family Event (p.69). See also `FAMILY_EVENT_STRUCTURE`.

### `DSCR` (Description) `<g7:DSCR>`

An Individual Attribute (p.70). See also `INDIVIDUAL_ATTRIBUTE_STRUCTURE` (p.55).

### `EDUC` (Education) `<g7:EDUC>`

An Individual Attribute (p.70). See also `INDIVIDUAL_ATTRIBUTE_STRUCTURE`.

### `EMAIL` (Email) `<g7:EMAIL>`

An electronic mail address, as defined by any relevant standard such as RFC 3696, RFC 5321, or RFC 5322.

If an invalid email address is present upon import, it should be preserved as-is on export.

> *Note* — The version 5.5.1 specification contained a typo where this tag was sometimes written `EMAI` and sometimes written `EMAIL`. `EMAIL` should be used in version 7.0 and later.

### `EMIG` (Emigration) `<g7:EMIG>`

An Individual Event (p.67). See also `INDIVIDUAL_EVENT_STRUCTURE` (p.57).

### `ENDL` (Endowment, Latter-Day Saint) `<g7:ENDL>`

A Latter-Day Saint Ordinance (p.72). See also `LDS_INDIVIDUAL_ORDINANCE` (p.60).

### `ENGA` (Engagement) `<g7:ENGA>`

A Family Event (p.69). See also `FAMILY_EVENT_STRUCTURE` (p.53).

### `EVEN` (Event) `<g7:FAM-EVEN>`

See `g7:INDI-EVEN`.

### `EVEN` (Event) `<g7:INDI-EVEN>`

An event: a noteworthy happening related to an individual or family. If a specific event type exists, it should be used instead of a generic `EVEN` structure. Each `EVEN` must be classified by a subordinate use of the `TYPE` (p.99) tag and may be further described in the structure's payload.

> *Example* — A person that signed a lease for land dated October 2, 1837 and a lease for mining equipment dated November 4, 1837 would be written as:
>
> ```
> 0 @I1@ INDI
> 1 EVEN
> 2 TYPE Land Lease
> 2 DATE 2 OCT 1837
> 1 EVEN Mining equipment
> 2 TYPE Equipment Lease
> 2 DATE 4 NOV 1837
> ```

### `EVEN` (Event) `<g7:DATA-EVEN>`

A list of enumerated values from set `g7:enumset-EVENATTR` (p.103) indicating the types of events that were recorded in a particular source. Each event type is separated by a comma and space. For example, a parish register of births, deaths, and marriages would be `BIRT, DEAT, MARR`.

### `EVEN` (Event) `<g7:SOUR-EVEN>`

An enumerated value from set `g7:enumset-EVENATTR` indicating the type of event or attribute which was responsible for the source entry being recorded. For example, if the entry was created to record a birth of a child, then the type would be `BIRT` (p.75) regardless of the assertions made from that record, such as the mother's name or mother's birth date.

### `EXID` (External Identifier) `<g7:EXID>`

An identifier for the subject of the superstructure. The identifier is maintained by some external authority; the authority owning the identifier is provided in the TYPE substructure; see `EXID.TYPE` (p.100) for more details.

Depending on the maintaining authority, an `EXID` may be a unique identifier for the subject, an identifier for 1 of several views of the subject, or an identifier for the externally-maintained copy of the same information as is contained in this structure. Howev-

er, unlike UID (p.100) and REFN (p.92), EXID does not identify a structure; structures with the same EXID may have originated independently rather than by edits from the same starting point.

EXID identifiers are expected to be unique. Once assigned, an EXID identifier should never be re-used for any other purpose.

### FAM (Family record) <g7:record-FAM>

See FAMILY_RECORD (p.43)

> *Note* — The common case is that each couple has one FAM record, but that is not always the case.
>
> A couple that separates and then gets together again can be represented either as a single FAM with multiple events (MARR (p.86), DIV (p.79), etc.) or as a separate FAM for each time together. Some user interfaces may display these two in different ways and the two admit different semantics in sourcing. A single FAM with two MARR with distinct dates might also represent uncertainty about dates and a pair of FAM with same spouses might also be the result of merging multiple files.
>
> Implementers should support both representations, and should choose between them based on user input or other context beyond that provided in the datasets themselves.

### FACT (Fact) <g7:FAM-FACT>

See g7:INDI-FACT.

### FACT (Fact) <g7:INDI-FACT>

A noteworthy attribute or fact concerning an individual or family. If a specific attribute type exists, it should be used instead of a generic FACT structure. Each FACT must be classified by a subordinate use of the TYPE (p.99) tag and may be further described in the structure's payload.

> *Example* — If the attribute being defined was 1 of the person's skills, such as woodworking, the FACT tag would have the value of "Woodworking", followed by a subordinate TYPE tag with the value "Skills".

```
0 @I1@ INDI
1 FACT Woodworking
2 TYPE Skills
```

### FAMC (Family child) <g7:INDI-FAMC>

The family in which an individual appears as a child. It is also used with a g7:FAMC-STAT (p.96) substructure to show individuals who are not children of the family. See FAMILY_RECORD (p.43) for more details.

### `FAMC` (Family child) `<g7:FAMC>`

The family with which this individual event is associated.

### `FAMC` (Family child) `<g7:ADOP-FAMC>`

The individual or couple that adopted this individual.

Adoption by an individual, rather than a couple, may be represented either by pointing to a `FAM` where that individual is a `HUSB` (p.83) or `WIFE` (p.101) and using a `g7:FAMC-ADOP` (p.73) substructure to indicate which 1 performed the adoption; or by using a `FAM` where the adopting individual is the only `HUSB`/`WIFE`.

### `FAMS` (Family spouse) `<g7:FAMS>`

The family in which an individual appears as a partner. See `FAMILY_RECORD` for more details.

### `FAX` (Facsimile) `<g7:FAX>`

A fax telephone number appropriate for sending data facsimiles. See `PHON` (p.90) for additional comments on telephone numbers.

### `FCOM` (First communion) `<g7:FCOM>`

An Individual Event (p.67). See also `INDIVIDUAL_EVENT_STRUCTURE` (p.57).

### `FILE` (File reference) `<g7:FILE>`

A reference to an external file. See the File Path datatype (p.35) for more details.

### `FORM` (Format) `<g7:FORM>`

The media type (p.34) of the file referenced by the superstructure.

### `FORM` (Format) `<g7:PLAC-FORM>`

A comma-separated list of jurisdictional titles, which has the same number of elements and in the same order as the `PLAC` (p.91) structure. As with `PLAC`, this shall be ordered from lowest to highest jurisdiction.

> *Example* — The following represents Baltimore, a city that is not within a county.
>
> ```
> 2 PLAC Baltimore, , Maryland, USA
> 3 FORM City, County, State, Country
> ```

### `FORM` (Format) `<g7:HEAD-PLAC-FORM>`

Any `PLAC` with no `FORM` (p.82) shall be treated as if it has this `FORM` (p.82).

### `GEDC` (GEDCOM) `<g7:GEDC>`

A container for information about the entire document.

It is recommended that applications write `GEDC` with its required substructure `g7:GEDC-VERS` (p.101) as the first substructure of `HEAD`.

### `GIVN` (Given name) `<g7:GIVN>`

A given or earned name used for official identification of a person.

### `GRAD` (Graduation) `<g7:GRAD>`

An Individual Event (p.67). See also `INDIVIDUAL_EVENT_STRUCTURE` (p.57).

### `HEAD` (Header) `<g7:HEAD>`

A pseudo-structure for storing metadata about the document. See The Header and Trailer (p.17) for more details.

### `HEIGHT` (Height in pixels) `<g7:HEIGHT>`

How many pixels to display vertically for the image. See `CROP` (p.77) for more details.

> *Note* — `HEIGHT` is a number of pixels. The correct tag for the height of an individual is the `DSCR` (p.79) attribute.
>
> > *Example* —
> >
> > ```
> > 0 @I45@ INDI
> > 1 DSCR brown eyes, 5ft 10in, 198 pounds
> > ```

### `HUSB` (Husband) `<g7:HUSB>`

A structure for storing information related to one partner in the context of a `FAMILY_EVENT`; in particular, the partner referenced in the `g7:FAM-HUSB` substructure of the `g7:record-FAM` (p.81) superstructure of the `FAMILY_EVENT`.

> *Example* — The following indicates that individual `@I1@` was 32 years old at the time of the marriage, without indicating an age for individual `@I2@`.
>
> ```
> 0 @F1@ FAM
> 1 HUSB @I1@
> 1 WIFE @I2@
> 1 MARR
> 2 HUSB
> 3 AGE 32y
> ```

### `HUSB` (Husband) `<g7:FAM-HUSB>`

This is a partner in a `FAM` record. See `FAMILY_RECORD` (p.43) for more details.

### `IDNO` (Identification number) `<g7:IDNO>`

An Individual Attribute (p.70). See also `INDIVIDUAL_ATTRIBUTE_STRUCTURE`.

### `IMMI` (Immigration) `<g7:IMMI>`

An Individual Event (p.67). See also `INDIVIDUAL_EVENT_STRUCTURE` (p.57).

### `INDI` (Individual) `<g7:record-INDI>`

See `INDIVIDUAL_RECORD` (p.45).

### `INIL` (Initiatory, Latter-Day Saint) `<g7:INIL>`

A Latter-Day Saint Ordinance (p.72). See also `LDS_INDIVIDUAL_ORDINANCE` (p.60). Previously, GEDCOM versions 3.0 through 5.3 called this `WAC`; it was not part of 5.4 through 5.5.1. FamilySearch GEDCOM 7.0 reintroduced it with the name `INIL` for consistency with `BAPL` (p.75), `CONL` (p.76), and `ENDL` (p.79).

### `LANG` (Language) `<g7:LANG>`

The primary human language of the superstructure. The primary language in which the `Text` (p.27)-typed payloads of the superstructure and its substructures appear.

The payload of the `LANG` structure is a language tag, as defined by BCP 47. A registry of component subtags is maintained publicly by the IANA.

In the absence of a `LANG` structure, the language is assumed to be unspecified; that may also be recorded explicitly with language tag `und` (meaning “undetermined”). See `g7:HEAD-LANG` (p.85) for information about applying language-specific algorithms to text in an unspecified language.

If the text is primarily in one language with a few parts in a different language, it is recommended that a language tag identifying the primary language be used. If no one language is primary, the language tag `mul` (meaning “multiple”) may be used, but most language-specific algorithms will treat `mul` the same way they do `und`.

> *Note* — Conversations are ongoing about adding part-of-payload language tagging in a future version of the specification to provide more fidelity for multilingual text.

If the text is not in any human language and should not be treated as lingual content, the language tag `zxx` (meaning “no linguistic content” or “not applicable”) may be used. An example of `zxx` text might be a diagram approximated using characters for their shape, not their meaning.

> *Note* — This specification does not permit `LANG` in every place where human language text might appear. Conversations are ongoing about adding it in more places in a future version of the specification. Using the current specification, additional language tagging can be accomplished using a documented extension tag (p.21) by including the following in the header:
>
> ```
> 1 SCHEMA
> 2 TAG _LANG https://gedcom.io/terms/v7/LANG
> ```
>
> and using the extension tag like so:
>
> ```
> 2 DATE 31 AUG 2018
> 3 PHRASE 2018年8月31日
> 4 _LANG cmn
> ```

### `LANG` (Language) `<g7:HEAD-LANG>`

A default language which may be used to interpret any `Text` (p.27)-typed payloads that lack a specific language tag from a `g7:LANG` (p.84) structure. An application may choose to use a different default based on its knowledge of the language preferences of the user.

The payload of the `LANG` structure is a language tag, as defined by BCP 47.

> *Note* — Some algorithms on text are language-specific. Examples include sorting sequences, name comparison and phonetic name matching algorithms, spell-checking, computer-synthesized speech, Braille transcription, and language translation. When the language of the text is given through a `g7:LANG`, that should be used. When `g7:LANG` is not available, `g7:HEAD-LANG` provides the file creator's suggested default language. For some language-specific algorithms, the user's preferred language may be a more appropriate default than the file's default language. User language preferences can be found in a variety of platform-specific places, such as the default language from operating system settings, user locales, Input Method Editors (IMEs), etc.

### `LANG` (Language) `<g7:SUBM-LANG>`

A language the subject of that record understands.

The payload of the `LANG` structure is a language tag, as defined by BCP 47.

### `LATI` (Latitude) `<g7:LATI>`

A latitudinal coordinate.

### `LEFT` (Left crop width) `<g7:LEFT>`

Left is a number of pixels to not display from the left side of the image. See `CROP` (p.77) for more details.

### `LONG` (Longitude) `<g7:LONG>`

A longitudinal coordinate.

### `MAP` (Map) `<g7:MAP>`

A representative point for a location, as defined by `LATI` and `LONG` substructures.

Note that `MAP` provides neither a notion of accuracy (for example, the `MAP` for a birth event may be some distance from the point where the birth occurred) nor a notion of region size (for example, the `MAP` for a place "Belarus" may be anywhere within that nation's 200,000 square kilometer area).

### `MARB` (Marriage banns) `<g7:MARB>`

A Family Event (p.69). See also `FAMILY_EVENT_STRUCTURE` (p.53).

### `MARC` (Marriage contract) `<g7:MARC>`

A Family Event (p.69). See also `FAMILY_EVENT_STRUCTURE`.

### `MARL` (Marriage license) `<g7:MARL>`

A Family Event (p.69). See also `FAMILY_EVENT_STRUCTURE`.

### `MARR` (Marriage) `<g7:MARR>`

A Family Event (p.69). See also `FAMILY_EVENT_STRUCTURE`.

### `MARS` (Marriage settlement) `<g7:MARS>`

A Family Event (p.69). See also `FAMILY_EVENT_STRUCTURE`.

### `MEDI` (Medium) `<g7:MEDI>`

An enumerated value from set `g7:enumset-MEDI` (p.103) providing information about the media or the medium in which information is stored.

When `MEDI` is a substructure of a `g7:CALN` (p.75), it is recommended that its payload describes the medium directly found at that call number rather than a medium from which it was derived.

> *Example* — Consider an asset in a repository that is a digital scan of a book of compiled newspapers; for this asset, the `CALN.MEDI` is recommended to be `ELECTRONIC` rather than `BOOK` or `NEWSPAPER`.

When `MEDI` is a substructure of a `g7:FORM` (p.82), it is recommended that its payload describes the medium from which it was derived.

> *Example* — Consider a digital photo in a multimedia record; for this asset, the `FORM.MEDI` is recommended to be `PHOTO` rather than `ELECTRONIC`.

### `MIME` (Media type) `<g7:MIME>`

Indicates the media type (p.34) of the payload of the superstructure.

As of version 7.0, there are two standard media types for this structure:

- `text/plain` shall be presented to the user as-is, preserving all spacing, line breaks, and so forth.
- `text/html` uses HTML tags to provide presentation information. Applications should support at least the following:
  - `p` and `br` elements for paragraphing and line breaks.
  - `b`, `i`, `u`, and `s` elements for bold, italic, underlined, and strike-through text (or corresponding display in other locales; see HTML §4.5 for more).
  - `sup` and `sub` elements for super- and sub-script.
  - The 3 XML entities that appear in text: `&`, `<` `>`. Note that `&quote;` and `'` are only needed in attributes. Other entities should be represented as their respective Unicode characters instead.

  Supporting more of HTML is encouraged. Unsupported tags should be ignored during display.

> *Note* — Applications are welcome to support more XML entities or HTML character references in their user interface. However, exporting must only use the core XML entities, translating any other entities into their corresponding Unicode characters.

> *Note* — Applications are welcome to support additional HTML elements, but they should ensure that content is meaningful if those extra elements are ignored and only their content text is displayed.

> *Note* — Media types are also used by external files, as described under `FORM` (p.82). External file media types are not limited to `text/plain` and `text/html`.

If needed, `text/html` can be converted to `text/plain` using the following steps:

1. Replace any sequence of 1 or more spaces, tabs, and line breaks with a single space
2. Case-insensitively replace each `<p…>`, `</p…>`, and `<br…>` with a line break
3. Remove all other `<…>` tags
4. Replace each `<` with `<` and `>` with `>`
5. Replace each `&` with `&`

Other `text` media types not discussed above are also permitted, though not recommended. If present, they are considered extensions. Such extensions do not require an extension tag (p.18) because the definition of `g7:MIME` (p.87) is sufficient to cover this kind of extension.

### `NAME` (Name) `<g7:NAME>`

The name of the superstructure's subject, represented as a simple string.

### `NAME` (Name) `<g7:INDI-NAME>`

A `PERSONAL_NAME_STRUCTURE` (p.63) with parts, translations, sources, and so forth.

### `NATI` (Nationality) `<g7:NATI>`

An Individual Attribute (p.70). See also `INDIVIDUAL_ATTRIBUTE_STRUCTURE` (p.55).

### `NATU` (Naturalization) `<g7:NATU>`

An Individual Event (p.67). See also `INDIVIDUAL_EVENT_STRUCTURE` (p.57).

### `NCHI` (Number of children) `<g7:FAM-NCHI>`

A Family Attribute (p.71). See also `FAMILY_ATTRIBUTE_STRUCTURE` (p.53).

### `NCHI` (Number of children) `<g7:INDI-NCHI>`

An Individual Attribute (p.70). See also `INDIVIDUAL_ATTRIBUTE_STRUCTURE`.

### `NICK` (Nickname) `<g7:NICK>`

A descriptive or familiar name that is used instead of, or in addition to, one's official or legal name.

> *Note* — The label "nickname" and description text of this structure were introduced with version 5.5 in 1996, but are understood differently by different users. Some use `NICK` only for names that would be inappropriate in formal settings. Some use it for pseudonyms regardless of where they are used. Some use it for any variant of a name that is not the one used on legal documents. Because all of these uses, and likely others as well, are common in existing data, no further clarification of the meaning of the `NICK` structure is possible without contradicting some existing data.

### `NMR` (Number of marriages) `<g7:NMR>`

An Individual Attribute (p.70). See also `INDIVIDUAL_ATTRIBUTE_STRUCTURE`.

### `NO` (Did not happen) `<g7:NO>`

An enumerated value from set `g7:enumset-EVEN` (p.102) identifying an event type which did not occur to the superstructure's subject. A specific payload `NO XYZ` should only appear where `XYZ` would be legal.

See `NON_EVENT_STRUCTURE` for more details.

### `NOTE` (Note) `<g7:NOTE>`

A `NOTE_STRUCTURE` (p.62), containing additional information provided by the submitter for understanding the enclosing data.

When a substructure of `HEAD` (p.83), it should describe the contents of the document in terms of "ancestors or descendants of" so that the person receiving the data knows what genealogical information the document contains.

### `NPFX` (Name prefix) `<g7:NPFX>`

Text that appears on a name line before the given and surname parts of a name.

### `NSFX` (Name suffix) `<g7:NSFX>`

Text which appears on a name line after or behind the given and surname parts of a name.

### `OBJE` (Object) `<g7:OBJE>`

See `MULTIMEDIA_LINK` (p.61).

### `OBJE` (Object) `<g7:record-OBJE>`

See `MULTIMEDIA_RECORD` (p.46).

### `OCCU` (Occupation) `<g7:OCCU>`

An Individual Attribute (p.70). See also `INDIVIDUAL_ATTRIBUTE_STRUCTURE` (p.55).

### `ORDN` (Ordination) `<g7:ORDN>`

An Individual Event (p.67). See also `INDIVIDUAL_EVENT_STRUCTURE` (p.57).

### `PAGE` (Page) `<g7:PAGE>`

A specific location within the information referenced. For a published work, this could include the volume of a multi-volume work and the page number or numbers. For a periodical, it could include volume, issue, and page numbers. For a newspaper, it could include a date, page number, and column number. For an unpublished source or microfilmed works, this could be a film or sheet number, page number, or frame number. A census record might have an enumerating district, page number, line number, dwelling number, and family number.

It is recommended that the data in this field be formatted comma-separated with label: value pairs

*Example* —

```
2 SOUR @S1@
3 PAGE Film: 1234567, Frame: 344, Line: 28
```

If the superstructure's pointer is `@VOID@` then there is no information referenced and the `PAGE` may describe the entire source.

*Example* —

```
1 DSCR Tall enough his head touched the ceiling
2 SOUR @VOID@
3 PAGE His grand-daughter Lydia told me this in 1980
```

### `PEDI` (Pedigree) `<g7:PEDI>`

An enumerated value from set `g7:enumset-PEDI` (p.104) indicating the type of child-to-family relationship represented by the superstructure.

### `PHON` (Phone) `<g7:PHON>`

A telephone number. Telephone numbers have many regional variations and can contain non-digit characters. Users should be encouraged to use internationalized telephone numbers rather than local versions. As a starting point for this recommendation, there are international standards that use a "'+'" shorthand for the international prefix (for example, in place of "011" in the US or "00" in the UK). Examples are `+1 (555) 555-1234` (US) or `+44 20 1234 1234` (UK).

See ITU standards E.123 and E.164 for more information.

### `PHRASE` (Phrase) `<g7:PHRASE>`

Textual information that cannot be expressed in the superstructure due to the limitations of its data type. A `PHRASE` may restate information contained in the superstructure, but doing so is not recommended unless it is needed for clarity.

*Example* — A date interpreted from the phrase "The Feast of St John" might be

```
2 DATE 24 JUN 1852
3 PHRASE During the feast of St John
```

*Example* — A record using `1648/9` to indicate a change in new year might become

```
2 DATE 30 JAN 1649
3 PHRASE 30th of January, 1648/9
```

*Example* — A record using `1648/9` to indicate uncertainty in the year might become

```
2 DATE BET 1648 AND 1649
3 PHRASE 1648/9
```

*Example* — A record using `Q1 1867` to indicate an event occurred sometime within the first quarter of 1867 might become

```
2 DATE BET 1 JAN 1867 AND 31 MAR 1867
3 PHRASE Q1 1867
```

*Example* — A record defining the Maid of Honor in a marriage might become

```
1 MARR
2 ASSO @I2@
3 ROLE OTHER
4 PHRASE Maid of Honor
```

*Example* — A name given to a foundling orphan might be

```
1 NAME Mary //
2 GIVN Mary
2 TYPE OTHER
3 PHRASE given by orphanage
```

*Example* — A record specifying a writer's "pen name" (a type of professional name) might become

```
1 NAME Mark /Twain/
2 TYPE PROFESSIONAL
3 PHRASE Pen
```

### `PLAC` (Place) `<g7:PLAC>`

The principal place in which the superstructure's subject occurred, represented as a List (p.32) of jurisdictional entities in a sequence from the lowest to the highest jurisdiction, where "jurisdiction" includes units in a political, ecclesiastical, and geographical hierarchies and may include units of any size, such as a continent, "at sea", or a specific building, farm, or cemetery. As with other lists, the jurisdictions are separated by commas. Any jurisdiction's name that is missing is still accounted for by an empty string in the list.

The type of each jurisdiction is given in the `PLAC.FORM` (p.82) substructure, if present, or in the `HEAD.PLAC.FORM` structure. If neither is present, the jurisdictional types are unspecified beyond the lowest-to-highest order noted above.

### `PLAC` (Place) `<g7:HEAD-PLAC>`

This is a placeholder for providing a default `PLAC.FORM`, and must not have a payload.

### `POST` (Postal code) `<g7:POST>`

A code used by a postal service to identify an area to facilitate mail handling. See `ADDRESS_STRUCTURE` (p.50) for more details.

### `PROB` (Probate) `<g7:PROB>`

An Individual Event (p.67). See also `INDIVIDUAL_EVENT_STRUCTURE` (p.57).

### `PROP` (Property) `<g7:PROP>`

An Individual Attribute (p.70). See also `INDIVIDUAL_ATTRIBUTE_STRUCTURE` (p.55).

### `PUBL` (Publication) `<g7:PUBL>`

When and where the record was created. For published works, this includes information such as the city of publication, name of the publisher, and year of publication.

For an unpublished work, it includes the date the record was created and the place where it was created, such as the county and state of residence of a person making a declaration for a pension or the city and state of residence of the writer of a letter.

### `QUAY` (Quality of data) `<g7:QUAY>`

An enumerated value from set `g7:enumset-QUAY` (p.104) indicating the credibility of a piece of information, based on its supporting evidence. Some systems use this feature to rank multiple conflicting opinions for display of most likely information first. It is not intended to eliminate the receivers' need to evaluate the evidence for themselves.

### `REFN` (Reference) `<g7:REFN>`

A user-defined number or text that the submitter uses to identify the superstructure. For instance, it may be a record number within the submitter's automated or manual system, or it may be a page and position number on a pedigree chart.

This is metadata about the structure itself, not data about its subject. Multiple structures describing different aspects of the same subject must not have the same `REFN` value.

### `RELI` (Religion) `<g7:RELI>`

A religious denomination associated with the event or attribute described by the superstructure.

### `RELI` (Religion) `<g7:INDI-RELI>`

An Individual Attribute (p.70). See also `INDIVIDUAL_ATTRIBUTE_STRUCTURE` (p.55).

### `RESN` (Restriction) `<g7:RESN>`

A List (p.32) of enumerated values from set `g7:enumset-RESN` (p.105) signifying access to information may be denied or otherwise restricted.

The `RESN` structure is provided to assist software in filtering data that should not be exported or otherwise used in a particular context. It is recommended that tools provide an interface to allow users to filter data on export such that certain `RESN` structure payload entries result in the `RESN` structure and its superstructure being removed from the export. Such removal must abide by some constraints: see Removing data (p.25) for more details.

This is metadata about the structure itself, not data about its subject.

### `REPO` (Repository) `<g7:REPO>`

See `SOURCE_REPOSITORY_CITATION`.

### `REPO` (Repository) `<g7:record-REPO>`

See `REPOSITORY_RECORD`.

### `RESI` (Residence) `<g7:FAM-RESI>`

A Family Attribute (p.71). See also `FAMILY_ATTRIBUTE_STRUCTURE` (p.53).

See `g7:INDI-RESI` for comments on the use of payload strings in `RESI` (p.71) structures.

### `RESI` (Residence) `<g7:INDI-RESI>`

An Individual Attribute (p.70). See also `INDIVIDUAL_ATTRIBUTE_STRUCTURE` (p.55).

Where possible, the residence should be identified in `PLAC` (p.91) and/or `ADDR` (p.73) substructures of the `RESI` structure. The payload text should not duplicate `PLAC` or `ADDR` information, but may be used for residence information that cannot be expressed by those structures.

> *Example* — The following two examples show situations where a `RESI` payload may be appropriate:

```
1 RESI living with an aunt
2 DATE ABT MAR 1894
```

```
1 RESI in a mobile caravan
2 PLAC , , Austro-Hungarian Empire
3 FORM City, County, Country
```

### `RETI` (Retirement) `<g7:RETI>`

An Individual Event (p.67). See also `INDIVIDUAL_EVENT_STRUCTURE` (p.57).

### `ROLE` (Role) `<g7:ROLE>`

An enumerated value from set `g7:enumset-ROLE` (p.105) indicating what role this person played in an event or person's life.

> *Example* — The following indicates a child's birth record as the source of the mother's name:
>
> ```
> 0 @I1@ INDI
> 1 NAME Mary //
> 2 SOUR @S1@
> 3 EVEN BIRT
> 4 ROLE MOTH
> ```

> *Example* — The following indicates that a person's best friend was a witness at their baptism:
>
> ```
> 0 @I2@ INDI
> 1 ASSO @I3@
> 2 ROLE FRIEND
> 3 PHRASE best friend
> 1 BAPM
> 2 ASSO @I3@
> 3 ROLE WITN
> ```

### `SCHMA` (Extension schema) `<g7:SCHMA>`

A container for storing meta-information about the extension tags used in this document. See Extensions (p.18) for more details.

### `SDATE` (Sort date) `<g7:SDATE>`

A date to be used as a sorting hint. It is intended for use when the actual date is unknown, but the display order may be dependent on date.

If both a `DATE` (p.78) and `SDATE` are present in the same structure, the `SDATE` should be used for sorting and positioning while the `DATE` should be displayed as the date of the structure.

`SDATE` and its substructures (including `PHRASE` (p.90), `TIME` (p.96), and any extension structures) should be used only as sorting hints, not to convey historical meaning.

It is recommended to use a payload that matches `[[day D] month D] year [D epoch]`. Other DateValue forms may have unreliable effects on sorting. Including a month and day is encouraged to help different applications sort dates the same way, as the relative ordering of dates with different levels of precision is not well defined.

### SEX (Sex) <g7:SEX>

An enumerated value from set g7:enumset-SEX (p.106) that indicates the sex of the individual at birth.

### SLGC (Sealing, child) <g7:SLGC>

A Latter-Day Saint Ordinance (p.72). See also LDS_INDIVIDUAL_ORDINANCE (p.60).

### SLGS (Sealing, spouse) <g7:SLGS>

A Latter-Day Saint Ordinance (p.72). See also LDS_SPOUSE_SEALING (p.61).

### SNOTE (Shared note) <g7:SNOTE>

A pointer to a note that is shared by multiple structures. See NOTE_STRUCTURE (p.62) for more details.

### SNOTE (Shared note) <g7:record-SNOTE>

A note that is shared by multiple structures. See SHARED_NOTE_RECORD (p.48) for more details.

### SOUR (Source) <g7:SOUR>

A description of the relevant part of a source to support the superstructure's data. See SOURCE_CITATION (p.65) for more details.

### SOUR (Source) <g7:record-SOUR>

A description of an entire source. See SOURCE_RECORD (p.49) for more details.

### SOUR (Source) <g7:HEAD-SOUR>

An identifier for the product producing this dataset. A registration process for these identifiers existed for a time, but no longer does. If an existing identifier is known, it should be used. Otherwise, a URI owned by the product should be used instead.

### SPFX (Surname prefix) <g7:SPFX>

A name piece used as a non-indexing pre-part of a surname.

### SSN (Social security number) <g7:SSN>

An Individual Attribute (p.70). See also INDIVIDUAL_ATTRIBUTE_STRUCTURE (p.55).

### STAE (State) <g7:STAE>

A geographical division of a larger jurisdictional area, such as a state within the United States of America. See ADDRESS_STRUCTURE (p.50) for more details.

### `STAT` (Status) `<g7:ord-STAT>`

An enumerated value from set `g7:enumset-ord-STAT` (p.107) assessing of the state or condition of an ordinance.

### `STAT` (Status) `<g7:FAMC-STAT>`

An enumerated value from set `g7:enumset-FAMC-STAT` (p.107) assessing of the state or condition of a researcher's belief in a family connection.

### `SUBM` (Submitter) `<g7:SUBM>`

A contributor of information in the substructure. This is metadata about the structure itself, not data about its subject.

### `SUBM` (Submitter) `<g7:record-SUBM>`

A description of a contributor of information to the document. See `SUBMITTER_RECORD` (p.50) for more details.

### `SURN` (Surname) `<g7:SURN>`

A family name passed on or used by members of a family.

### `TAG` (Extension tag) `<g7:TAG>`

Information relating to a single extension tag as used in this document. See Extensions (p.18) for more details.

### `TEMP` (Temple) `<g7:TEMP>`

The name of a temple of The Church of Jesus Christ of Latter-day Saints. Previous versions recommended using a set of abbreviations for temple names, but the list of abbreviations is no longer published by the Church and using abbreviations is no longer recommended.

### `TEXT` (Text from Source) `<g7:TEXT>`

A verbatim copy of any description contained within the source. This indicates notes or text that are actually contained in the source document, not the submitter's opinion about the source. This should be, from the evidence point of view, "what the original record keeper said" as opposed to the researcher's interpretation.

### `TIME` (Time) `<g7:TIME>`

A `Time` (p.31) value in a 24-hour clock format.

### `TITL` (Title) `<g7:TITL>`

The title, formal or informal, of the superstructure.

A published work, such as a book, might have a title plus the title of the series of which the book is a part. A magazine article would have a title plus the title of the magazine that published the article.

For an unpublished work, including most digital files, titles should be descriptive and appropriate to the work.

> *Example —*
>
> - The `TITL` of a letter might include the date, the sender, and the receiver.
> - The `TITL` of a transaction between a buyer and seller might have their names and the transaction date.
> - The `TITL` of a family Bible containing genealogical information might have past and present owners and a physical description of the book.
> - The `TITL` of a personal interview would cite the informant and interviewer.

Some sources may have a citation text that cannot readily be represented using the `SOURCE_RECORD` (p.49) substructures `AUTH` (p.75), `PUBL` (p.92), `REPO` (p.93), and so on. In such cases, the entire citation text may be presented as the payload of the `SOUR.TITL`.

### `TITL` (Title) `<g7:INDI-TITL>`

An Individual Attribute (p.70). See also `INDIVIDUAL_ATTRIBUTE_STRUCTURE` (p.55).

### `TOP` (Top crop width) `<g7:TOP>`

A number of pixels to not display from the top side of the image. See `CROP` (p.77) for more details.

### `TRAN` (Translation)

A representation of the superstructure's data in a different format.

In some situations it is desirable to provide the same semantic content in multiple formats. Where this is desirable, a `TRAN` substructure is used, where the specific format is given in its language tag substructure, media type substructure, or both.

Different `TRAN` structures are used in different contexts to fully capture the structure of the information being presented in multiple formats. In all cases, a `TRAN` structure's payload and substructures should provide only information also contained in the `TRAN` structures' superstructure, but provide it in a new language, script, or media type.

Each `TRAN` substructure must have either a language tag or a media type or both. Each `TRAN` structure must differ from its superstructure and from every other `TRAN` substructure of its superstructure in either its language tag or its media type or both.

### `TRAN` (Translation) `<g7:NAME-TRAN>`

A type of `TRAN` substructure specific to Personal Names (p.33). Each `NAME.TRAN` must have a `LANG` (p.84) substructure. See also `INDI.NAME` (p.88).

> *Example* — The following presents a name in Mandarin, transliterated using Pinyin
>
> ```
> 1 NAME /孔/德庸
> 2 GIVN 德庸
> 2 SURN 孔
> 2 TRAN /Kǒng/ Déyōng
> 3 GIVN Déyōng
> 3 SURN Kǒng
> 3 LANG zh-pinyin
> ```

### `TRAN` (Translation) `<g7:PLAC-TRAN>`

A type of `TRAN` (p.97) substructure specific to places. Each `PLAC.TRAN` must have a `LANG` (p.84) substructure. See also `PLAC` (p.91).

> *Example* — The following presents a place in Japanese with a romaji transliteration and English translation
>
> ```
> 2 PLAC 千代田, 東京, 日本
> 3 FORM 区, 都, 国
> 3 LANG ja
> 3 TRAN Chiyoda, Tokyo, Nihon
> 4 LANG ja-Latn
> 3 TRAN Chiyoda, Tokyo, Japan
> 4 LANG en
> ```

### `TRAN` (Translation) `<g7:NOTE-TRAN>`

A type of `TRAN` (p.97) for unstructured human-readable text, such as is found in `NOTE` (p.89) and `SNOTE` (p.95) payloads. Each `g7:NOTE-TRAN` must have either a `LANG` (p.84) substructure or a `MIME` (p.87) substructure or both. If either is missing, it is assumed to have the same value as the superstructure. See also `NOTE` and `SNOTE`.

> *Example* — The following presents the same note in HTML-format English; in plain-text with the same language as the superstructure (English); and in Spanish with the same media type as the superstructure (HTML).
>
> ```
> 1 NAME Arete /Hernandez/
> 2 NOTE Named after Arete from The Odyssey
> 3 LANG en
> 3 MIME text/html
> 3 TRAN Named after Arete from "The Odyssey"
> 4 MIME text/plain
> 3 TRAN Nombrada en honor a Arete de La Odisea
> 4 LANG es
> ```

It is recommended that text given in `text/html` should only be translated into `text/plain` if the resulting text is different from the text created by the HTML-to-text conversion process defined in `g7:MIME` (p.87).

### `TRAN` (Translation) `<g7:FILE-TRAN>`

A type of `TRAN` (p.97) for external media files. Each `g7:NOTE-TRAN` (p.98) must have a `FORM` (p.82) substructure. See also `FILE` (p.82) and the File Path datatype (p.35).

> *Example* — If an mp3 audio file has been transcoded as an ogg file and a timestamped transcript has been extracted as a WebVTT file, the resulting set of files might be presented as follows:

```
0 @EX@ OBJE
1 FILE media/original.mp3
2 FORM audio/mp3
2 TRAN media/derived.oga
3 FORM audio/ogg
2 TRAN media/transcript.vtt
3 FORM text/vtt
```

Note that `FILE.TRAN` (p.99) refers to translation to a different digital format, not to translation to a different human language. Files that differ in the human language of their content should each be given their own `FILE` (p.82) structure.

### `TRLR` (Trailer) `<g7:TRLR>`

A pseudo-structure marking the end of a dataset. See The Header and Trailer (p.17) for more details.

### `TYPE` (Type) `<g7:TYPE>`

A descriptive word or phrase used to further classify the superstructure.

When both a `NOTE` (p.89) and free-text `TYPE` are permitted as substructures of the same structure, the displaying systems should always display the `TYPE` value when they display the data from the associated structure; `NOTE` will typically be visible only in a detailed view.

`TYPE` must be used whenever the generic `EVEN`, `FACT` and `IDNO` (p.84) tags are used. It may also be used for any other event or attribute.

Using the subordinate `TYPE` classification method provides a further classification of the superstructure but does not change its basic meaning.

> *Example* — A `ORDN` (p.89) with a `TYPE` could clarify what kind of ordination was performed:
>
> ```
> 0 @I1@ INDI
> 1 ORDN
> 2 TYPE Bishop
> ```
>
> This classifies the entry as an ordination as a bishop, which is still a ordination event. The event could be further clarified with `RELI` (p.92), `DATE` (p.78), and other substructures.
>
> Other descriptor values might include, for example,
>
> - "Stillborn" as a qualifier to `BIRT` (p.75) (birth)
> - "Civil" as a qualifier to `MARR` (p.86) (marriage)
> - "College" as a qualifier to `GRAD` (p.83) (graduation)
> - "Oral" as a qualifier to `WILL` (p.102)
>
> See also `FACT` and `EVEN` for additional examples.

### `TYPE` (Type) `<g7:NAME-TYPE>`

An enumerated value from set `g7:enumset-NAME-TYPE` (p.109) indicating the type of the name.

### `TYPE` (Type) `<g7:EXID-TYPE>`

The authority issuing the `EXID` (p.80), represented as a URI. It is recommended that this be a URL.

If the authority maintains stable URLs for each identifier it issues, it is recommended that the `TYPE` (p.99) payload be selected such that appending the `EXID` payload to it yields that URL. However, this is not required and a different URI for the set of issued identifiers may be used instead.

Registered URIs are listed in the exid-types registry, where fields are defined using the YAML file format.

Additional type URIs can be registered by filing a GitHub pull request.

### `UID` (Unique Identifier) `<g7:UID>`

A globally-unique identifier of the superstructure, to be preserved across edits. If a globally-unique identifier for the record already exists, it should be used without modification, not even whitespace or letter case normalization. It is recommended that new globally unique identifiers be created and formatted using the UUID production specified in RFC 9562 Section 4.

This is metadata about the structure itself, not data about its subject. Multiple structures describing different aspects of the same subject would have different `UID` values.

Because the `UID` identifies a structure, it can facilitate inter-tool collaboration by distinguishing between a structure being edited and a new structure being created. If an application allows structures to be edited in a way that completely changes their meaning (e.g., changing all the contents of an `INDI` record to have it describe a completely different person) then any `UID`s should also be changed.

> *Note* — Some systems used a 16-byte UUID with a custom 2-byte checksum for a total of 18 bytes:
>
> - checksum byte 1 = (sum of ($byte_i$) for $i$ 1 through 16) mod 256
> - checksum byte 2 = (sum of ((16 – $i$) × ($byte_i$)) for $i$ 1 through 16) mod 256
>
> Use of checksums for UIDs is discouraged except in cases where error-prone input is expected and an appropriate action to take in case of an error is known.

### `VERS` (Version) `<g7:VERS>`

An identifier that represents the version level assigned to the associated product. It is defined and changed by the creators of the product.

### `VERS` (Version) `<g7:GEDC-VERS>`

The version number of the official specification that this document's data conforms to. This must include the major and minor version (for example, "`7.0`"); it may include the patch as well (for example, "`7.0.1`"), but doing so is not required. See A Guide to Version Numbers (p.8) for more details about version numbers.

### `WIDTH` (Width in pixels) `<g7:WIDTH>`

How many pixels to display horizontally for the image. See `CROP` (p.77) for more details.

### `WIFE` (Wife) `<g7:WIFE>`

A structure for storing information related to one partner in the context of a `FAMILY_EVENT`; in particular, the partner referenced in the `g7:FAM-WIFE` substructure of the `g7:record-FAM` (p.81) superstructure of the `FAMILY_EVENT`.

> *Example* — The following indicates that individual `@I2@` was 32 years old at the time of the marriage, without indicating an age for individual `@I1@`.

```
0 @F1@ FAM
1 HUSB @I1@
1 WIFE @I2@
1 MARR
2 WIFE
3 AGE 32y
```

### `WIFE` (Wife) `<g7:FAM-WIFE>`

A partner in a `FAM` record. See `FAMILY_RECORD` (p.43) for more details.

### `WILL` (Will) `<g7:WILL>`

An Individual Event (p.67). See also `INDIVIDUAL_EVENT_STRUCTURE` (p.57).

### `WWW` (Web address) `<g7:WWW>`

A URL or other locator for a World Wide Web page of the subject of the superstructure, as defined by any relevant standard such as whatwg/url, RFC 3986, RFC 3987, and so forth.

Like other substructures, the `WWW` structure provides details about the subject of its superstructure. For example, a `MARR` (p.69).`WWW` is a world wide web page of the marriage event, not the personal website of the couple or an entry in an online database serving as a source documenting the marriage. However, the meaning of `WWW` was only implicit when it was introduced in version 5.5.1 and many files were created that use `WWW` to store a more tangentially-related web address, so applications are recommended to interpret the `WWW` structure's meaning cautiously.

If an invalid or no longer existing web address is present upon import, it should be preserved as-is on export.

## 3.4. Enumeration Values

Unless otherwise specified in the enumeration description in this section, each enumeration value defined in this section has a URI constructed by concatenating `g7:enum-` to the enumeration value; for example, the `HUSB` (p.83) enumeration value has the URI `http://gedcom.io/terms/v7/enum-HUSB`.

Each set of enumeration values has its own URI.

### `<g7:enumset-ADOP>`

| Value | Meaning |
|---|---|
| `HUSB` | Adopted by the `HUSB` of the `FAM` pointed to by `FAMC`.<br>The URI of this value is `g7:enum-ADOP-HUSB` |
| `WIFE` | Adopted by the `WIFE` of the `FAM` pointed to by `FAMC`.<br>The URI of this value is `g7:enum-ADOP-WIFE` |
| `BOTH` | Adopted by both `HUSB` and `WIFE` of the `FAM` pointed to by `FAMC` |

### `<g7:enumset-EVEN>`

An event-type tag name, but not the generic `EVEN` tag. See Events (p.66).

Most values in this enumeration set use the same tag and URI as the corresponding event, except for tags used with different URIs for `FAM` vs `INDI`; these are given generic definitions with URIs constructed by concatenating `g7:enum-` to the enumeration value:

| Value | Meaning |
|---|---|
| `CENS` | A census event; either `g7:INDI-CENS` (p.76) or `g7:FAM-CENS` (p.76) |

## <g7:enumset-EVENATTR>

An event- or attribute-type tag name. See Events (p.66) and Attributes (p.70).

Most values in this enumeration set use the same tag and URI as the corresponding event or attribute, except for tags used with different URIs for `FAM` vs `INDI`; these are given generic definitions with URIs constructed by concatenating `g7:enum-` to the enumeration value:

| Value | Meaning |
|---|---|
| `CENS` | A census event; either `g7:INDI-CENS` or `g7:FAM-CENS` |
| `NCHI` | A count of children; either `g7:INDI-NCHI` (p.88) or `g7:FAM-NCHI` (p.88) |
| `RESI` | A residence attribute; either `g7:INDI-RESI` (p.93) or `g7:FAM-RESI` (p.93) |
| `FACT` | A generic attribute; either `g7:INDI-FACT` (p.81) or `g7:FAM-FACT` (p.81) |
| `EVEN` | A generic event; either `g7:INDI-EVEN` (p.80) or `g7:FAM-EVEN` (p.80) |

## <g7:enumset-MEDI>

| Value | Meaning |
|---|---|
| `AUDIO` | An audio recording |
| `BOOK` | A bound book |
| `CARD` | A card or file entry |
| `ELECTRONIC` | A digital artifact such as a computer file or a scan |
| `FICHE` | Microfiche |
| `FILM` | Microfilm |
| `MAGAZINE` | Printed periodical |
| `MANUSCRIPT` | Written pages |
| `MAP` | Cartographic map |
| `NEWSPAPER` | Printed newspaper |
| `PHOTO` | Photograph |

| Value | Meaning |
|---|---|
| `TOMBSTONE` | Burial marker or related memorial |
| `VIDEO` | Motion picture recording |
| `OTHER` | A value not listed here; should have a `PHRASE` substructure |

## <g7:enumset-PEDI>

| Value | Meaning |
|---|---|
| `ADOPTED` | Adoptive parents |
| `BIRTH` | Family structure at time of birth |
| `FOSTER` | The child was included in a foster or guardian family |
| `SEALING` | The child was sealed to parents other than birth parents |
| `OTHER` | A value not listed here; should have a `PHRASE` substructure |

> *Note* — It is known that some users have interpreted `BIRTH` to mean "genetic parent" and others to mean "social parent at time of birth". Definitions differ in many circumstances (infidelity, surrogacy, sperm donation, and so on). Hence, applications should refrain from asserting it has either meaning in imported data.

> *Note* — The structures for foster children in particular, and family relationships in general, are known to have undesirable limitations and are likely to change in a future version of this specification.

> *Note* — `SEALING` implies that a `SLGC` (p.95) event was performed, and it is recommended that this enumeration value only be used when the `SLGC` event is present in the GEDCOM file. `ADOPTED`, on the other hand, only implies a social relationship which may or may not have any associated `ADOP` event.

## <g7:enumset-QUAY>

| Value | Meaning |
|---|---|
| `0` | Unreliable evidence or estimated data |
| `1` | Questionable reliability of evidence (interviews, census, oral genealogies, or potential for bias, such as an autobiography) |
| `2` | Secondary evidence, data officially recorded sometime after the event |
| `3` | Direct and primary evidence used, or by dominance of the evidence |

Although the values look like integers, they do not have numeric meaning.

> *Note* — The structures for representing the strength of and confidence in various claims are known to be inadequate and are likely to change in a future version of this specification.

## <g7:enumset-RESN>

| Value | Meaning |
|---|---|
| `CONFIDENTIAL` | This data was marked as confidential by the user. |
| `LOCKED` | Some systems may ignore changes to this data. |
| `PRIVACY` | This data is not to be shared outside of a trusted circle, generally because it contains information about living individuals. This definition is known to admit multiple interpretations, so use of the `PRIVACY` restriction notice is not recommended. |

It is recommended that applications allow users to chose how `CONFIDENTIAL` and/or `PRIVACY` data is handled when interfacing with other users or applications, for example by allowing them to exclude such data when exporting.

When a List (p.32) of `RESN` (p.93) enumeration values are present, all apply.

> *Example* — The line `1 RESN CONFIDENTIAL, LOCKED` means the superstructure's data is both considered confidential *and* read-only.

Since `RESN` was introduced in version 5.5 the intent of the `PRIVACY` value has been interpreted differently by different applications. Known interpretations include

- Some assign `PRIVACY` by algorithm or policy, unlike the user-assigned `CONFIDENTIAL`
- Some use `PRIVACY` to mark records that have already had private data removed
- Some use the English definitions of "privacy" and "confidential" to inform different restrictions for each

There may also be applications using `PRIVACY` with interpretations not listed above.

Because these different interpretations became widespread before they were identified, determining which one is meant generally requires knowledge of which application applied the `PRIVACY` restriction notice. It is anticipated that a future version will deprecate the `PRIVACY` option and introduce new values for each of its current use cases.

## <g7:enumset-ROLE>

| Value | Meaning |
|---|---|
| `CHIL` | Child |

| Value | Meaning |
|---|---|
| `CLERGY` | Religious official in event; implies `OFFICIATOR` |
| `FATH` | Father; implies `PARENT` |
| `FRIEND` | Friend |
| `GODP` | Godparent or related role in other religions |
| `HUSB` | Husband; implies `SPOU` |
| `MOTH` | Mother; implies `PARENT` |
| `MULTIPLE` | A sibling from the same pregnancy (twin, triplet, quadruplet, and so on). A `PHRASE` can be used to specify the kind of multiple birth. |
| `NGHBR` | Neighbor |
| `OFFICIATOR` | Officiator of the event |
| `PARENT` | Parent |
| `SPOU` | Spouse |
| `WIFE` | Wife; implies `SPOU` |
| `WITN` | Witness |
| `OTHER` | A value not listed here; should have a `PHRASE` substructure |

These should be interpreted in the context of the recorded event and its primary participants. For example, if you cite a child's birth record as the source of the mother's name, the value for this field is "`MOTH`." If you describe the groom of a marriage, the role is "`HUSB` (p.83)."

## <g7:enumset-SEX>

| Value | Meaning |
|---|---|
| `M` | Male |
| `F` | Female |
| `X` | Does not fit the typical definition of only Male or only Female |
| `U` | Cannot be determined from available sources |

This can describe an individual's reproductive or sexual anatomy at birth. Related concepts of gender identity or sexual preference are not currently given their own tag. Cultural or personal gender preference may be indicated using the `FACT` tag.

## <g7:enumset-FAMC-STAT>

| Value | Meaning |
|---|---|
| `CHALLENGED` | Linking this child to this family is suspect, but the linkage has been neither proven nor disproven. |
| `DISPROVEN` | There has been a claim by some that this child belongs to this family, but the linkage has been disproven. |
| `PROVEN` | Linking this child to this family has been proven. |

When these enumeration values were introduced in version 5.5.1 it was assumed, but never specified, that “proven” referred to the definition provided by the Board for Certification of Genealogists. Because that meaning was not specified and other definitions of “proven” exist, existing files might use these values in other ways.

> *Note* — The structures for representing the strength of and confidence in various claims are known to be inadequate and are likely to change in a future version of this specification.

## <g7:enumset-ord-STAT>

These values were formerly used by The Church of Jesus Christ of Latter-day Saints for coordinating between temples and members. They are no longer used in that way, meaning their interpretation is subject to individual user interpretation

The definition of some of these values combined with the official policies of the church mean that some values only make sense under a subset of ordinance structures (p.72). These contexts are identified in the “applies to” column below, and it is recommended that applications follow those guidelines. These recommendations were not present when these enumeration values were first introduced in version 5.3 so uses that do not conform to the “applies to” guidelines may be encountered; if so, they should be treated by applications like any other user-specified but semantically-strange data.

The definition of some of these values combined with the official policies of the church and the move of the church away from using GEDCOM for handling ordinance requests make them redundant and/or no longer relevant. If so, that is indicated in the “status” column below. Like the “applies to” column, the “status” column is a recommendation, not a requirement, and applications should be prepared to encounter non-current values.

| Value | Applies to | Meaning | Status |
|---|---|---|---|
| `BIC` | `SLGC` | Born in the covenant, so child to parent sealing ordinance is not required. | Current |
| `CANCELED` | `SLGS` | Canceled and considered invalid. | Current |
| `CHILD` | All but `SLGC` | Died before 8 years old, so ordinances other than child to parent sealing are not required. | Current |
| `COMPLETED` | All | Completed, but the date is not known. | Deprecated, use `DATE BEF date` instead. This status was defined for use with TempleReady which is no longer in use. |
| `EXCLUDED` | All | Patron excluded this ordinance from being cleared in this submission. | Deprecated. This status was defined for use with TempleReady which is no longer in use. |
| `DNS` | `SLGC`, `SLGS` | This ordinance is not authorized. | Current |
| `DNS_CAN` | `SLGS` | This ordinance is not authorized, and the previous ordinance is cancelled. | Current |
| `INFANT` | All but `SLGC` | Died before less than 1 year old, baptism or endowment not required. | Deprecated. Use `CHILD` instead. |

| Value | Applies to | Meaning | Status |
|---|---|---|---|
| `PRE_1970` | All | Ordinance was likely completed because an ordinance for this person was converted from temple records of work completed before 1970. | Deprecated. Use `DATE BEF 1970` instead. |
| `STILLBORN` | All | Born dead, so no ordinances are required. | Current |
| `SUBMITTED` | All | Ordinance was previously submitted. | Deprecated. This status was defined for use with TempleReady which is no longer in use. |
| `UNCLEARED` | All | Data for clearing the ordinance request was insufficient. | Deprecated. This status was defined for use with TempleReady which is no longer in use. |

## <g7:enumset-NAME-TYPE>

| Value | Meaning |
|---|---|
| `AKA` | Also known as, alias, etc. |
| `BIRTH` | Name given at or near birth. |
| `IMMIGRANT` | Name assumed at the time of immigration. |
| `MAIDEN` | Maiden name, name before first marriage. |
| `MARRIED` | Married name, assumed as part of marriage. |
| `PROFESSIONAL` | Name used professionally (pen, screen, stage name). |
| `OTHER` | A value not listed here; should have a `PHRASE` substructure |

# 4. The FamilySearch GEDZIP file format

It is often useful to transmit a dataset together with a set of external files. The FamilySearch GEDZIP 7.0 file format is provided for this purpose. Version 7.0 was the first version of GEDZIP released; the version number of a GEDZIP file is the same as the version number of the dataset it contains.

A GEDZIP file is a zip archive, as defined by the .ZIP File Format Specification and standardized by ISO/IEC 21320-1:2015.

Each GEDZIP file contains the following entries:

- An entry with name `gedcom.ged` containing a data stream.
- An entry for each *local file* `g7:type-FilePath` payload in `gedcom.ged`, with the same zip *file name* as the payload. If there is a local file named `gedcom.ged`, it must be renamed to a new unused filename with the same extension prior to constructing the GEDZIP.

All file names inside a GEDZIP are case-sensitive UTF-8. Whereas a `g7:type-FilePath` payload (p.35) must percent-escape characters in the file name that cannot appear literally in a URI reference or valid URL string, zip *file names* are not percent-escaped.

Many other zip-based file formats (such as jar, epub, docx, GEDCOM-X) assign special meaning to the zip directory `META-INF` and the zip file names `MANIFEST.MF` and `META-INF/MANIFEST.MF`. These have no special meaning in GEDZIP and it is recommended that they not be used in a GEDZIP file, both to avoid confusing systems that look inside zip archives to determine their file type, and to leave open the possibility of their addition in a future version of this specification.

When saved as a file, a GEDZIP should use the filename extension `.gdz`.

Zip archives can encrypt their contained files' contents, with multiple encryption algorithms supported by the zip archive specification. Encrypted GEDZIP files are a portable way to encrypt FamilySearch GEDCOM data. Only contained file contents are encrypted by zip's encryption: the names and sizes of its contained files are not encrypted. Encrypting the entire GEDZIP file with an external encryption scheme can encrypt file names and sizes, but requires an external method for communicating the encryption scheme chosen.

> *Note* — A few details about the zip archive format are useful to fully understand GEDZIP:
>
> - An archive can contain 1 or more files.
> - Files within an archive can be added, removed, or updated individually without needing to re-process the rest of the archive. Libraries such as libzip allow applications to operate directly on the zip archive as if it were a normal directory tree.
> - What the zip specification calls a “file name” is actually a local path and may contain directories.
> - Directory separators are `/` internally and are converted to the appropriate form by the zip processing tool during zipping and unzipping. Because of this, unzipping a GEDZIP in any local directory results in all GEDZIP file paths working as-is for the resulting `gedcom.ged` without the need for any additional processing.
> - The zip specification supports multiple levels of compression, which can be applied independently to individual files in the archive. Adding compression to an already-compressed file (such as most image and video files) tends to be slow and to not shrink the file. Applications may find performance improved by not applying compression at all or by applying compression only to the text files (including `gedcom.ged`) within the GEDZIP archive.

# 5. Contributors

This document was based on *The GEDCOM Standard Release 5.5.1*, and could not have existed without the contributors to that and previous versions of the specification. Appreciation is extended to all family history participants that have made FamilySearch GEDCOM the *de facto* standard for saving and transferring genealogical information.

New contributions in this edition benefited from the input of a large number of people:

**Managing Editors**

- Gordon Clarke, **FamilySearch**
- Luther Tychonievich, **FHISO** and **University of Virginia**

**Taskforce**

- David Pugmire, **FamilySearch**
- Jimmy Zimmerman, **FamilySearch**
- Larry Telford, **FamilySearch**
- Matt Misbach, **FamilySearch**
- Russell Lynch, **FamilySearch**
- Robert Raymond, **FamilySearch**
- Gaylon Findlay, **Ancestral Quest**
- Derek Maude, **Ancestry**
- Simon Orde, **Family Historian**
- James Tanner, **The Family History Guide**
- John Cardinal, **Family History Hosting**
- Albert Emmerich, **GEDCOM-L**
- Dave Berdan, **Legacy Family Tree**
- Evgen Zherebniy, **Software Mackiev**
- Jason Fletcher, **Midlera Software**
- Uri Gonen, **MyHeritage**
- Dallan Quass, **OurRoots.org**
- Tony Proctor, **SVG Family-Tree Generator**
- Bill Harten, **Puzzilla**
- Bruce Buzbee and Mike Booth, **RootsMagic**

**Development Teams**

- **Tags team**: Luther Tychonievich, Albert Emmerich, Russell Lynch, Tony Proctor, John Cardinal
- **Extensions team**: Luther Tychonievich, Tony Proctor, Jimmy Zimmerman
- **Notes team**: Dallan Quass, David Pugmire, Jason Fletcher, Russell Lynch

- **External Media team**: Dallan Quass, Jason Fletcher, Derek Maude

# 6. Appendix A: Known Calendars and Dates

## 6.1. Known Calendars

This specification defines 4 calendars: `GREGORIAN`, `JULIAN` (p.115), `FRENCH_R` (p.115), and `HEBREW` (p.116). Previous versions also provided for, but did not define the meaning of, `ROMAN` and `UNKNOWN` calendars.

Extension calendars should use the usual rules for extensions, including using `_` as the leading character of the calendar name.

Each calendar must list its permitted epochs and their meaning.

All month tags must either be standard tags defined for the month name in some standard calendar or be extension tags.

Each month defined in this section has a URI constructed by concatenating `g7:month-` to the standard tag; for example, the month of Elul has the standard tag `ELL` and the URI `http://gedcom.io/terms/v7/month-ELL`.

Months with extension tags are permitted in standard calendars only when they are documented extension tags with standard URIs defined by the calendar. This is intended for future compatibility, to accommodate cases where an extension calendar later becomes standardized without changing the URI, at which point the calendar name and month names with `_` are acceptable with the now-standard URIs.

### `GREGORIAN`

The Gregorian calendar is the now-ubiquitous calendar introduced by Pope Gregory XIII in 1582 to correct the Julian calendar which was slowly drifting relative to the seasons.

Permitted months are

| stdTag | Name |
|---|---|
| `JAN` | January |
| `FEB` | February |
| `MAR` | March |
| `APR` | April |
| `MAY` | May |
| `JUN` | June |

| stdTag | Name |
|---|---|
| JUL | July |
| AUG | August |
| SEP | September |
| OCT | October |
| NOV | November |
| DEC | December |

The epoch marker BCE is permitted in this calendar; year *y* BCE indicates a year *y* years before year 1. Thus, there is no year 0; year 1 BCE was followed by year 1.

The URI for this calendar is g7:cal-GREGORIAN

## JULIAN

The Julian calendar was 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 3 decades. Years had been counted from various starting epochs during the Julian calendar's use; the version specified by this specification uses the same starting epoch as the Gregorian calendar.

This calendar uses the same months as the Gregorian calendar, differing only in which years February has 29 days.

The epoch marker BCE is permitted in this calendar; year *y* BCE indicates a year *y* years before year 1. Thus, there is no year 0; year 1 BCE was followed by year 1.

The URI for this calendar is g7:cal-JULIAN

## FRENCH_R

The French Republican calendar or French Revolutionary calendar are the names given to the new calendar adopted in 1794 by the French National Convention. This calendar was adopted on Gregorian day 22 September 1792, which was 1 Vendémiaire 1 in this calendar. It was abandoned 18 years later.

Permitted months are

| stdTag | Name |
|---|---|
| VEND | Vendémiaire |
| BRUM | Brumaire |
| FRIM | Frimaire |
| NIVO | Nivôse |
| PLUV | Pluviôse |

| stdTag | Name |
|---|---|
| VENT | Ventôse |
| GERM | Germinal |
| FLOR | Floréal |
| PRAI | Prairial |
| MESS | Messidor |
| THER | Thermidor |
| FRUC | Fructidor |
| COMP | Jour Complémentaires |

No epoch marker is permitted in this calendar.

The URI for this calendar is `g7:cal-FRENCH_R`

## HEBREW

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 traditionally marks new days at sunset, not midnight. Its first day (1 Tishrei 1) primarily overlapped with Gregorian 7 September 3761 BCE and Julian 7 October 3761 BCE (starting at sunset on the 6th day of those months).

| stdTag | Name |
|---|---|
| TSH | Tishrei (תִּשְׁרֵי) |
| CSH | Marcheshvan (מַרְחֶשְׁוָן) or Cheshvan (חֶשְׁוָן) |
| KSL | Kislev (כִּסְלֵו) |
| TVT | Tevet (טֵבֵת) |
| SHV | Shevat (שְׁבָט) |
| ADR | Adar I, Adar Rishon, First Adar, or Adar Aleph (אדר א׳) |
| ADS | Adar (אֲדָר); or Adar II, Adar Sheni, Second Adar, or Adar Bet (אדר ב׳) |
| NSN | Nisan (נִיסָן) |
| IYR | Iyar (אִייָר) |
| SVN | Sivan (סִיוָן) |
| TMZ | Tammuz (תַּמּוּז) |
| AAV | Av (אָב) |
| ELL | Elul (אֱלוּל) |

To keep the lunar-based months synchronized with the solar-based years, some years have Adar I and others do not, instead proceeding from Shevat directly to Adar II. However, in common (non-leap) years, it is common to simply write “Adar” not “Adar II”, which users not aware of the distinction might incorrectly encode as `ADR` (p.116) instead of `ADS`. It is recommended that systems knowing which years had Adar I and which did not replace `ADR` in common years with `ADS`.

No epoch marker is permitted in this calendar.

The URI for this calendar is `g7:cal-HEBREW`

## 6.2. Dual dates

The day on which a new year began and the year number increased varied at different times and places during the use of the Gregorian and Julian calendars. For example, England measured the new year as 25 March until 1752, when it switched to 1 January as the new year. In periods of transition, or when writing after a change about dates occurring before a change, it was sometimes common to indicate 2 years with a slash, for example, “30 January 1648/49” meaning “1648 if you count the new year as coming after 30 January, 1649 if you count it as coming before 30 January”. Other notations, such as abbreviations for phrases like “new style” and “old style”, were also sometimes employed.

```
2 DATE 30 JAN 1649
3 PHRASE 30 January 1648/49
```

Many nations transitioned from using the Julian calendar to using the Gregorian calendar. This transition caused a change in dates by several days, which (depending on the date in question) could change the month and year as well. In periods of transition, or when writing after a change about dates occurring before a change, it was sometimes common to indicate 2 dates with slashes, for example “23/6 November/December 1907” meaning “Julian 23 November 1907, Gregorian 6 December 1907”. Other notations, such as abbreviations for phrases like “new style” and “old style”, were also sometimes employed.

```
2 DATE 6 DEC 1907
3 PHRASE 23/6 November/December 1907
```

Some documents also used slashes to indicate approximate dates, such as writing a birth year as “1903/4” when it was computed from a year-granularity age at a given date.

```
2 DATE BET 1903 AND 1904
3 PHRASE 1903/4
```

Versions 5.3 through 5.5.1 had special syntax for recording the first of these 3 concepts with a slash in the year. However, because slashes appear in historical documents with all 3 of the above meanings, some users misused this notation to record the other 2 situations as well. The result is ambiguity in the intended meaning of the resulting data. Version 7.0 removed the year slash notation; a `PHRASE` (p.90) substructure should be used instead to clarify meaning.

## 6.3. Calendars in date ranges and date periods

Calendars apply to the subsequent `date` (p.28) production, not to the entire `DateValue`. Hence,

- `DATE FROM 1670 TO 1800` means `DATE FROM GREGORIAN 1670 TO GREGORIAN 1800`
- `DATE FROM 1670 TO JULIAN 1800` means `DATE FROM GREGORIAN 1670 TO JULIAN 1800`
- `DATE FROM JULIAN 1670 TO 1800` means `DATE FROM JULIAN 1670 TO GREGORIAN 1800`

Because some systems may show dates as-is to users and because not all users understand the above rule, it is recommended that `calendar` tags be included if any `date` is non-`GREGORIAN` (p.114). It is recommended that the `calendar` tag be omitted if all `date`s in a payload are in the Gregorian calendar. Hence, the recommended forms of the previous 3 dates are

- `DATE FROM 1670 TO 1800`
- `DATE FROM GREGORIAN 1670 TO JULIAN 1800`
- `DATE FROM JULIAN 1670 TO GREGORIAN 1800`