Extended Legacy Format (ELF): Serialisation Format

7 January 2019

> *Editorial note* — This is an **exploratory draft** of the serialisation format for FHISO's proposed suite of Extended Legacy Format (ELF) standards. This document is not endorsed by the FHISO membership, and may be updated, replaced or obsoleted by other documents at any time.
>
> Comments on this draft should be directed to the tsc-public@fhiso.org mailing list.

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

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

> *Note* — Strictly, [GEDCOM 5.5] was the last version to be publicly released back in 1995. However a draft dated 2 October 1999 of a proposed [GEDCOM 5.5.1] was made public; it is generally considered to have the status of a standard and has been widely implemented as such.

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

— **ELF: Serialisation Format**. This standard defines a general-purpose serialisation format based on the GEDCOM data format which encodes a *dataset* as a hierarchical series of *lines*, and provides low-level facilities such as escaping and extensibility mechanisms.
— **ELF: Date, Age and Time Microformats**. This standard defines microformats for representing dates, ages and times in arbitrary calendars, together with how they are applied to the Gregorian, Julian, French Republican and Hebrew calendars.
— **ELF: Data Model**. This standard defines a data model based on the lineage-linked GEDCOM form, reformulated in terms of the serialisation model described in this document. It is not a major update to the GEDCOM data model, but rather a basis for future extension.

1 Conventions used

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

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

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

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

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

> *Note* — In particular, precise meaning of *string*, *character*, *whitespace* and *term* are given in [Basic Concepts].

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

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

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

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

`elf`	`https://terms.fhiso.org/elf/`
`ex`	`https://example.com/`

> *Note* — Although *prefix notation* is included in this standard document (see §4.2.2), that is only in the context of serialised data. When used outside of a serialised example, prefix notation is simply a notational convenience to make the standard easier to read.

2 Overview

A **dataset** consists of *metadata* and a *document*, each containing a collection of *structures*. An **octet stream** consists of an ordered sequence of octets, values between 0 and 255. This document specifies how a *dataset* is serialised into an *octet stream* and how an *octet stream* is parsed into a *dataset*.

2.1 Serialisation

The semantics of serialisation are defined by the following procedural outline.

1. Each *structure* is assigned a *tag* based on its *structure type identifier*, *superstructure type identifier*, and a *schema* which MAY be augmented during serialisation to allow all *structures* to have a *tag*.
2. The *tagged structures* are ordered and additional *tagged structures* created to represent *serialisation metadata*.
   This step cannot happen before tagging because tagging may generate *serialisation metadata* that needs to be included in the *tagged structures*.
3. Payloads are converted to create *xref structures* by simultaneously
   — assigning *xref_ids* and replacing *pointer*-valued *payloads* with *string*-valued *xrefs*
   — escaping `@` *characters*
   — preserving valid *escapes*
   — escaping unrepresentable *characters*
   Semantically, these actions must happen concurrently because none of them should be applied to the others' results.
   This step cannot happen before tagging because tags are needed to determine the set of valid *escapes*. This step cannot happen before adding *serialisation metadata* because it is applied to the *serialisation metadata* as well.
4. The *dataset* is converted to a sequence of *lines* by
   — assigning *levels*
   — splitting *payloads*, if needed, using `CONT` and `CONC`
   — ordering *substructures* in a preorder traversal of the *tagged structures*
   This step cannot happen before payload conversion because valid split points are dependant on proper escaping. This step must happen before encoding as octets because valid split points are determined by *character*, not octet.
5. The sequence of *lines* is converted to an octet stream by
   — concatenating the *lines* with *line-break* terminators
   — converting *strings* to octets using the *character encoding*

2.2 Parsing

The semantics of parsing are defined by the following procedural outline.

1. An octet stream is converted to a sequence of *string* lines by
    a. determining a *character encoding* by
        i. detecting a *character encoding*
        ii. using that *detected character encoding* to look for a *character encoding* specified in the *serialisation metadata*
    b. converting octets to *characters* using that *character encoding*
    c. splitting on *line breaks*
2. *string* lines are parsed as *lines* by
    — parsing the *level*, *tag*, *xref_id*, and *payload* of each *line*
    — creating an *error line* if that fails
3. *lines* are parsed into *xref structures* by
    — re-merging `CONC` and `CONT`-split *payloads*; violations of splitting rules are ignored
    — using *levels* to properly nest *xref structures*
    — converting to *error lines* before parsing if the *levels* are inconsistent
4. *xref structures* are parsed into *tagged structures* by simultaneously
    — converting *xrefs* to *pointers*, with a special “point to null” if this fails
    — unescaping `@` *characters*
    — preserving valid *escapes* and removing others
    — converting *unicode escapes* into their represented *characters*,
5. the *tagged structures* that represent the *schema* are parsed
6. *tags* are converted into *structure type identifiers* using the *schema* and the resulting *structures* placed in the *metadata* or *document* as appropriate. *Tags* with no corresponding *structure type identifier* are converted into appropriate *undefined tag identifiers*.

2.3 Constructs

This document uses five externally-visible constructs: *dataset*, *metadata*, *document*, *structure*, and *octet stream*. For clarity of presentation, it also uses several intermediate constructs internally: *line*, *xref structure*, and *tagged structure*. Each is defined in §2.4.

2.4 Glossary

Character encoding
: The scheme used to map between an *octet stream* and a *string* of *characters*.

Dataset
: *Metadata* and a *document*.

Delimiter
: A sequence of one or more space or tabulation characters.

```
Delim ::= [#20#9]+
```

During serialisation, a single space (U+0020) SHOULD be used each place a *delimiter* is expected.

Document

An unordered set of *structures*.

Line

1. A *level*, a non-negative *integer*
2. An optional *xref_id*
3. A *tag*, a *string* matching production `Tag`
4. An optional *payload*, which is a *string* containing any number of *characters*, but which *must not* contain a *line-break*.

Line break

A sequence of one or more newline and/or carriage return characters.

```
LB ::= [#A#D]+
```

During serialisation, each *line break* MUST be one of

— a single newline (U+000A)
— a single carriage return (U+000D)
— a single carriage return followed by a single newline (U+000D U+000A)

The same string SHOULD be used each place a *line break* is expected.

Metadata

A collections of *structures* intended to describe information about the dataset as a whole.
The relative order of *structures* with the same *structure type identifier* SHALL be preserved within this collection; the relative order of *structures* with distinct *structure type identifiers* is not defined by this specification.

Octet

One of 256 values, often represented as the numbers 0 through 255. Also called a “byte.”

Octet Stream

A sequence of octets.

Record

A *structure*, *tagged structure*, or *xref structure* whose *superstructure* is the *document*.

ELF Schema

Information needed to correctly parse *tagged structures* into *structures*: a mapping between *structure type identifiers* and *tags* and metadata relating to valid *escapes* and *prefixes*.

Serialisation Metadata

Tagged structures inserted during serialisation and removed (with all its *substructures*) during parsing. They are used to serialise the *character encoding* and *ELF schema* as well as to separate the *metadata* and the *document*.

Structure

— A **structure type identifier**, which is a *term*.
— Optionally, a **payload** which is one of
  — A **pointer** to another *structure*, which *must* be a *record* within the same *dataset*.

    - A *string* or subtype thereof.
- One **superstructure**, which is one of
    - Another *structure*; *superstructure* links MUST be acyclic.
    - The *document*.
    - The *metadata*.
- A collection of any number of **substructures**, which are *structures*.
  The relative order of *structures* with the same *structure type identifier* SHALL be preserved within this collection; the relative order of *structures* with distinct *structure type identifiers* is not defined by this specification.

Superstructure type identifier

A *term* identifying the type of the *superstructure* of a *structure*. If the *superstructure* is the *document*, this is `elf:Document`. If the *superstructure* is the *metadata*, this is `elf:Metadata`. Otherwise, this is the *structure type identifier* of a *structures*'s *superstructure*.

> *Note* — *Superstructure type identifier* is not transitive, applying only to the immediate superstructure.

> *Example* — Suppose an `elf:INDIVIDUAL_RECORD` is the superstructure of an `elf:GRADUATION` and the `elf:GRADUATION` is the superstructure of an `elf:AGE_AT_EVENT`. The *superstructure type identifier* of the `elf:AGE_AT_EVENT` is `elf:GRADUATION`, not `elf:INDIVIDUAL_RECORD`.

Tagged Structure

Like a *structure*, except

- it has a *tag* instead of a *structure type identifier*.
- its *substructures* are stored in a sequence with defined order, not in a partially-ordered collection.

Undefined tag identifier

A *term* containing a single # (U+0023) with `elf:Undefined` before it and a *string* matching production `Tag` after it. The *string* after the U+0023 is called the *tag* of the *undefined tag identifier*.

Xref Structure

Like a *tagged structure*, except

- it may have an optional *xref_id*.
- its payload, if present, is always a *string*, not a *pointer*.

3 Tags

3.1 Definitions

A **tag** is a *string* that matches production `Tag`

```
Tag ::= [0-9a-zA-Z_]+
```

A *tag* SHOULD be no more than 15 characters in length.

> *Note* — [GEDCOM 5.5.1] required tags to be unique within the first 15 characters and no more than 31 characters in length. As memory constraints that motivated those requirements are no longer common, ELF has changed that recommended status instead.

A *tag* SHOULD begin with an underscore (_, U+005F) unless it is defined in a FHISO standard.

> *Note* — [GEDCOM 5.5.1] required all tags other than those it defined to begin with an underscore. ELF's use of *structure type identifiers* largely obviates that need, but it remains RECOMMENDED in ELF 1.0.0 to support legacy systems that have special-case handling for underscore-prefixed *tags*. FHISO is considering removing that recommendation in a subsequent version of ELF.

> *Example* — "`HEAD`" is a valid *tag*; so is "`_UUID`". "`23`" and "`UUID`" are also valid, but SHOULD NOT be used as they are not defined in a FHISO standard and do not begin with an underscore. "`_UNCLE_OF_THE_BRIDE`" is valid, but SHOULD NOT be used as it is 19 *characters* long, more than the 15-*character* recommended maximum length.

Structure type identifiers are serialised as *tags* by utilizing *tag definitions* and *supertypes*, as outlined below.

3.1.1 Supertypes

A **supertype definition** specifies one *structure type identifier* that is defined to be a **supertype** of another.

> *Example* — The following are example *supertype definitions* in the *default ELF schema*:
>
> — `elf:FamilyEvent` is a supertype of `elf:MARRIAGE`
> — `elf:Event` is a supertype of `elf:FamilyEvent`
> — `elf:Agent` is a supertype of `elf:SUBMITTER_RECORD`
> — `elf:Record` is a supertype of `elf:SUBMITTER_RECORD`

An **eventual supertype** of a *structure type identifier* is either

— the *structure type identifier* itself
— an *eventual supertype* of at least one of the *structure type identifier*'s *supertypes*

> *Example* — Continuing the previous example,
>
> — `elf:MARRIAGE`, `elf:FamilyEvent`, and `elf:Event` are *eventual supertypes* of `elf:MARRIAGE`
> — `elf:FamilyEvent`, and `elf:Event` are *eventual supertypes* of `elf:FamilyEvent`
> — `elf:SUBMITTER_RECORD`, `elf:Agent`, and `elf:Record` are *eventual supertypes* of `elf:SUBMITTER_RECORD`

If X is an *eventual supertype* of Y, then Y is an **eventual subtype** of X.

> *Example* — Continuing the previous example,
>
> — `elf:MARRIAGE`, `elf:FamilyEvent`, and `elf:Event` are *eventual subtypes* of `elf:Event`
> — `elf:SUBMITTER_RECORD` and `elf:Agent` are *eventual subtypes* of `elf:Agent`
> — `elf:SUBMITTER_RECORD` and `elf:Record` are *eventual subtypes* of `elf:Record`

The *supertype* defined in this specification is only intended to facilitate *tag definitions* and MUST NOT be taken to indicate any semantic relationship between the structure types they describe.

> *Note* — It is expected that underlying data models will often define a semantic supertype-like relationship that mirrors the *supertype definitions* in this document; see [Elf-DataModel] for an example of what this might look like. The prohibition against assuming such from the *supertype definitions* alone provides a clearer separation between data model and serialisation.

> *Editorial note* — We could decide to REQUIRE that any *supertype definition* has meaning in the underlying data model; I chose not to do so in this draft as it required discussing semantics, which this specification otherwise does not need to do.

3.1.2 Tag definitions

The correspondence between *tags* and *structure type identifiers* is provided by a set of **tag definitions**. Each *tag definition* gives the unique *structure type identifier* that a particular *tag* corresponds to if its *superstructure type identifier* is an *eventual subtype* of a given *superstructure type identifier*.

> *Example* — The following are example *tag definitions* in the *default ELF schema*:
>
> — the *structure type identifier* of "HUSB" is `elf:Parent1Age` if its *superstructure* is an `elf:FamilyEvent`.
> — the *structure type identifier* of "HUSB" is `elf:PARENT1_POINTER` if its *superstructure* is an `elf:FAM_RECORD`.
> — the *structure type identifier* of "CAUS" is `elf:CAUSE_OF_EVENT` if its *superstructure* is an `elf:Event`.

> If a *tagged structure* has *tag* "CAUS" and *superstructure type identifier* `elf:MARRIAGE`, it's *structure type identifier* is `elf:CAUSE_OF_EVENT` because of the last of the above *tag definitions* and because `elf:MARRIAGE` is an *eventual subtype* of `elf:Event`.

The set of *tag definitions* and *supertype definitions* MUST NOT provide two (or more) different *structure type identifiers* for any single *structure*.

> *Example —* The following, taken together, are not permitted
>
> — `elf:Agent` is a *supertype* of `elf:SUBMITTER_RECORD`.
> — `elf:Record` is a *supertype* of `elf:SUBMITTER_RECORD`.
> — `ex:AgentKind` is the *structure type identifier* of an "`_EX_KIND`" if its *superstructure* is an `elf:Agent`.
> — `ex:RecordKind` is the *structure type identifier* of an "`_EX_KIND`" if its *superstructure* is an `elf:Record`.
>
> These provide two contradictory *tag definitions* for the tag "`_EX_KIND`" as a *substructure* of an `elf:SUBMITTER_RECORD`.

> *Example —* The following, taken together, are permitted
>
> — `elf:Agent` is a *supertype* of `elf:SUBMITTER_RECORD`.
> — `elf:Record` is a *supertype* of `elf:SUBMITTER_RECORD`.
> — `ex:Kind` is the *structure type identifier* of an "`_EX_KIND`" if its *superstructure* is an `elf:Agent`.
> — `ex:Kind` is the *structure type identifier* of an "`_EX_KIND`" if its *superstructure* is an `elf:Record`.
>
> These provide two *tag definitions* for the tag "`_EX_KIND`" as a *substructure* of an `elf:SUBMITTER_RECORD`, but because both provide the same *structure type identifier* they are permitted.

A *tag definition* is said to apply to a *structure* if and only if the *structure*'s *structure type identifier* is that of the *tag definition* and its *superstructure type identifier* is an *eventual subtype* of the *tag definition*'s *superstructure type identifier*.

A *tag definition* is said to apply to a *tagged structure* if and only if the *tagged structure*'s *tag* is that of the *tag definition* and its *superstructure type identifier* is an *eventual subtype* of the *tag definition*'s *superstructure type identifier*.

3.2 Serialisation

During serialisation, a *conformant* application SHALL ensure the presence of sufficient *tag definitions* that at each *structure* has a defined *tag*, creating new *tag definitions* if needed to achieve this end.

> *Note* — The above is not the same as saying that a *tag definition* is created for each *structure type identifier* because a *structure* with identifier "`elf:Undefined`" or an *undefined tag identifier* has a defined *tag* without a *tag definition*.

New *tag definitions* may be selected arbitrarily, subject to the limitations on *tags* (see §3.1) and *tag definitions* (see §3.1.2) and to the following:

— the *tag* MUST NOT be "`CONT`", "`CONC`", "`ERROR`", "`UNDEF`", or the *tag* of any *undefined tag identifier* in the *dataset*.

> *Note* — "`CONT`", "`CONC`", "`ERROR`", and "`UNDEF`" are special *tags* that can be created at any location within the dataset during deserialisation.

— the *structure type identifier* MUST NOT be any of "`elf:Document`", "`elf:Metadata`", "`elf:Undefined`", or an *undefined tag identifier*.

> *Note* — "`elf:Undefined`" *structures* are used for errors and are serialised differently than other *structures*.

— the (*tag*, *superstructure type identifier*) pair MUST NOT be any of (`HEAD`, `elf:Document`), (`TRLR`, `elf:Document`), (`CHAR`, `elf:Metadata`), or (`SCHMA`, `elf:Metadata`).

> *Note* — These tags and contexts are reserved for encoding *serialisation metadata*.

— all *tag definitions* for a given *structure type identifier* SHOULD use the same *tag*

> *Note* — [GEDCOM 5.5.1] never intentionally violates the above RECOMMENDATION, but via a typo it provides both `EMAI` and `EMAIL` as *tags* for `elf:ADDRESS_EMAIL`. Other aliases exist due to similar mistakes in applications and to multiple extensions inserting the same concept via different *tags*. The ability to handle these aliases is the reason this is a RECOMMENDATION, not a REQUIREMENT, in ELF.

— the *tag definitions* in the default ELF Schema (see §9) SHOULD be used in place of any alternative *tag definitions* for the same *structures* in the same contexts.

Each *structure* is converted to a *tagged structure* with the *tag* being

— `UNDEF` if the *structure type identifier* is `elf:Undefined`.
— The *tag* of the *undefined tag identifier* if the *structure type identifier* is an *undefined tag identifier*.
— The *tag* from one of the *tag definitions* that applies to that *structure* otherwise.
In the event that more than one such *tag* exists, applications SHOULD select the same *tag* in each instance where this choice exists.

> *Note* — If processing *structures* into *tagged structures* in place, it may be easiest to perform a postorder traversal of each *structure* hierarchy; this way the *superstructure* of a *structure* being converted will still have a *structure type identifier*, not a *tag*, which will simplify looking up applicable *tag definitions*.

The *substructures* of a *tagged structure* are stored in a sequence, not set. This ordering of *substructures* of a *tagged structure* MUST maintain the relative order of those *substructures* that were ordered in the corresponding *structure*. It is RECOMMENDED that all *substructures* with the same *tag* be grouped together, but doing so is NOT REQUIRED.

> *Example* — Consider the following *structure* hierarchy
>
> — `elf:INDIVIDUAL_RECORD`
> — `elf:BIRTH`
> — `elf:DATE_VALUE` 20 JUN 1881
> — `elf:GRADUATIONs`:
> 1. `elf:GRADUATION`
> — `elf:AGE_AT_EVENT` 18
> 2. `elf:GRADUATION`
> — `elf:AGE_AT_EVENT` 22
>
> This may be converted to any of the following three *tagged structure* hierarchies, though the second is NOT RECOMMENDED:
>
> — `INDI`
> 1. `BIRTH`
> — DATE 20 JUN 1881
> 2. `GRAD`
> — AGE 18
> 3. `GRAD`
> — AGE 22
> — `INDI`
> 1. `GRAD`
> — AGE 18
> 2. `BIRTH`
> — DATE 20 JUN 1881
> 3. `GRAD`
> — AGE 22
> — `INDI`
> 1. `GRAD`
> — AGE 18
> 2. `GRAD`
> — AGE 22
> 3. `BIRTH`

> — DATE 20 JUN 1881
>
> However, the following puts the *tagged structure* graduations in a different order than the corresponding *structure* graduations and MUST NOT be used:
>
> — INDI
> 1. GRAD
> — AGE 22
> 2. GRAD
> — AGE 18
> 3. BIRTH
> — DATE 20 JUN 1881

3.3 Parsing

When parsing *tagged structures* into *structures*, add the *structure type identifier* from the the applicable *tag definition*.

If there is no applicable *tag definition*, or if there are multiple applicable *tag definitions* providing different *structure type identifiers*, then the *structure type identifier* SHALL be `elf:Undefined` if the *tag* is `UNDEF`, or the *undefined tag identifier* constructed by concatenating `elf:Undefined#` and the *tag* otherwise.

> *Note* — The special tag "`ERROR`" does not require special handling; because it never has a *tag definition*, it becomes the *undefined tag identifier* `elf:Undefined#ERROR`.

4 Serialisation metadata

The *tagged structures* representing the *dataset* are ordered as follows:

1. A *serialisation metadata tagged structure* with *tag* "`HEAD`" and the following *substructures*:
   - A *serialisation metadata tagged structure* with *tag* "`CHAR`" and *payload* identifying the *character encoding* used; see §4.1 for details.
   - A *serialisation metadata tagged structure* with *tag* "`SCHMA`" and no *payload*, with *substructures* encoding the *ELF Schema*; see §4.2 for details.
   - Each *tagged structure* with the *superstructure type identifier* `elf:Metadata`, in an order consistent with the partial order of *structures* present in the *metadata*.
2. Each *tagged structure* with the *superstructure type identifier* `elf:Document`, in arbitrary order.
3. A *serialisation metadata tagged structure* with *tag* "`TRLR`" and no *payload* or *substructures*.

4.1 Charcter encoding names

The *character encoding* SHALL be serialised in the "CHAR" *tagged structure*'s *payload* encoding name in the following table that corresponds to the *character encoding* used to convert the *string* to an *octet stream* (see §7).

Encoding	Description
ASCII	The US version of ASCII defined in [ASCII].
ANSEL	The extended Latin character set for bibliographic use defined in [ANSEL].
UNICODE	Either the UTF-16LE or the UTF-16BE encodings of Unicode defined in [ISO 10646].
UTF-8	The UTF-8 encodings of Unicode defined in [ISO 10646].

It is REQUIRED that the encoding used should be able to represent all code points within the *string*; *unicode escapes* (see §5.3) allow this to be achieved for any supported encoding. It is RECOMMENDED that UTF-8 be used for all datasets.

4.2 ELF Schema

The **ELF Schema** is a *serialisation metadata tagged structure* with *tag* "SCHMA" and no *payload*; it may contain as *substructures* any number of *external schema structures*, *prefix abbreviation structures*, *IRI definition structures*, and *escape preservation structures*.

If, during parsing, no *ELF Schema* is found, the *default ELF schema* defined in §9 SHALL be used.

> *Editorial note —* Do we need to make the default dependant on the GEDC metadata?

If multiple *ELF Schemas* are found, they SHALL be treated as if all of their *substructures* were part of the same *ELF schema*.

During serialisation exactly one *ELF Schema* SHOULD be included.

4.2.1 External schema structure

An **external schema structure** is a *tagged structure* with an *ELF Schema* as its *superstructure*, *tag* SCHMA, no *substructures*, and an IRI as its *payload*. The IRI SHOULD use the http or https scheme and an HTTP GET request sent to it with an Accept header of application/x-fhiso-elf1-schema SHOULD return a *dataset* serialised in accordance with this specification containing an *ELF Schema* defining the full data model in *structure type descriptions*.

> *Editorial note —* Is application/x-fhiso-elf1-schema a MIME-type we are happy with?

> *Example* — When using the [ELF-DataModel] version 1.0.0, the *serialisation schema* could be serialised as
>
> ```
> 0 HEAD
> 1 SCHMA
> 2 SCHMA https://fhiso.org/TR/elf-data-model/v1.0.0
> ```

> *Example* — An HTTP GET request sent to it with an `Accept` header of `application/x-fhiso-elf1-schema` to `https://fhiso.org/TR/elf-data-model/v1.0.0` will return the contents of §9 or the equivalent.

When retrieving a serialised *dataset* via an HTTP GET request to the IRI of an *external schema structure*, all contents of that *dataset* except *ELF Schemas* SHALL be ignored. Additional *external schema structure* SHOULD NOT be present within that *ELF Schema* and if they are, they MAY be ignored.

> *Note* — The recommendation against external schema structures inside other external schema structures is designed to simplify parsing.

4.2.2 Prefix abbreviation structure

> *Editorial note* — Should this section cite §4.3 of [Basic Concepts] instead of its current text?

A **prefix abbreviation structure** is a *tagged structure* with an *ELF Schema* as its *superstructure*, *tag* `PRFX`, and no *substructures*. Its *payload* consist of two *whitespace*-separated tokens: the first is a **prefix** and the second is that *prefix*'s corresponding IRI.

To **prefix expand** a *string*, if that *string* begins with a defined *prefix* followed by a colon (U+003A `:`) then replace that *prefix* and colon with the *prefix*'s corresponding IRI. To **prefix shorten** a *string*, replace it with a *string* that *prefix expansion* would convert to the original *string*.

> *Example* — Given a `PRFX`
>
> ```
> 2 PRFX elf https://fhiso.org/elf/
> ```
>
> the IRI `https://fhiso.org/elf/ADDRESS` may be abbreviated as `elf:ADDRESS`.

4.2.3 IRI definition structure

An **IRI definition structure** is a *tagged structure* with an *ELF Schema* as its *superstructure* and *tag* "`IRI`". Its payload is an IRI, which MAY be *prefix shortened* during serialisation and MUST be *prefix expanded* during parsing. The remainder of this section calls this *prefix expanded* payload I. An *IRI definition structure* may have, as *substructures*, any number of *supertype definition structures* and *tag definition structures*.

A **supertype definition structure** is a *tagged structure* with an *IRI definition structure* as its *superstructure*, *tag* "`ISA`", and no *substructures*. Its payload is a *structure type identifier* which MAY be *prefix shortened* during serialisation and MUST be *prefix expanded* during parsing. The remainder of this section calls this *prefix expanded* payload I'. Each *supertype definition structure* encodes a single *supertype definition*, specifying that I' is a *supertype* of I.

> *Example —* That `elf:ParentPointer` is a *supertype* of `elf:PARENT1_POINTER` can be encoded in a *supertype definition structure* as
>
> ```
> 2 IRI elf:PARENT1_POINTER
> 3 ISA elf:ParentPointer
> ```

A **tag definition structure** is a *tagged structure* with an *IRI definition structure* as its *superstructure*, *tag* "`TAG`", and no *substructure*. Its payload is a *whitespace*-separated list of two or more tokens. The first token T MUST match production `Tag`; each remaining token S is an IRI, which MAY be *prefix shortened* during serialisation and MUST be *prefix expanded* during parsing. Each such S encodes an *tag definition* between *structure type identifier* I and (*tag*, *superstructure type identifier*) pair (T, S).

> *Example —* The following *tag definitions*
>
> — the *structure type identifier* of "`HUSB`" is `elf:Parent1Age` if its *superstructure* is an `elf:FamilyEvent`.
> — the *structure type identifier* of "`HUSB`" is `elf:PARENT1_POINTER` if its *superstructure* is an `elf:FAM_RECORD`.
> — the *structure type identifier* of "`FORM`" is `elf:MULTIMEDIA_FORMAT` if its *superstructure* is an `elf:MULTIMEDIA_RECORD`.
> — the *structure type identifier* of "`FORM`" is `elf:MULTIMEDIA_FORMAT` if its *superstructure* is an `elf:MULTIMEDIA_FILE_REFERENCE`.
> — the *structure type identifier* of "`EMAIL`" is `elf:ADDRESS_EMAIL` if its *superstructure* is an `elf:Agent`.
> — the *structure type identifier* of "`EMAI`" is `elf:ADDRESS_EMAIL` if its *superstructure* is an `elf:Agent`.
>
> can be encoded in *tag definition structures* as
>
> ```
> 0 HEAD
> 1 SCHMA
> ```

```
2 PRFX elf https://fhiso.org/elf/
2 IRI elf:PARENT1_POINTER
3 TAG HUSB elf:FAM_RECORD
2 IRI elf:Parent1Age
3 TAG HUSB elf:FamilyEvent
2 IRI elf:MULTIMEDIA_FORMAT
3 TAG FORM elf:MULTIMEDIA_FILE_REFERENCE elf:MULTIMEDIA_RECORD
2 IRI elf:ADDRESS_EMAIL
3 TAG EMAIL elf:Agent
3 TAG EMAI elf:Agent
```

4.2.4 Escape-preserving tags

> *Note* — This entire section, and all of the related functionality, is present to help cope with the idiosyncratic behaviour of date escapes in [GEDCOM 5.5.1]. Escapes in previous editions of GEDCOM were serialisation-specific and if encountered in ELF should generally be ignored, but date escapes are instead part of a microformat. While escape-preserving tags are not elegant, they are adequate to handle this idiosyncrasy.

> *Editorial note* — I wrote the above note from somewhat fuzzy memory. It might be good to review and summarise all the uses of escapes in various GEDCOM releases…

Some *tags* may be defined as **escape-preserving tags**, each with a list of single-character **preserved escape types** each of which MUST match production `UserEscType`.

```
UserEscType ::= [A-TV-Z]
```

An **escape preservation structure** is a *tagged structure* with an *ELF schema* as its *superstructure*, *tag* "`ESC`", and no *substructures*. Its payload is composed of two *whitespace*-separated tokens; the first is the *escape-preserving tag* and the second is a concatenation of all *preserved escape types* of that *tag*; each *preserved escape type* SHOULD be included in the second token only once.

Two *escape preservation structures* MUST NOT differ only in the set of *preserved escape sequences* they define for a given *tag*.

Escape-preserving tags are included for backwards compatibility, and MUST NOT be used for new extensions.

> *Note* — The only known *escape-preserving tag* is "`DATE`", with the *preserved escape type* of "`D`"

> *Example* — The following is the only *escape preservation structure* in ELF 1.0.0:
>
> ```
> 0 HEAD
> 1 SCHMA
> ```

```
2 ESC DATE D
```

Example — The following defines *tag* `_OLD_EXTENSION` to preserve `G` and `Q` escapes:

```
0 HEAD
1 SCHMA
2 ESC _OLD_EXTENSION QG
```

The `ESC` could have equivalently been written as

```
2 ESC _OLD_EXTENSION GQ
```

or even

```
2 ESC _OLD_EXTENSION QGGQQQGGGG
```

… though that last version is needlessly redundant and verbose and is NOT RECOMMENDED.

Such a definition MUST NOT be used except as backwards compatibility support for an escape-dependent `_OLD_EXTENSION` that predates ELF 1.0.0.

Note — This specification uses *tag* and not *structure type* to indicate *escape preservation* because the main motivating case (`DATE`) applies it to all of the several *structure types* that share that *tag*.

5 Encoding with @

ELF uses the *character* U+0040 (commercial at, `@`) to encode several special cases when encoding a *tagged structure* as an *xref structure*. In particular,

- *pointers* are encoded be assigning an *xref_id* to the pointed-to *tagged structure* and using it as an *xref* in the pointing *payload*
- *characters* outside the *character encoding* are encoded as *unicode escapes*
- *escapes* that are not *preserved escapes* are removed
- `@` that are not part of *escapes* are encoded as `@@`

All of these steps involve `@`s, and MUST NOT be applied to one another's `@`s; semantically, they are applied concurrently.

During parsing, there is an inherent ambiguity when there are several contiguous `@` in the payload. These SHALL be resolved in an earliest-match-first order.

Example — The following *xref structure*'s *payloads* are split into sequences as indicated:

payload of *xref structure*	decomposed as
“name@example.com”	“name”, “@”, “example.com”
“name@@example.com”	“name”, “@@”, “example.com”
“name@@@example.com”	“name”, “@@”, “@”, “example.com”
“name@@@@example.com”	“name”, “@@”, “@@”, “example.com”
“some@#XYZ@ thing”	“some”, “@#XYZ@ ”, “thing”
“some@@#XYZ@ thing”	“some”, “@@”, “#XYZ”, “@”, “ thing”
“some@@@#XYZ@ thing”	“some”, “@@”, “@#XYZ@ ”, “thing”

5.1 Pointer conversion

If a *tagged structure* is pointed to by the *pointer*-valued *payload* of another *tagged structure*, the pointe-to *tagged structure*’s corresponding *xref structure* SHALL be given an **xref_id**, a *string* matching production `XrefID`.

```
XrefID  ::= "@" ID "@"
ID      ::= [0-9A-Z_a-z] [#x20-#x3F#x41-#x7E]*
```

It MUST NOT be the case that two different *xref structures* be given the same *xref_id*. *Conformant* implementations MUST NOT attach semantic importance to the contents of an *xref_id*.

It is RECOMMENDED that an *xref_id* be no more than 22 characters (20 characters plus the leading and trailing U+0040)

> *Note* — [GEDCOM 5.5.1] REQUIRED that *xref_id* be no more than 22 characters. ELF weakens this to a RECOMMENDATION.

Each *record* SHOULD be given an *xref_id*; each non-*record structure* SHOULD NOT; and each *serialisation metadata tagged structure* MUST NOT be given an *xref_id*.

> *Editorial note* — Since a pointed-to structure SHALL have an *xref_id* and a non-*record* MUST NOT, implicitly a *structure* SHOULD NOT point to a non-*record*. We should probably either make that explicit or remove it—the latter may make more sense as what is pointed to seems to be more a data model decision than a serialisation decision. However, GEDCOM is fairly clear that pointers to non-*records* might in the future be enabled with a non-standard *xref_id* syntax.

The *xref structure* that corresponds to a *tagged structure* with a *pointer*-valued *payload* has, as its *payload*, an **xref**: a *string* identical to the *xref_id* of the *xref structure* corresponding to the pointed-to *tagged structure*.

When parsing, if *xref payloads* are encountered that do not correspond to exactly one *xref structure*’s *xref_id*, that *payload* SHALL be converted to to a *pointer* to a *record* with *tag* “`UNDEF`”, which SHALL NOT have a *payload* nor *substructures*. It is RECOMMENDED that one such “`UNDEF`” *tagged structure* be inserted for each distinct *xref*.

> *Note* — The undefined pointer rule is designed to minimize the information loss in the event of a bad serialised input.

> *Note* — This rule does not handle pointer-to-wrong-type; information needed to determine that is not known be serialisation and thus must be handled by the data model instead.

> *Editorial note* — We could also allow pointer-to-nothing or pointer-to-multiple-things to be dropped from the dataset, and/or provide disambiguation heuristics for pointer-to-multiple-things situations. This draft does not do so as it is not obvious that the benefit is worth the complexity.

5.2 Escape preservation and removal

An **escape** is a substring of a *string*-valued *payload* of either a *tagged structure* or *xref structure* which matched production `Escape`. Its **escape type** is the portion of the *escape* that was matched by `EscType`.

```
Escape   ::= "@#" EscType EscText "@ "
EscType  ::= [A-Z]
EscText  ::= [^#xA#xD#x40]*
```

If the **escape type** is `U` (U+0055), the *escape* is a *unicode escape* and its handling is discussed in §5.3; otherwise, it is handled according to this section.

5.2.1 Serialisation

If an *escape* is in the *payload* of an *tagged structure* whose *tag* is an *escape preserving tag*, and if the escape's escape type* is in the *tag*'s set of *preserved escape types*, then the *escape* SHALL be preserved unmodified in the corresponding *xref structure*'s *payload*.

> *Example* — If a "`DATE`" *tagged structure* has *payload* "`ABT @#DJULIAN@ 1540`", its corresponding *xref structure*'s *payload* is also "`ABT @#DJULIAN@ 1540`".

Otherwise, a modification of the *escape* SHALL be placed in the *xref structure*'s *payload* which is identical to the original *escape* except that each of the two `@` SHALL each be replaced with a pair of consecutive U+0040 `@`.

> *Example* — If a "`NOTE`" *tagged structure* has *payload* "`ABT @#DJULIAN@ 1540`", its corresponding *xref structure*'s *payload* is "`ABT @@#DJULIAN@@ 1540`".

5.2.2 Parsing

If an *escape* is in the *payload* of an *xref structure* whose *tag* is an *escape preserving tag*, and the escape's escape type* is in the *tag*'s set of *preserved escape types*, the *escape* SHALL be preserved unmodified in the corresponding *tagged structure*'s *payload*.

> *Example* — If a "`DATE`" *xref structure* has *payload* "`ABT @#DJULIAN@ 1540`", its corresponding *tagged structure*'s *payload* is also "`ABT @#DJULIAN@ 1540`".

Otherwise, the *escape* SHALL be omitted from the corresponding *tagged structure*'s *payload*.

> *Example* — If a "`NOTE`" *xref structure* has *payload* "`ABT @#DJULIAN@ 1540`", its corresponding *tagged structure*'s *payload* is "`ABT 1540`".

> *Note* — The decision to remove most *escapes* is motivated in part because [GEDCOM 5.5.1] does not provide any meaning for an *escape* other than a *date escape*. This caused some ambiguity in how such escapes were handled, which ELF seeks to remove. Lacking a semantics to assign these *escapes*, ELF chooses to simply remove them. Implementations that had assigned semantics to them were actually imposing non-standard semantics to those payloads which are more accurately handled by using an alternative *ELF schema* to map those *tags* to different *structure type identifiers* with those semantics documented.

5.3 Unicode escapes

> *Note* — [GEDCOM 5.5.1] neither has a notion of *unicode escape* nor any other feature for achieving the same end. *Unicode escapes* are designed to provide a means for encoding any *character* in any *character encoding* in a way that is maximally backwards-compatible from [GEDCOM 5.5.1].

Any *character* MAY be represented with a **unicode escape** consisting of:

1. The three characters U+0040, U+0023, and U+0055 (i.e., "@#U")
2. A hexadecimal encoding of the *character*'s *code point*
3. The two characters U+0040 and U+0020 (i.e., "@ ")

A *unicode escape* MUST be used for each *character* that cannot be encoded in the target *character encoding*; and SHOULD NOT be used otherwise.

> *Editorial note* — Earlier drafts of this specification suggested using `@#U20@` in place of U+0020 when a *line*'s *payload* begins or ends with a space. Given the inherent ambiguity in the handling of *delimiters* at the ends of a *line*'s *payloads*, it is not clear if that idea was better than simply clarifying that ambiguity.

> *Example* — If a *tagged structure*'s *payload* is "`João`" and the *character encoding* is `ASCII`, the *xref structure*'s *payload* is "`Jo@#UE3@ o`" (or "`Joa@U#303@ o`" if the original used a combining diacritic).

5.4 Encoding @s

> *Editorial note* — It might be worthwhile to restrict this entire section to non-*escape preserving tags*; without that we have a (somewhat obscure) problem with the current system:
>
> Consider the *escape-preserving tag* `DATE`. A serialisation/parsing sequence applied to the *string* "`@@#Dx@@ yz`" yields
>
> 1. encoded "`@@#Dx@@ yz`"
> 2. decoded "`@#Dx@ yz`"
> 3. encoded "`@#Dx@ yz`" – not with `@@` because it matches a date escape

During serialisation, each U+0040 (@) that is not part of an *escape* SHALL be encoded as two consecutive U+0040 (@@).

> *Example* — The *tagged structure payload* "`name@example.com`" is serialised as the *xref structure payload* "`name@@example.com`"

During parsing, each consecutive pair of U+0040 (@@) SHALL be parsed as a single U+0040 (@).

> *Example* — The *xref structure payload* "`name@@example.com`" is parsed as the *tagged structure payload* "`name@example.com`"

During parsing, a lone U+0040 is left unmodified.

> *Example* — If an *xref structure*'s *payload* is "`name@example.com`", it is parsed as the *tagged structure payload* "`name@example.com`"; that in turn will be re-serialised as "`name@@example.com`".

6 Levels and lines

Each *xref structure* is encoded as a sequence of one or more *lines*.

These are of three kinds, in order:

1. The **first line** of the *xref structure*
2. Zero or more **additional lines** of the *xref structure*
3. The *lines* that encode each of the *xref structure*'s *substructures* (if any)

> *Note* — The constraint that *additional lines* come before the *lines* of *substructures* is never mentioned by [GEDCOM 5.5.1]. ELF includes it because it appears to have been universally applied by GEDCOM implementations, and some may depend upon it.

The **level** of each line is a non-negative integer. The *level* of a *first line* is 0 if the *xref structure* is a *record* or the *serialisation metadata tagged structures* with *tag* "`HEAD`" and "`TRLR`"; otherwise it is one greater than the *level* of the *first line* of its *superstructure*. The *level* of an *additional line* is one greater than the *level* of its *xref structure*'s *first line*.

Each *first line* has the same *xref_id* (if any) and *tag* as its corresponding *xref line*. Each *additional line* has no *xref_id* and either "`CONT`" or "`CONC`" as its *tag*.

> *Note* — Because an *xref structure* MUST NOT have either "`CONC`" or "`CONT`" as its *tag* (see §3.2), it is unambiguous which *lines* are *additional lines* and which *first line* they correspond to.

The *payload* of the *xref structure* is the concatenation of the *payloads* of the *first line* and all *additional lines*, with a *line break* inserted before the *payload* of each *additional line* with *tag* "`CONT`". Because the *payload* of a *line* MUST NOT contain a *line-break*, there MUST be exactly one "`CONT`"*-tagged additional line* per *line-break* in the *xref structure*'s *payload*. The number of "`CONC`"*-tagged additional lines* may be picked arbitrarily, subject to the following:

— Each *line* SHOULD be no more than 255 octets after being encoded in the *character encoding*.

> *Note* — GEDCOM REQUIRED that *lines* not exceed 255 *characters*; this does not seem to be a real restriction in most current applications, and hence has been reduced to RECOMMENDED status. We recommend bytes instead of *characters* because the implied purpose of this limit (enabling code to use fixed-width buffers) would limit by bytes, not characters.

— The *payload* of a *line* preceding a "`CONC`"*-tagged line* SHOULD NOT have an empty *payload*.
— The *payload* of a *line* preceding a "`CONC`"*-tagged line* MUST NOT end with a *delimiter*.
— A "`CONC`"*-tagged line*' *payload* SHOULD NOT begin with a *delimiter*.

> *Note* — [GEDCOM 5.5.1] is inconsistent in its discussion of leading and trailing *whitespace*.
>
> — The first of rule in the section "Grammar Rules" in Chapter 1 REQUIRES that spaces be after, not before, a `CONC` split; they (nonsensically) require the same for `CONT`s as well.
> — The grammar for `optional_line_value` in Chapter 1 allows both leading and trailing space, with no permission to remove it.
> — The definition of `CONC {CONCATENATION}` in Appendix A says an implementation MAY "look for the first non-space starting after the tag to determine the beginning of the value" and hence leading spaces MUST NOT appear.
> — The definition of `CONT {CONTINUED}` in Appendix A says an implementation MUST keep leading spaces in a `CONT` as an exception to the usual rules.

> — The definition of `NOTE_STRUCTURE` in Chapter 2 says that "most operating systems will strip off the trailing space and the space is lost in the reconstitution of the note."
>
> The RECOMMENDATIONS above are compatible with the most restrictive of these, while the REQUIREMENTS with the most limiting of them.

> *Example —*
>
> Suppose an *xref structure tag* is "`NOTE`"; it's *payload* is "`This is a test\nwith one line break`"; and its *superstructure*'s *superstructure* is a *record*. This *xref structure* requires at least two *lines* (because it contains one *line break*) and may use more. It could be serialised in many ways, such as
>
> ```
> 2 NOTE This is a test
> 3 CONT with one line break
> ```
>
> or
>
> ```
> 2 NOTE This i
> 3 CONC s a test
> 3 CONT with on
> 3 CONC e line break
> ```

— Each *line*'s *payload* MUST contain an even number of U+0040 (@). However, during parsing, this constraint SHALL NOT be enforced in any way.

> *Note* — [GEDCOM 5.5.1] gives no guidance how to handle unmatched "@", but they are relatively common in gedcom files. The above policy is intended to resolve common invalid files in an intuitive way.

> *Example —* Given the following non-*conformant* data
>
> ```
> 1 EMAI name@example.com
> 2 DATE @#DGREG
> 3 CONC ORIAN@ 2 JAN 2019
> ```
>
> a *conformant* application will concatenate these *lines* normally during parsing
>
> ```
> 1 EMAI name@example.com
> 2 DATE @#DGREGORIAN@ 2 JAN 2019
> ```
>
> creating a valid date escape in the `DATE`-*tagged extended line*. The unmatched `@` in the `EMAI`-*tagged line* is left unchanged during parsing.
>
> Upon re-serialisation, the unmatched @ in the "`EMAI`" will be doubled when converting to an *xref structure*, but the date escape will not be modified
>
> ```
> 1 EMAI name@@example.com
> ```

```
2 DATE @#DGREGORIAN@ 2 JAN 2019
```

If the serialisation decides to split either *extended line* with CONCs, it MUST NOT do so in a way that splits up the pairs of "@"s.

6.1 Error lines

A *string* line SHALL be parsed as an **error line** if it is either *unparseable* or its *line* is *too deep*.

An **unparseable** line is one that does not match production `Parseable`

```
Parseable ::= Delim? [0-9]+ (Delim XrefID)? Delim Tag (Delim [^#A#D]*)?
```

A **too deep** *line* is one that has a *level* more than one greater than its *previous level*. The **previous level** of a *line* is the *level* of the closest preceding *line* that is not an *error line* and has a *tag* that is neither "CONT" nor "CONC" nor "ERROR". If there is no such preceding *line* then the *line* is *too deep* unless its *level* is 0.

An **error line** has *tag* ERROR; and a *level* equal to the 1 + its *previous level*. If generated by a *too deep line* with an *xref_id*, the *error line* has that same *xref_id*; otherwise it has no *xref_id*. Its *payload* is

— the entire *unparseable* line, if generated by an *unparseable* line; or
— the serialised *level*, *tag*, and *payload* of the *too deep line* otherwise.

Editorial note — To do: pick one of the following:

— The *string* line that represents an *error line* is just the *error line*'s *payload*; the *level* and *tag* MUST NOT be included.
— The *string* line that represents an *error line* SHALL include the *error line*'s *level* and *tag*, like any other *line*.
— The *string* line that represents an *error line* MAY either include the *error line*'s *level* and *tag*, like any other *line*; or MAY be just the *error line*'s *payload* with no *level* or *tag*.

Note — Because "ERROR" MUST NOT be the *tag* of any *tag definition*, the *structure type definition* of the *structure* corresponding to an *error line* is the *undefined tag identifier* `elf:Undefined#ERROR`.

Example — The following invalid input

```
0 HEAD
1 CHAR UTF-8
1 SCHMA
unexpected nonsense
2 SCHMA https://fhiso.org/TR/elf-data-model/v1.0.0
0 @N1@ NOTE This is text
```

```
1 CONT more text
2 CONT still more text
1 SOUR @S1@
1 CONT attached to nothing
0 @S1@ SOUR
2 @XYZ@ NOTE text
0 TRLR
```

contains three *error lines*:

— "`unexpected nonsense`" is *unparseable* and is treated as it if were "`2 ERROR unexpected nonsense`"
— "`2 CONT still more text`" is *too deep*. It's *level* is 2 greater than the *previous level* (because the "`1 CONT more text`" line is not counted as a *previous level*) and is treated as it if were "`1 ERROR 2 CONT still more text`"
— "`2 @XYZ@ NOTE text`" has a *level* 2 greater than the *previous level* and is treated as it if were "`1 @XYZ@ ERROR 2 NOTE text`"

One other *line* will be identified as `elf:Undefined`:

— "`1 CONT attached to nothing`" is not an *additional line* as a *substructure* precedes it, and `CONT` has no *tag definition*

7 Octet string

7.1 Serialising

The sequence of *lines* representing the *dataset* is encoded as an *octet stream* by

1. Convert each *line* into a *string* consisting of
   a. Optionally any amount of *whitespace*, which SHOULD be omitted
   b. The *level* encoded as a decimal integer with no leading 0
   c. A *delimiter*, should SHOULD be a single space (U+0020)
   d. If the *line* has an *xref_id*,
      i. The *xref_id*
      ii. A *delimiter*, should SHOULD be a single space (U+0020)
   e. The *tag*
   f. If the *line* has a *payload*,
      i. A single space (U+0020)
      ii. The *payload*

> *Note* — [GEDCOM 5.5.1] does not allow parsing delimiters other than a single space in steps (c) and (d.ii) above, but other delimiters are used in some extant files.

> *Note* — [GEDCOM 5.5.1] is inconsistent in if it allows more-than-one-space for (f.i) above; in some places it clearly states that only a single space is allowed and in others it implies any delimiter may be present. Requiring a single space allows leading spaces to in *payloads*, which otherwise would require *unicode escapes* to encode.

2. Concatenate those *strings* into a single *string* with a single *line break* between each *line*. All of the *line breaks* used SHOULD be the same, and SHOULD be one of the *character* U+000A, the *character* U+000D, or the two-*character* sequence U+000D U+000A.
3. Encoding the resulting *string* into and *octet stream* using the same *character encoding* that was documented in the *serialisation metadata tagged structure* with *tag* "CHAR" (see §4.1)

7.2 Parsing

In order to parse an ELF document, an application must determine the *character encoding* to use to map the raw stream of octets read from the network or disk into *characters*. Determining it is a two-stage process, with the first stage is to determine the **detected character encoding** of the document per §7.2.1.

> *Note* — The *detected character encoding* might not be the actual *character encoding* used in the document, but if the document is *conformant*, it will be similar enough to allow a limited degree of parsing as basic ASCII *characters* will be correctly identified.

7.2.1 Detected character encoding

If a character encoding is specified via any supported external means, such as an HTTP `Content-Type` header, this SHOULD be the *detected character encoding*.

> *Example* — Suppose the ELF file was download using HTTP and the response included this header:
>
> ```
> Content-Type: text/plain; charset=UTF-8
> ```
>
> If an application supports taking the *detected character encoding* from an HTTP `Content-Type` header, the *detected character encoding* SHOULD be UTF-8.
>
> Note that the use of the MIME type `text/plain` is NOT RECOMMENDED for ELF. It is used here purely as an example.

Otherwise, if the document begins with a byte-order mark (U+FEFF) encoded in UTF-8, or UTF-16 of either endianness, this encoding SHALL be the *detected character encoding*. The byte-order mark is removed from the data stream before further processing.

Otherwise, if the document begins with the digit 0 (U+0030) encoded in UTF-16 of either endianness, this encoding SHALL be the *detected character encoding*.

> *Note* — The digit 0 is tested for because an ELF file MUST begin with the *line* "`0 HEAD`".

Otherwise, applications MAY try to detect other character encodings by examining the octet stream, but it is NOT RECOMMENDED that they do so.

> *Note* — One situation where it might be desirable to try to detect another encoding is if the application needs to support (as an extension) a character encoding like EBCDIC which is not compatible with ASCII.

Otherwise, there is no *detected character encoding*.

> *Note* — These cases can be summarised as follows:
>
> | Initial octets | Detected character encoding |
> |---|---|
> | EF BB BF | UTF-8 (with byte-order mark) |
> | FF FE | UTF-16, little endian (with byte-order mark) |
> | FE FF | UTF-16, big endian (with byte-order mark) |
> | 30 00 | UTF-16, little endian (without byte-order mark) |
> | 00 30 | UTF-16, big endian (without byte-order mark) |
> | Otherwise | None |

7.2.2 Character encoding

A prefix of the *octet stream* shall be decoded using the *detected character encoding*, or an unspecified ASCII-compatible encoding if there is no *detected character encoding*. This prefix is parsed into *lines*, stopping at the second instance of a *line* with *level* 0. If a *line* with *level* 1 and *tag* `CHAR` was found, its *payload* is the **specified character encoding** of the document.

If there is a *specified character encoding*, it SHALL be used as the *character encoding* of the octet stream. Otherwise, if there is a *detected character encoding*, it SHALL be used as the *character encoding* of the octet stream. Otherwise, the *character encoding* SHALL be determined to be ANSEL.

8 References

8.1 Normative references

[ANSEL]

NISO (National Information Standards Organization). *ANSI/NISO Z39.47-1993. Extended Latin Alphabet Coded Character Set for Bibliographic Use*. 1993. (See http://www.niso.org/apps/group_public/project/details.php?project_id=10.) Standard withdrawn, 2013.

[Basic Concepts]

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

[ASCII]

ANSI (American National Standards Institute). *ANSI X3.4-1986. Coded Character Sets – 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII)*. 1986.

[ISO 10646]

ISO (International Organization for Standardization). *ISO/IEC 10646:2014. Information technology — Universal Coded Character Set (UCS)*. 2014.

[RFC 2119]

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

[XML]

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

8.2 Other references

[GEDCOM 5.5.1]

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

[GEDCOM 5.5]

The Church of Jesus Christ of Latter-day Saints. *The GEDCOM Standard*, release 5.5. 1996.

[XML Names]

W3 (World Wide Web Consortium). *Namespaces in XML 1.1*, 2nd edition. Tim Bray, Dave Hollander, Andrew Layman and Richard Tobin, eds., 2006. W3C Recommendation. See https://www.w3.org/TR/xml-names11/.

[ELF-DataModel]

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

9 Appendix A: Default Schema

The following is a minimal ELF file with the default *ELF Schema*, which includes all *tag definitions* and *supertype definitions* listed in [Elf-DataModel].

```
0 HEAD
1 CHAR UTF-8
1 GEDC
2 VERS 5.5.1
2 FORM LINEAGE-LINKED
2 ELF 1.0.0
1 SCHMA
2 PRFX elf https://terms.fhiso.org/elf/
2 PRFX elfm https://terms.fhiso.org/elf/metadata/
2 ESC DATE D
2 IRI elf:ADDRESS
3 TAG ADDR elf:Agent elf:Event
2 IRI elf:ADDRESS_CITY
3 TAG CITY elf:ADDRESS
2 IRI elf:ADDRESS_COUNTRY
3 TAG CTRY elf:ADDRESS
2 IRI elf:ADDRESS_EMAIL
3 TAG EMAIL elf:Agent
3 TAG EMAI elf:Agent
2 IRI elf:ADDRESS_FAX
3 TAG FAX elf:Agent
2 IRI elf:ADDRESS_LINE1
3 TAG ADR1 elf:ADDRESS
2 IRI elf:ADDRESS_LINE2
3 TAG ADR2 elf:ADDRESS
2 IRI elf:ADDRESS_LINE3
3 TAG ADR3 elf:ADDRESS
2 IRI elf:ADDRESS_POSTAL_CODE
3 TAG POST elf:ADDRESS
2 IRI elf:ADDRESS_STATE
3 TAG STAE elf:ADDRESS
2 IRI elf:ADDRESS_WEB_PAGE
3 TAG WWW elf:Agent
2 IRI elf:ADOPTED_BY_WHICH_PARENT
3 TAG ADOP elf:ADOPTIVE_FAMILY
2 IRI elf:ADOPTION
3 ISA elf:IndividualEvent
3 TAG ADOP elf:INDIVIDUAL_RECORD
```

```
2 IRI elf:ADOPTIVE_FAMILY
3 TAG FAMC elf:ADOPTION
2 IRI elf:ADULT_CHRISTENING
3 ISA elf:IndividualEvent
3 TAG CHRA elf:INDIVIDUAL_RECORD
2 IRI elf:AGE_AT_EVENT
3 TAG AGE elf:IndividualEvent elf:Parent1Age elf:Parent2Age
2 IRI elf:ALIAS_POINTER
3 TAG ALIA elf:INDIVIDUAL_RECORD
2 IRI elf:ANCESTOR_INTEREST_POINTER
3 TAG ANCI elf:INDIVIDUAL_RECORD
2 IRI elf:ANNULMENT
3 ISA elf:FamilyEvent
3 TAG ANUL elf:FAM_RECORD
2 IRI elf:ASSOCIATION_STRUCTURE
3 TAG ASSO elf:INDIVIDUAL_RECORD
2 IRI elf:ATTRIBUTE_DESCRIPTOR
3 ISA elf:IndividualAttribute
3 TAG FACT elf:INDIVIDUAL_RECORD
2 IRI elf:AUTOMATED_RECORD_ID
3 TAG RIN elf:Record
2 IRI elf:Agent
2 IRI elf:BAPTISM
3 ISA elf:IndividualEvent
3 TAG BAPM elf:INDIVIDUAL_RECORD
2 IRI elf:BAR_MITZVAH
3 ISA elf:IndividualEvent
3 TAG BARM elf:INDIVIDUAL_RECORD
2 IRI elf:BAS_MITZVAH
3 ISA elf:IndividualEvent
3 TAG BASM elf:INDIVIDUAL_RECORD
2 IRI elf:BINARY_OBJECT
3 TAG BLOB elf:MULTIMEDIA_RECORD
2 IRI elf:BIRTH
3 ISA elf:IndividualEvent
3 TAG BIRT elf:INDIVIDUAL_RECORD
2 IRI elf:BLESSING
3 ISA elf:IndividualEvent
3 TAG BLES elf:INDIVIDUAL_RECORD
2 IRI elf:BURIAL
3 ISA elf:IndividualEvent
3 TAG BRI elf:INDIVIDUAL_RECORD
2 IRI elf:CASTE_NAME
```

```
3 ISA elf:IndividualAttribute
3 TAG CAST elf:INDIVIDUAL_RECORD
2 IRI elf:CAUSE_OF_EVENT
3 TAG CAUS elf:Event
2 IRI elf:CENSUS#Family
3 ISA elf:FamilyEvent
3 TAG CENS elf:FAM_RECORD
2 IRI elf:CENSUS#Individual
3 ISA elf:IndividualEvent
3 TAG CENS elf:INDIVIDUAL_RECORD
2 IRI elf:CERTAINTY_ASSESSMENT
3 TAG QUAY elf:SOURCE_CITATION
2 IRI elf:CHANGE_DATE
3 TAG CHAN elf:Record
2 IRI elf:CHANGE_DATE_DATE
3 TAG DATE elf:CHANGE_DATE
2 IRI elf:CHILD_LINKAGE_STATUS
3 TAG STAT elf:CHILD_TO_FAMILY_LINK
2 IRI elf:CHILD_POINTER
3 TAG CHIL elf:FAM_RECORD
2 IRI elf:CHILD_TO_FAMILY_LINK
3 TAG FAMC elf:INDIVIDUAL_RECORD
2 IRI elf:CHRISTENING
3 ISA elf:IndividualEvent
3 TAG CHR elf:INDIVIDUAL_RECORD
2 IRI elf:CONFIRMATION
3 ISA elf:IndividualEvent
3 TAG CONF elf:INDIVIDUAL_RECORD
2 IRI elf:CONTINUED_BINARY_OBJECT
3 TAG OBJE elf:MULTIMEDIA_RECORD
2 IRI elf:COPYRIGHT_GEDCOM_FILE
3 TAG COPR elf:Metadata
2 IRI elf:COPYRIGHT_SOURCE_DATA
3 TAG COPR elf:NAME_OF_SOURCE_DATA
2 IRI elf:COUNT_OF_CHILDREN#Family
3 TAG NCHI elf:FAM_RECORD
2 IRI elf:COUNT_OF_CHILDREN#Individual
3 ISA elf:IndividualAttribute
3 TAG NCHI elf:INDIVIDUAL_RECORD
2 IRI elf:COUNT_OF_MARRIAGES
3 ISA elf:IndividualAttribute
3 TAG NMR elf:INDIVIDUAL_RECORD
2 IRI elf:CREMATION
```

```
3 ISA elf:IndividualEvent
3 TAG CREM elf:INDIVIDUAL_RECORD
2 IRI elf:DATE_PERIOD
3 TAG DATE elf:EVENTS_RECORDED
2 IRI elf:DATE_VALUE
3 TAG DATE elf:Event
2 IRI elf:DEATH
3 ISA elf:IndividualEvent
3 TAG DEAT elf:INDIVIDUAL_RECORD
2 IRI elf:DEFAULT_PLACE_FORMAT
3 TAG PLAC elf:Metadata
2 IRI elf:DESCENDANT_INTEREST_POINTER
3 TAG DESI elf:INDIVIDUAL_RECORD
2 IRI elf:DESCRIPTIVE_TITLE
3 TAG TITL elf:MULTIMEDIA_FILE_REFERENCE elf:MULTIMEDIA_LINK
  → elf:MULTIMEDIA_RECORD
2 IRI elf:DIVORCE
3 ISA elf:FamilyEvent
3 TAG DIV elf:FAM_RECORD
2 IRI elf:DIVORCE_FILED
3 ISA elf:FamilyEvent
3 TAG DIVF elf:FAM_RECORD
2 IRI elf:DOCUMENT_SOURCE
3 TAG SOUR elf:Metadata
2 IRI elf:Document
2 IRI elf:EMIGRATION
3 ISA elf:IndividualEvent
3 TAG EMIG elf:INDIVIDUAL_RECORD
2 IRI elf:ENGAGEMENT
3 ISA elf:FamilyEvent
3 TAG ENGA elf:FAM_RECORD
2 IRI elf:ENTRY_RECORDING_DATE
3 TAG DATE elf:SOURCE_CITATION_DATA
2 IRI elf:EVENT#Family
3 ISA elf:FamilyEvent
3 TAG EVEN elf:FAM_RECORD
2 IRI elf:EVENT#Individual
3 ISA elf:IndividualEvent
3 TAG EVEN elf:INDIVIDUAL_RECORD
2 IRI elf:EVENTS_RECORDED
3 TAG EVEN elf:SOURCE_RECORD_DATA
2 IRI elf:EVENT_OR_FACT_CLASSIFICATION
3 TAG TYPE elf:Event
```

```
2 IRI elf:EVENT_TYPE_CITED_FROM
3 TAG EVEN elf:SOURCE_CITATION
2 IRI elf:Event
2 IRI elf:FAM_RECORD
3 ISA elf:Record
3 TAG FAM elf:Document
2 IRI elf:FILE_NAME
3 TAG FILE elf:Metadata
2 IRI elf:FIRST_COMMUNION
3 ISA elf:IndividualEvent
3 TAG FCOM elf:INDIVIDUAL_RECORD
2 IRI elf:FamilyEvent
3 ISA elf:Event
2 IRI elf:GEDCOM_CONTENT_DESCRIPTION
3 TAG NOTE elf:Metadata
2 IRI elf:GEDCOM_FORM
3 TAG FORM elf:GEDCOM_FORMAT
2 IRI elf:GEDCOM_FORMAT
3 TAG GEDC elf:Metadata
2 IRI elf:GRADUATION
3 ISA elf:IndividualEvent
3 TAG GRAD elf:INDIVIDUAL_RECORD
2 IRI elf:IMMIGRATION
3 ISA elf:IndividualEvent
3 TAG IMMI elf:INDIVIDUAL_RECORD
2 IRI elf:INDIVIDUAL_RECORD
3 ISA elf:Record
3 TAG INDI elf:Document
2 IRI elf:IndividualAttribute
3 ISA elf:Event
2 IRI elf:IndividualEvent
3 ISA elf:Event
2 IRI elf:LANGUAGE_OF_TEXT
3 TAG LANG elf:Metadata
2 IRI elf:LANGUAGE_PREFERENCE
3 TAG LANG elf:SUBMITTER_RECORD
2 IRI elf:MAP_COORDINATES
3 TAG MAP elf:PLACE_STRUCTURE
2 IRI elf:MARRIAGE
3 ISA elf:FamilyEvent
3 TAG MARR elf:FAM_RECORD
2 IRI elf:MARRIAGE_BANN
3 ISA elf:FamilyEvent
```

```
3 TAG MARB elf:FAM_RECORD
2 IRI elf:MARRIAGE_CONTRACT
3 ISA elf:FamilyEvent
3 TAG MARC elf:FAM_RECORD
2 IRI elf:MARRIAGE_LICENSE
3 ISA elf:FamilyEvent
3 TAG MARL elf:FAM_RECORD
2 IRI elf:MARRIAGE_SETTLEMENT
3 ISA elf:FamilyEvent
3 TAG MARS elf:FAM_RECORD
2 IRI elf:MULTIMEDIA_FILE_REFERENCE
3 TAG FILE elf:MULTIMEDIA_LINK elf:MULTIMEDIA_RECORD
2 IRI elf:MULTIMEDIA_FORMAT
3 TAG FORM elf:MULTIMEDIA_FILE_REFERENCE elf:MULTIMEDIA_LINK
  → elf:MULTIMEDIA_RECORD
2 IRI elf:MULTIMEDIA_LINK
3 TAG OBJE elf:Event elf:FAM_RECORD elf:INDIVIDUAL_RECORD
  → elf:SOURCE_CITATION elf:SOURCE_RECORD elf:SUBMITTER_RECORD
2 IRI elf:MULTIMEDIA_RECORD
3 ISA elf:Record
3 TAG OBJE elf:Document
2 IRI elf:Metadata
2 IRI elf:NAME_OF_BUSINESS
3 ISA elf:Agent
3 TAG CORP elf:DOCUMENT_SOURCE
2 IRI elf:NAME_OF_PRODUCT
3 TAG NAME elf:DOCUMENT_SOURCE
2 IRI elf:NAME_OF_REPOSITORY
3 TAG NAME elf:REPOSITORY_RECORD
2 IRI elf:NAME_OF_SOURCE_DATA
3 TAG DATA elf:DOCUMENT_SOURCE
2 IRI elf:NAME_PHONETIC_VARIATION
3 ISA elf:PersonalName
3 TAG FONE elf:PERSONAL_NAME_STRUCTURE
2 IRI elf:NAME_PIECE_GIVEN
3 TAG GIVN elf:PersonalName
2 IRI elf:NAME_PIECE_NICKNAME
3 TAG NICK elf:PersonalName
2 IRI elf:NAME_PIECE_PREFIX
3 TAG NPFX elf:PersonalName
2 IRI elf:NAME_PIECE_SUFFIX
3 TAG NSFX elf:PersonalName
2 IRI elf:NAME_PIECE_SURNAME
```

```
3 TAG SURN elf:PersonalName
2 IRI elf:NAME_PIECE_SURNAME_PREFIX
3 TAG SPFX elf:PersonalName
2 IRI elf:NAME_ROMANIZED_VARIATION
3 ISA elf:PersonalName
3 TAG ROMN elf:PERSONAL_NAME_STRUCTURE
2 IRI elf:NAME_TYPE
3 TAG TYPE elf:PERSONAL_NAME_STRUCTURE
2 IRI elf:NATIONAL_ID_NUMBER
3 ISA elf:IndividualAttribute
3 TAG IDNO elf:INDIVIDUAL_RECORD
2 IRI elf:NATIONAL_OR_TRIBAL_ORIGIN
3 ISA elf:IndividualAttribute
3 TAG NATI elf:INDIVIDUAL_RECORD
2 IRI elf:NATURALIZATION
3 ISA elf:IndividualEvent
3 TAG NATU elf:INDIVIDUAL_RECORD
2 IRI elf:NOBILITY_TYPE_TITLE
3 ISA elf:IndividualAttribute
3 TAG TITL elf:INDIVIDUAL_RECORD
2 IRI elf:NOTE_RECORD
3 ISA elf:Record
3 TAG NOTE elf:Document
2 IRI elf:NOTE_STRUCTURE
3 TAG NOTE elf:ASSOCIATION_STRUCTURE elf:CHANGE_DATE
  → elf:CHILD_TO_FAMILY_LINK elf:Event elf:PLACE_STRUCTURE
  → elf:PersonalName elf:Record elf:SOURCE_CITATION elf:SOURCE_RECORD_DATA
  → elf:SOURCE_REPOSITORY_CITATION elf:SPOUSE_TO_FAMILY_LINK
2 IRI elf:OCCUPATION
3 ISA elf:IndividualAttribute
3 TAG OCCU elf:INDIVIDUAL_RECORD
2 IRI elf:ORDINATION
3 ISA elf:IndividualEvent
3 TAG ORDN elf:INDIVIDUAL_RECORD
2 IRI elf:PARENT1_POINTER
3 ISA elf:ParentPointer
3 TAG HUSB elf:FAM_RECORD
2 IRI elf:PARENT2_POINTER
3 ISA elf:ParentPointer
3 TAG WIFE elf:FAM_RECORD
2 IRI elf:PEDIGREE_LINKAGE_TYPE
3 TAG PEDI elf:CHILD_TO_FAMILY_LINK
2 IRI elf:PERSONAL_NAME_STRUCTURE
```

```
3 ISA elf:PersonalName
3 TAG NAME elf:INDIVIDUAL_RECORD
2 IRI elf:PHONETIC_TYPE
3 TAG TYPE elf:NAME_PHONETIC_VARIATION elf:PLACE_PHONETIC_VARIATION
2 IRI elf:PHONE_NUMBER
3 TAG PHON elf:Agent
2 IRI elf:PHYSICAL_DESCRIPTION
3 ISA elf:IndividualAttribute
3 TAG DSCR elf:INDIVIDUAL_RECORD
2 IRI elf:PLACE_HIERARCHY
3 TAG FORM elf:DEFAULT_PLACE_FORMAT elf:PLACE_STRUCTURE
2 IRI elf:PLACE_LATITUDE
3 TAG LATI elf:MAP_COORDINATES
2 IRI elf:PLACE_LONGITUDE
3 TAG LONG elf:MAP_COORDINATES
2 IRI elf:PLACE_PHONETIC_VARIATION
3 TAG FONE elf:PLACE_STRUCTURE
2 IRI elf:PLACE_ROMANIZED_VARIATION
3 TAG ROMN elf:PLACE_STRUCTURE
2 IRI elf:PLACE_STRUCTURE
3 TAG PLAC elf:Event
2 IRI elf:POSSESSIONS
3 ISA elf:IndividualAttribute
3 TAG PROP elf:INDIVIDUAL_RECORD
2 IRI elf:PROBATE
3 ISA elf:IndividualEvent
3 TAG PROB elf:INDIVIDUAL_RECORD
2 IRI elf:PUBLICATION_DATE
3 TAG DATE elf:NAME_OF_SOURCE_DATA
2 IRI elf:Parent1Age
3 TAG HUSB elf:FamilyEvent
2 IRI elf:Parent2Age
3 TAG WIFE elf:FamilyEvent
2 IRI elf:ParentPointer
2 IRI elf:PersonalName
2 IRI elf:RECEIVING_SYSTEM_NAME
3 TAG DEST elf:Metadata
2 IRI elf:RELATION_IS_DESCRIPTOR
3 TAG RELA elf:ASSOCIATION_STRUCTURE
2 IRI elf:RELIGIOUS_AFFILIATION
3 TAG RELI elf:Event
2 IRI elf:RELIGIOUS_AFFILIATION#Individual
3 ISA elf:IndividualAttribute
```

```
3 TAG RELI elf:INDIVIDUAL_RECORD
2 IRI elf:REPOSITORY_RECORD
3 ISA elf:Agent
3 ISA elf:Record
3 TAG REPO elf:Document
2 IRI elf:RESIDENCE
3 ISA elf:FamilyEvent
3 TAG RESI elf:FAM_RECORD
2 IRI elf:RESIDES_AT
3 ISA elf:IndividualAttribute
3 TAG RESI elf:INDIVIDUAL_RECORD
2 IRI elf:RESPONSIBLE_AGENCY
3 TAG AGNC elf:Event elf:SOURCE_RECORD_DATA
2 IRI elf:RESTRICTION_NOTICE
3 TAG RESN elf:Event elf:FAM_RECORD elf:INDIVIDUAL_RECORD
2 IRI elf:RETIREMENT
3 ISA elf:IndividualEvent
3 TAG RETI elf:INDIVIDUAL_RECORD
2 IRI elf:ROLE_IN_EVENT
3 TAG ROLE elf:EVENT_TYPE_CITED_FROM
2 IRI elf:ROMANIZED_TYPE
3 TAG TYPE elf:NAME_ROMANIZED_VARIATION elf:PLACE_ROMANIZED_VARIATION
2 IRI elf:Record
2 IRI elf:SCHOLASTIC_ACHIEVEMENT
3 ISA elf:IndividualAttribute
3 TAG EDUC elf:INDIVIDUAL_RECORD
2 IRI elf:SEX_VALUE
3 TAG SEX elf:INDIVIDUAL_RECORD
2 IRI elf:SOCIAL_SECURITY_NUMBER
3 ISA elf:IndividualAttribute
3 TAG SSN elf:INDIVIDUAL_RECORD
2 IRI elf:SOURCE_CALL_NUMBER
3 TAG CALN elf:SOURCE_REPOSITORY_CITATION
2 IRI elf:SOURCE_CITATION
3 TAG SOUR elf:ASSOCIATION_STRUCTURE elf:Event elf:FAM_RECORD
  → elf:INDIVIDUAL_RECORD elf:PersonalName
2 IRI elf:SOURCE_CITATION_DATA
3 TAG DATA elf:SOURCE_CITATION
2 IRI elf:SOURCE_DESCRIPTIVE_TITLE
3 TAG TITL elf:SOURCE_RECORD
2 IRI elf:SOURCE_FILED_BY_ENTRY
3 TAG ABBR elf:SOURCE_RECORD
2 IRI elf:SOURCE_JURISDICTION_PLACE
```

```
3 TAG PLAC elf:EVENTS_RECORDED
2 IRI elf:SOURCE_MEDIA_TYPE
3 TAG MEDI elf:MULTIMEDIA_FORMAT elf:SOURCE_CALL_NUMBER
2 IRI elf:SOURCE_ORIGINATOR
3 TAG AUTH elf:SOURCE_RECORD
2 IRI elf:SOURCE_PUBLICATION_FACTS
3 TAG PUBL elf:SOURCE_RECORD
2 IRI elf:SOURCE_RECORD
3 ISA elf:Record
3 TAG SOUR elf:Document
2 IRI elf:SOURCE_RECORD_DATA
3 TAG DATA elf:SOURCE_RECORD
2 IRI elf:SOURCE_REPOSITORY_CITATION
3 TAG REPO elf:SOURCE_RECORD
2 IRI elf:SPOUSE_TO_FAMILY_LINK
3 TAG FAMS elf:INDIVIDUAL_RECORD
2 IRI elf:SUBMITTER_NAME
3 TAG NAME elf:SUBMITTER_RECORD
2 IRI elf:SUBMITTER_POINTER
3 TAG SUBM elf:FAM_RECORD elf:INDIVIDUAL_RECORD elf:Metadata
2 IRI elf:SUBMITTER_RECORD
3 ISA elf:Agent
3 ISA elf:Record
3 TAG SUBM elf:Document
2 IRI elf:Structure
2 IRI elf:TEXT_FROM_SOURCE
3 TAG TEXT elf:SOURCE_CITATION elf:SOURCE_CITATION_DATA elf:SOURCE_RECORD
2 IRI elf:TIME_VALUE
3 TAG TIME elf:CHANGE_DATE_DATE elf:TRANSMISSION_DATE
2 IRI elf:TRANSMISSION_DATE
3 TAG DATE elf:Metadata
2 IRI elf:USER_REFERENCE_NUMBER
3 TAG REFN elf:Record
2 IRI elf:USER_REFERENCE_TYPE
3 TAG TYPE elf:USER_REFERENCE_NUMBER
2 IRI elf:VERSION_NUMBER
3 TAG VERS elf:DOCUMENT_SOURCE elf:GEDCOM_FORMAT
2 IRI elf:WHERE_WITHIN_SOURCE
3 TAG PAGE elf:SOURCE_CITATION
2 IRI elf:WILL
3 ISA elf:IndividualEvent
3 TAG WILL elf:INDIVIDUAL_RECORD
2 IRI elf:WITHIN_FAMILY
```

```
3 TAG FAMC elf:BIRTH elf:CHRISTENING
1 SOUR https://fhiso.org/elf/
1 NOTE This file was automatically generated from data-model.md
2 CONT by schema-maker.py at 2019-01-04T17:32:07.909701
1 SUBM @fhiso_elf1@
0 @fhiso_elf1@ SUBM
1 NAME FHISO Extended Legacy Format, version 1
0 TRLR
```