Basic Concepts for Genealogical Standards

16 March 2018

> *Editorial note —* This is a **first public draft** of a standard covering basic concepts that are expected to be used in multiple FHISO standards. This document is not endorsed by the FHISO membership, and may be updated, replaced or obsoleted by other documents at any time.
>
> The public tsc-public@fhiso.org mailing list is the preferred place for comments, discussion and other feedback on this draft.
>
> | | |
> |---|---|
> | Latest public version: | `https://fhiso.org/TR/basic-concepts` |
> | This version: | `https://fhiso.org/TR/basic-concepts-20180316` |

FHISO's **Basic Concepts for Genealogical Standards** standard defines various low-level concepts that will be used in many FHISO standards, and whose definitions do not logically belong in any one particular higher-level standard.

The definition of a *string* which is used in multiple FHISO standards is given in §2 of this standard, together with various related concepts such as *characters* and *whitespace*, and §3 defines briefly how FHISO standards use *language tags*. *Terms* are defined in §4 as a form of extensible identifier using IRIs, and §4.1 discusses information that may be retrieved from these IRIs. The notion of a *datatype* is defined in §6, which also includes details on how to specify a new *datatype*.

The concepts of a *classes* and *properties* are defined in §5. They provide an infrastructure for defining extensions to FHISO standards and new, compatible standards in such a way that applications can use a *discovery* mechanism to find out about unknown components, allowing them to be processed. The facilities in these sections will primarily be of use to parties defining extensions or implementing *discovery*.

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.

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 *prefix notation*, as defined in §4.3 of this standard, when discussing specific *terms*. The following *prefix* bindings are assumed in this standard:

`rdf`	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
`rdfs`	`http://www.w3.org/2000/01/rdf-schema#`
`xsd`	`http://www.w3.org/2001/XMLSchema#`
`types`	`https://terms.fhiso.org/types/`

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

2 Characters and strings

> *Editorial note* — The concepts related to *strings* were originally defined in the CEV Concepts draft. This section has been moved here to be more generally usable.

Characters are specified by reference to their **code point** number in [ISO 10646], without regard to any particular character encoding. In this standard, *characters* may be identified in this standard by their hexadecimal *code point* prefixed with "U+".

> *Note* — The character encoding is a property of the serialisation, and not defined in this standard. Non-Unicode encodings are not precluded, so long as it is defined how characters in that encoding corresponds to Unicode characters.

Characters MUST match the `Char` production from [XML].

```
Char  ::=  [#1-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]
```

> *Note* — This includes all *code points* except the null character, surrogates (which are reserved for encodings such as UTF-16 and not characters in their own right), and the invalid characters U+FFFE and U+FFFF.

A **string** is a sequence of zero or more *characters*, and SHOULD only be used to encode textual data.

> *Note* — This definition of a *string* is identical to the definition of the `string` datatype defined in [XSD Pt2], used in many XML and Semantic Web technologies.
>
> This definition of a *string* differs very slightly from JSON's definition of a string, as defined in [RFC 7159], as a JSON string may include the null *character* (U+0000). This is the only difference between a JSON string and FHISO's definition of a *string*. As a *string* SHOULD NOT be used to contain raw binary data, this difference is not anticipated to cause a problem. If an application needs to store binary data in *string*, it SHOULD encode it in a textual form, for example with the Base64 data encoding scheme defined in [RFC 4648].

Applications MAY convert any *string* into Unicode Normalization Form C, as defined in any version of Unicode Standard Annex #15 [UAX 15].

> *Note* — Normalization Form C and Normalization Form D allow easier searching, sorting and comparison of *strings* by picking a canonical representation of accented characters. The conversion between Normalization Forms C and D is lossless and therefore reversible, but the initial conversion to either form is not reversible. This allows a *conformant* application to normalise *strings* internally and not retain the unnormalised form; however, an application doing so MUST ensure the *string* is in Normalization Form C upon export, this being the more usual form for use in documents.

Characters matching the `RestrictedChar` production from [XML] SHOULD NOT appear in *strings*, and applications MAY process such characters in an implementation-defined manner or reject *strings* containing them.

```
RestrictedChar  ::=  [#x1-#x8] | [#xB-#xC] | [#xE-#x1F]
                       | [#x7F-#x84] | [#x86-#x9F]
```

> *Note* — This includes all C0 and C1 control characters except tab (U+0009), line feed (U+000A), carriage return (U+000D) and next line (U+0085).

> *Example* — As *conformant* applications can process C1 control characters in an implementation-defined manner, they can opt to handle Windows-1252 quotation marks in data masquerading as Unicode. Applications MUST NOT treat non-ASCII characters (other than C1 control characters) as ANSEL, the character set properly used in [GEDCOM], as [ANSEL]'s non-ASCII characters do not correspond to `RestrictedChars`.

Conformant applications MUST be able to store and process *strings* containing arbitrary *characters* other than those matching the `RestrictedChar`. In particular, applications MUST be able to handle *characters* which correspond to unassigned Unicode *code points* as they may be assigned in future versions of [ISO 10646]. Applications MUST also be able to handle *characters* outside Unicode's Basic Multilingual Plane — that is, *characters* with a *code point* of U+10000 or higher.

> *Note* — This means applications MUST NOT represent *strings* internally in the UCS-2 encoding which does not accommodate *characters* outside the Basic Multilingual Plane. The UTF-16 encoding defined in §2.6 of [ISO 10646] provides a 16-bit encoding that is backwards compatible with UCS-2 but allows arbitrary *characters* to be represented through the use of Unicode surrogate pairs.

Whitespace is defined as a sequence of one or more space *characters*, carriage returns, line feeds, or tabs. It matches the production `S` from [XML].

```
S  ::=  (#x20 | #x9 | #xD | #xA)+
```

> *Note* — This definition only includes common ASCII whitespace *characters* and does not include every *character* in [ISO 10646] that could be considered to be a whitespace. For example, the vertical tab (U+000B), no-break space (U+00A0) and em space (U+2003) are all excluded.

Whitespace normalisation is the process of discarding any leading or trailing *whitespace*, and replacing other *whitespace* with a single space (U+0020) *character*.

> *Note* — The definition of *whitespace normalisation* is identical to that in [XML].

In the event of a difference between the definitions of the `Char`, `RestrictedChar` and `S` productions given here and those in [XML], the definitions in the latest edition of XML 1.1 specification are definitive.

3 Language tags

> *Editorial note* — The material in this section is new in this draft.

A **language tag** is a *string* that is used to represent a human language, and where appropriate the script and regional variant or dialect used. They are commonly used to tag other *strings* to identify their language in a machine-readable manner.

The *language tag* SHALL match the `Language-Tag` production from [RFC 5646], or from any future RFC published by the IEFT that obsoletes [RFC 5646] (hereinafter referred to as RFC 5646's successor RFC), and SHOULD be valid, as defined in §2.2.9 of [RFC 5646].

Valid *language tags* have the meaning that is assigned to them by [RFC 5646] and any successor RFC. Applications MAY discard any *language tag* that is not well-formed and replace it with `und`, meaning a undetermined language, but MUST NOT discard any *language tag* that is well-formed even if it is not valid.

> *Note* — [RFC 5646] says that to be valid, a *language tag* MUST consist of tags that have been registered in the [IANA Lang Subtags] registry. This is freely available online in a machine-readable form defined in §3.1.1 of [RFC 5646], and gives the meaning of every tag. Currently it includes:
>
> — two-letter language tags from [ISO 639-1];
> — three-letter language tags from [ISO 639-2] (the "terminology" codes where they differ from the "bibliographic" codes), [ISO 639-3] and [ISO 639-5] for languages with no two-letter code;
> — four-letter script tags from [ISO 15924];
> — two-letter country codes currently assigned in [ISO 3166-1], together with certain formerly assigned or reserved codes;
> — three-digit codes for supranational geographical areas and exceptionally countries from [UN M.49]; and
> — a small number of legacy tags that have been grandfathered into the scheme.
>
> The meanings of codes in the source ISO standards may change over time, but the procedure set out in §3.4 of [RFC 5646] governing the addition of tags to [IANA Lang Subtags] ensures the meanings there stable. This particularly affects [ISO 3166-1] country codes which historically have been reused, and may result in a gradual divergence between and [IANA Lang Subtags]. Applications SHOULD therefore avoid using [ISO 3166-1] codes that have not been registered in [IANA Lang Subtags].

> *Example* — A *string* tagged with the *language tag* `hu-CS` MUST be interpreted by a *conformant* application as being in the Hungarian language localised for use in the former state of Serbia and Montenegro, because this is how `hu` and `CS` are listed in [IANA Lang Subtags]. The code `CS` is perhaps better known as representing the former state of Czechoslovakia

> and appears in older lists of [ISO 3166-1] country codes as such, but neither IANA nor FHISO recognise this former meaning.
>
> This is one of five country codes whose meaning has materially changed in [ISO 3166-1], the other four being `AI`, `BQ`, `GE` and `SK`. In each case, because the reuse occurred before the creation of [IANA Lang Subtags], it is the current meaning that is listed in [IANA Lang Subtags]. If there is further reuse of country codes in the future, [RFC 5646] requires that the current meaning of the tag be retained and a numeric code be given to the new country in [IANA Lang Subtags].

A *conformant* application MAY convert any *language tag* into its canonical form, as defined by §4.5 of [RFC 5646] or an equivalent section of a successor RFC.

> *Note* — The chief purpose of canonical form is to replace deprecated language codes and other subtags with the value found in the `Preferred-Value` field in [IANA Lang Subtags]. It never result in the removal of script subtag, even when they are the default script for the language as defined by a `Suppress-Script` field.

> *Example* — The *language tag* `iw` is listed in [IANA Lang Subtags] as a deprecated language code for Hebrew which has now been removed from [ISO 639-1]. Its `Preferred-Value` field is `he`, so an application MAY replace `iw` with `he`.

A *conformant* application MAY alter a *language tag* in any other way that leaves its canonical form unchanged when compared in a case-insensitive manner.

> *Note* — Such changes are permitted for three reasons. First, it allows applications to revert new tags to older deprecated forms when exporting data to an older application. Secondly, it allows applications to remain *conformant* even if they are basing conversions on an outdated copy of the [IANA Lang Subtags] registry. This is because §3.4 of [RFC 5646] only allows certain compatible changes to the registry. Thirdly, it allows applications to apply the conventional capitalisation of *language tags* defined in §2.1.1 of [RFC 5646].

A *string* which is accompanied by a *language tag* which identifies the language in which the *string* is written is called a **language-tagged string**.

> *Note* — The *language tag* is not itself part of *string*, but is stored alongside it.

4 Terms

> *Editorial note* — The concept of a *term* was originally defined in the CEV Concepts draft. It has been moved here to be more generally usable.

A **term** is a form of identifier used in FHISO standards to represent a concept which it is useful to be able to reference. A *term* consists of a unique, machine-readable identifier, known as the **term name**, paired with a clearly-defined meaning for the concept or idea that it represents. *Term names* SHALL take the form of an IRI matching the `IRI` production in §2.2 of [RFC 3987].

> *Example* — This standard uses *terms* to name *datatypes*, as defined in §6 of this standard, and also to name *classes* and *properties*, defined in §5.1 and §5.2. For example, §6.6.3 of this standard defines a *datatype* for representing integers. This *datatype* is identified by a *term* whose *term name* in *prefix* notation is `xsd:integer`. This is short for the following IRI:
>
> ```
> http://www.w3.org/2001/XMLSchema#integer
> ```

> *Note* — IRIs have been chosen in preference to URIs because it is recognised that certain culture-specific genealogical concepts may not have English names, and in such cases the human-legibility of IRIs is advantageous. URIs are a subset of IRIs, and all the *terms* defined in this suite of standard are also URIs.

Term names are compared using the "simple string comparison" algorithm given in §5.3.1 of [RFC 3987]. If a *term name* does not compare equal to an IRI known to the application, the application MUST NOT make any assumptions about the *term*, its meaning or intended use, based on the form of the IRI or any similarity to other IRIs.

> *Note* — This comparison is a simple character-by-character comparison, with no normalisation carried out on the IRIs prior to comparison. It is also how XML namespace names are compared in [XML Names].

> *Example* — The following IRIs are all distinct for the purpose of the "simple string comparison" algorithm given in §5.3.1 of [RFC 3987], , even though an HTTP request to them would fetch the same resource.
>
> ```
> https://éléments.example.com/nationalité
> HTTPS://ÉLÉMENTS.EXAMPLE.COM/nationalit%C3%A9
> https://xn--lments-9uab.example.com/nationalit%c3%a9
> ```

An IRI MUST NOT be used as a *term name* unless it can be converted to a URI using the algorithm specified in §3.1 of [RFC 3987], and back to a IRI again using the algorithm specified in §3.2 of [RFC 3987], to yield the original IRI.

> *Note* — This requirement ensures that *term names* can be used in a context where a URI is required, and that the original IRI can be regenerated, for example for comparison with a list of known IRIs. The vast majority of IRIs, including those in non-Latin scripts, have this property. The effect of this requirement is to prohibit the use of IRIs that are already partly converted to a URI, for example through the use of unnecessary percent or punycode encoding.

> *Example* — Of the three IRIs given in the previous example on how to compare IRIs, only the first may be used as a *term name*. The second and third are prohibited as a result of the unnecessary percent-encoding, and the third is additionally prohibited as a result of unnecessary punycode-encoding.

The *terms* defined in FHISO standards all have *term names* that begin `https://terms.fhiso.org/`. Subject to the requirements in the applicable standards, third parties may also define additional *terms*. It is RECOMMENDED that any such *terms* use either the `http` or preferably the `https` IRI scheme defined in §2.7.1 and §2.7.2 of [RFC 7230] respectively, and an authority component consisting of just a domain name or subdomain under the control of the party defining the *term*.

> *Note* — An `http` or `https` IRI scheme is RECOMMENDED because the IRI is used to fetch a resource during *discovery*, and it is desirable that applications implementing *discovery* should only need to support a minimal number of transport protocols. URN schemes like the `uuid` scheme of [RFC 4122] are NOT RECOMMENDED as they do not have transport protocols that can be used during *discovery*.
>
> The preference for a `https` IRI is because of security considerations during *discovery*. A man-in-the-middle attack during *discovery* could insert malicious content into the response, which, if undetected, could cause an application to process user data incorrectly, potentially discarding parts of it or otherwise compromising its integrity. It is harder to stage a man-in-the-middle attack over TLS, especially if public key pinning is used per [RFC 7469].

4.1 IRI resolution

It is RECOMMENDED that an HTTP `GET` request to a *term name* IRI with an `http` or `https` scheme (once converted to a URI per §4.1 of [RFC 3987]), SHOULD result in a 303 “See Other” redirect to a document containing a human-readable definition of the *term* if the request was made without an `Accept` header or with an `Accept` header matching the format of the human-readable definition. It is further RECOMMENDED that this format should be HTML, and that documentation in alternative formats MAY be made available via HTTP content negotiation when the request includes a suitable `Accept` header, per §5.3.2 of [RFC 7231].

> *Note* — A 303 redirect is considered best practice for [Linked Data], so as to avoid confusing the *term name* IRI with the document containing its definition, which is found at the post-

> redirect URL. The *terms* defined in this suite of standards are not specifically designed for use in Linked Data, but the same considerations apply.

Parties defining *terms* SHOULD arrange for their *term name* to support **discovery**. This when an HTTP `GET` request to a *term name* IRI with an `http` or `https` scheme, made with an appropriate `Accept` header, yields 303 redirect to a machine-readable definition of the *term*.

> *Note* — This standard does not specify a specific version of HTTP, but at the current time, even though HTTP/2 is becoming more popular, HTTP 1.1 is the most widely implemented version of HTTP. While this remains true, applications and discovery servers are encouraged to support HTTP 1.1.

This standard does not define a *discovery* mechanism, but it is RECOMMENDED that parties defining *terms* support FHISO's [Triples Discovery] mechanism, and MAY additionally support other mechanisms. Support for *discovery* by applications is OPTIONAL.

> *Example* — Suppose an application wants to perform *discovery* on the hypothetical `https://example.com/events/Baptism` *term* used in several later examples in this standard. If the application supports FHISO's [Triples Discovery] mechanism, which uses [N-Triples] as its serialisation format, together with some other hypothetical *discovery* mechanism using the `application/x-discovery` MIME type, but prefers to use [Triples Discovery], it might make the following HTTP request:
>
> ```
> GET /events/Baptism HTTP/1.1
> Host: example.com
> Accept: application/n-triples, application/x-discovery; q=0.9
> ```
>
> In this example, the `q=0.9` in the `Accept` header is a quality value which, per §5.3 of [RFC 7231], indicates that the `x-discovery` format is less preferred than `n-triples` which by default has a quality value of 1.0.
>
> If the server supports `n-triples`, it MUST respond with a 303 redirect:
>
> ```
> HTTP/1.1 303 See Other
> Location: https://example.com/events/Baptism.n3
> Vary: Accept
> ```
>
> In this case the redirect is to the original IRI but with `.n3` appended, however the actual choice of IRI is up to the party defining the *term* and running the `example.com` web server. When a server's response is dependent on the contents of an `Accept` header, §7.1.4 of [RFC 7231] says that this SHOULD be recorded in a `Vary` header, as it is in this example.
>
> The application would normally then make a second HTTP request to follow the redirect:
>
> ```
> GET /events/Baptism.n3 HTTP/1.1
> Host: example.com
> Accept: application/n-triples, application/x-discovery; q=0.9
> ```

> This request uses the same `Accept` header as the first, as HTTP redirects contain no information about the MIME type of the destination resource, so at this point the application does not know which *discovery* mechanism the server is using, or whether the server does not support *discovery* or HTTP content negotiation and is serving a human-readable definition.
>
> The server's response to this request should be an N-Triples file containing information about the `Baptism` *term*.

A party defining a *term* MAY support *discovery* without using HTTP content negotiation on their web server by serving a machine-readable definition of the *term* unconditionally (which SHOULD be served via a 303 redirect), however it is RECOMMENDED that such servers implement HTTP content negotiation respecting the `Accept` header.

4.2 Namespaces

> *Editorial note* — The definition of a *namespace* is based on material in FHISO's Vocabularies policy.

The **namespace** of a *term* is another *term* which identifies a collection of related *terms* defined by the same party. The *term name* of the *namespace* is also referred to as its **namespace name**. The *namespace name* of the *namespace* of some *term* is found as follows.

If the *term name* ends with a non-empty fragment identifier, then its *namespace name* is formed by removing the fragment identifier, leaving an IRI ending with a #.

> *Example* — [Basic Concepts] uses a *datatype* identified by the following *term name* IRI:
>
> `http://www.w3.org/2001/XMLSchema#integer`
>
> This concludes with a fragment identifier, "`integer`", and therefore its *namespace name* is its *term name* with the fragment identifier removed:
>
> `http://www.w3.org/2001/XMLSchema#`

Otherwise, if the *term name* ends with a non-empty path segment, then its *namespace name* is formed by removing the path segment, leaving an IRI ending with a `/`.

> *Example* — [Basic Concepts] defines a *property* identified by the following *term name* IRI:
>
> `https://terms.fhiso.org/types/pattern`
>
> This concludes with a path segment, "`pattern`", and therefore its *namespace name* is its *term name* with the path segment removed:
>
> `https://terms.fhiso.org/types/`

Otherwise, the *namespace* is undefined.

> *Note* — This means the *namespace* of a *namespace* is necessarily undefined, as *namespace names* always end with a `#` or `/`, meaning they end with either an empty fragment identifier or an empty path segment.

4.3 Prefix notation

Term names are sometimes referred using **prefix notation**. This is a system whereby **prefixes** are assigned to *namespace names* which occur frequently in *term names*. Then, instead of writing the *term name* in full, the leading portion of the *term name* equal to the *namespace name* is replaced by its *prefix* followed by a colon (U+003A) separator.

> *Example* — The *term name* `http://www.w3.org/2000/01/rdf-schema#Class` is used in several places in this standard. Instead of writing this in full, if the `rdfs` *prefix* is bound to its *namespace name* `http://www.w3.org/2000/01/rdf-schema#`, then this IRI can be written in *prefix form* as `rdfs:Class`.

5 Underlying type system

> *Editorial note* — The material in this section is new in this draft, but draws heavily on FHISO's Vocabularies policy.

> *Note* — This section defines a basic type system for *terms* and a simple vocabulary for describing them. This formalism provides a solid theoretical framework for defining extensions to FHISO standards, and is used by applications during *discovery* (support for which is OPTIONAL). Parties who are simply implementing a higher level FHISO standards will typically not need to be familiar with this material.

5.1 Classes

Terms are used in many contexts in FHISO standards and it can be useful to have a concise, machine-readable way of stating the use for which it was defined.

A **class** is a *term* used to denote a particular context or use for which other *terms* may be defined. Standards defining such contexts SHOULD define a *class* to represent that context, and MUST do so if the third parties are permitted to define their own *terms* for use in that context.

> *Example* — A hypothetical standard might defined various *terms* representing events of genealogical interest that might occur during a person's lifetime. Examples might include:
>
> ```
> https://example.com/events/Baptism
> https://example.com/events/Ordination
> ```

```
https://example.com/events/Emigration
https://example.com/events/Death
```

The standard SHOULD provide a *class* to represent the abstract concept of an event type, and as the *class* is itself a *term*, it must have an IRI as its *term name*. Perhaps it might be:

```
https://example.com/events/EventType
```

This *class* might be referred to as the *class* of event types.

> *Note* — The words "class" and "type" are used in many contexts in computing. As used here, a *class* is similar to a datatype of which *terms* are values, or a class of which *terms* are instances, or a named enumeration type of which *terms* are values. FHISO's use of this word does not mean that the other notions associated with the word "class" in object-oriented programming apply here.

The *term name* of a *class* is also referred to as its **class name**.

5.1.1 The type of a term

When a *term* has been defined for use in the context denoted by some *class*, that *class* is referred to as the **type** of the *term*.

> *Example* — In *prefix notation*, with the *prefix* `ex` bound to `https://example.com/events/`, the *type* of `ex:Baptism` from the previous example is `ex:EventType`.

The *type* of a *term* is a piece of information which MUST be provided, perhaps implicitly, when defining a *term*. As such, the *type* is a *property* of the *term*, as defined in §5.2, and needs a *property term* to represent it. This standard uses the `rdf:type` *term* for this purpose:

Property definition

Name	`http://www.w3.org/1999/02/22-rdf-syntax-ns#type`
Type	`http://www.w3.org/1999/02/22-rdf-syntax-ns#Property`
Range	`http://www.w3.org/2000/01/rdf-schema#Class`

> *Note* — The table above sets out the formal *properties* of the `rdf:type` *property*. The first line of this definition states the *term name* of the `rdf:type` *property*. As required above, the *type* of a *term* MUST be specified when a *term* is defined and the `rdf:type` *property* is no exception. Its type is `rdf:Property` which is defined in §5.2 of this standard. The meaning of the *range* is given in §5.2.1.

> *Note* — The `rdf:type` *property term* is defined §3.3 of [RDF Schema], however implementers may safely use this *property term* for the purposes of this standard without

> reading [RDF Schema]. The decision to use this RDF *term* in FHISO's standards rather than invent a new *term* allows for greater compatibility with existing third-party vocabularies.

5.1.2 The class of classes

As a *class* is a *term*, defining a *class* is itself a context in which *terms* are defined, including by third parties. This means the general concept of a *class* needs a *term* defining to represent it. This standard uses the `rdfs:Class` *term* for this purpose:

Class definition

Name	`http://www.w3.org/2000/01/rdf-schema#Class`
Type	`http://www.w3.org/2000/01/rdf-schema#Class`
Superclass	`http://www.w3.org/2000/01/rdf-schema#Resource`
Required properties	`http://www.w3.org/1999/02/22-rdf-syntax-ns#type`

> *Note* — This can be thought of as a *class* of *classes*. It is not merely an arcane abstraction: it serves a useful role in *discovery*. If *discovery* is carried out on the *term name* of a *class*, it is useful to be able to indicate that the *term* is a *class*. This can be done by saying the *type* of the *term* is `rdfs:Class`.

> *Note* — Although the `rdfs:Class` *class* is defined in §2.2 of [RDF Schema], this standard does not require support for any of the facilities in [RDF Schema], nor are parties defining *classes* or *terms* required to do so in a manner compatible with RDF. An implementer may safely use the `rdfs:Class` *class* for the purposes of this standard using just the information given in this section without reading [RDF Schema] or otherwise being familiar with RDF.
>
> The decision to use `rdfs:Class` and other *terms* from [RDF Schema] is due to FHISO's practice of reusing facilities from existing standards when they are a good match for our requirements, rather than inventing our own versions with similar functionality. It also allows future standards and vendor extensions the option of reusing existing third-party vocabularies where appropriate, as most such vocabularies are also aligned with RDF.

The *type* of any *class* is therefore `rdfs:Class`.

> *Note* — There is no need for a further level of abstraction to represent the *type* of `rdfs:Class`. As `rdfs:Class` is just another *class*, albeit a fairly special one, the *type* of `rdfs:Class` is `rdfs:Class`.

5.1.3 Subclasses

A *class* MAY be defined as a **subclass** of another *class*. The latter *class* is referred to as the **superclass** of the former *class*. The *subclass* denotes a more specialised version of the context denoted by its *superclass*. A *term* whose *type* is *subclass* of some other *class* MAY be used wherever a *term* is required whose *type* is the *superclass*.

> *Example* — In the example above, a hypothetical standard was said to have defined a *class* representing event types. The same hypothetical standard might define a *subclass* of this called `IndividualEventType` to represent individual events for those events that are principally about a single person. In such a scheme, a baptism would be considered an individual event, while a marriage would probably not as it involves two principal participants. In a context where a *term* of *type* `EventType` is required, an `IndividualEventType` like `Baptism` MAY be used; but in a context where an `IndividualEventType` is required, others sorts of event such as `Marriage` MUST NOT be used.

The *superclass* of a *class* MUST be specified when defining a *class* to be the *subclass* of some other *class*. As such, the *superclass* of the *class* is a *property* of the *class*, as defined in §5.2 and needs a *property term* to represent it. This standard uses the `rdfs:subClassOf` *term* for this purpose:

Property definition

Name	`http://www.w3.org/2000/01/rdf-schema#subClassOf`
Type	`http://www.w3.org/1999/02/22-rdf-syntax-ns#Property`
Range	`http://www.w3.org/2000/01/rdf-schema#Class`

> *Note* — The `rdfs:subClassOf` *property term* is defined §3.4 of [RDF Schema], however implementers may safely use this *property term* for the purposes of this standard without reading [RDF Schema]. The decision to use this RDF *term* in FHISO's standards rather than invent a new *term* allows for greater compatibility with existing third-party vocabularies.

The notion of a *subclass* is transitive, meaning that if a *class* is a *subclass* of a second *class*, and that second *class* is a *subclass* of a third *class*, then the first *class* is a *subclass* of the third. The notion of a *subclass* is also reflexive, meaning that a *class* is by definition a *subclass* of itself. The notion of a *superclass* is similarly transitive and reflexive.

The `rdfs:subClassOf` *property* is defined as a *required property* of `rdfs:Class`, meaning its *supertypes* MUST be specified whenever a new *class* is defined. However this standard does not require every *superclass* to be identified explicitly. If a *class* has two or more *superclasses*, and one of the *superclasses* is itself a *superclass* of another of the *superclasses*, then the *superclass* of the *superclass* need not be identified explicitly.

> *Example* — Continuing the previous example, it is correct to say that the hypothetical `IndividualEventType` *class* is a *superclass* of `EventType`, but it is equally correct to say that it is a *superclass* of the `rdfs:Resource` *universal superclass* defined in §5.1.4, below. The

> `IndividualEventType` *class* therefore has two *superclasses*, one of which (`EventType`) is a *superclass* of the other (`rdfs:Resource`). Because of this, it is not necessary to state that `IndividualEventType` is a *subclass* of `rdfs:Resource`.

5.1.4 The universal superclass

This standard uses `rdfs:Resource` as the **universal superclass** defined to be the *superclass* of all *classes*.

Class definition

Name	`http://www.w3.org/2000/01/rdf-schema#Resource`
Type	`http://www.w3.org/2000/01/rdf-schema#Class`
Required properties	`http://www.w3.org/1999/02/22-rdf-syntax-ns#type`

> *Note* — The `rdfs:Resource` *class* is defined in §2.1 of [RDF Schema].

This *class* has no semantics of its own, other than to be *class* of all things that can be expressed in this data model.

> *Note* — The `rdfs:Resource` *class* is useful with the `rdfs:subClassOf` *property* when defining a *class* which has no other *superclass*.

5.2 Properties

During *discovery*, and in other situations when a formal definition of a particular *term* is needed, it is necessary to have a formalism for providing information about that *term*.

A **property** is a particular piece of information that might be provided when defining some entity. The thing being defined is typically a *term*, and is called the **subject** of the *property*.

> *Editorial note* — The *subject* of the *property* is only said to be typically a *term* so that *citation elements terms* (in [CEV Concepts]) can be made a subclass of *property terms*. The *subject* of a *citation element* is a *source* which is not a *term* as we don't require them to be identified by an IRI. It is likely that other genealogical concepts, possibly including individual attributes in ELF, may also be treated as *properties* whose *subjects* are not *terms*. In the case of individual attributes, the subject is an individual which is likely not identified by an IRI.

The *property* consists of two parts, both of which are REQUIRED to be present:

— a **property name**, which identifies the nature of the information in the *property*; and
— a **property value**, which contains the data about the *subject* of the *property*.

The *property name* SHALL be a *term* that has been defined to be used as a *property name* in the manner required by this standard; a *term* defined for this purpose is called a **property term**.

> *Note* — This nomenclature draws a distinction between a *property name* and a *property term*. The former is part of a *property*, and is therefore part of the description of the *subject* of the *property*, while the latter is an item of vocabulary reference by that description. The *property name* is a *property term*.

The *property value* SHALL be a *term*, a *string*, or a *language-tagged string*. The *property value* MAY additionally be tagged with a *datatype name*, which is a *term name* defined in §6.

> *Editorial note* — The ability to tag *property values* with a *datatype* is not currently used in this standard, but is required so that *citation elements*, as defined in [CEV Concepts], can be a subclass of *properties*. More work is needed to fully harmonise these concepts, and it may become necessary to pull the notion of a *localisation set* down into Basic Concepts.

Properties SHALL NOT have default *property values* that applies when the *property* is absent, however standards MAY define how an *conformant* application handles the absence of a *property*.

Standards which introduce such pieces of information SHOULD define a *property terms* to represent them, and MUST do so if third parties are permitted to define their own *terms* and if it is RECOMMENDED or REQUIRED that these third parties document or otherwise make available the information represented by the *property*.

> *Example* — An earlier example introduced several hypothetical *terms* for events of genealogical interest, such as birth, baptism, ordination, emigration and death. Many events can occur multiple times during a person's life: for example, a person might emigrate more than once. But other events cannot by definition occur more than once: birth and death are obvious examples. The number of times something is permitted to occur is sometimes called its cardinality, and if the authors of this hypothetical standard considered it a relevant concept, they SHOULD define a *property term* to represent the concept of cardinality:
>
> ```
> https://example.com/events/cardinality
> ```
>
> If the hypothetical standard allows third parties to define additional types of event, and either recommends or requires that they state the cardinality of the new events, then the standard MUST define a *property term* representing cardinality.

The *term name* of a *property term* is also referred to as its *property term name*.

The *class* of *property terms* has the following *class name*:

Class definition

Name	`http://www.w3.org/1999/02/22-rdf-syntax-ns#Property`
Type	`http://www.w3.org/2000/01/rdf-schema#Class`
Superclass	`http://www.w3.org/2000/01/rdf-schema#Resource`
Required properties	`http://www.w3.org/1999/02/22-rdf-syntax-ns#type`
	`http://www.w3.org/2000/01/rdf-schema#range`

> *Note* — The `rdf:Property` *term* is defined in §2.8 of [RDF Schema]. As with the `rdfs:Class` *term*, an implementer may safely use the `rdf:Property` *terms* for the purposes of this standard without reading [RDF Schema].

> *Editorial note* — The notion of *cardinality* may also be moved here from [CEV Concepts].

5.2.1 Range

The **range** of a *property term* is a formal specification of allowable *property values* for a *property* whose *property name* is that *property term*. The *range* SHALL be a *class name* or a *datatype name*.

> *Note* — *Datatypes* provide a formal description of the values allowed in a particular context. They are defined in §6 of this standard.

When the *range* is a *class*, the *property value* SHALL be a *term* whose *type* is that *class*; when the *range* is a *datatype*, the value associated with the property shall be a *string* in the *lexical space* of that *datatype*.

> *Example* — An earlier example gave a hypothetical `cardinality` *property term* that might be used when defining genealogical events. Most likely, the *property value* of this *property* would be a representation of "one" or "unbounded", depending on whether the event is one that can occur just once, or whether it can occur multiple times. The party defining this *property* would need to consider how best to represent these two values.
>
> One option is to define two *terms* to represent these options, say:
>
> ```
> https://example.com/events/SinglyOccurring
> https://example.com/events/MultiplyOccurring
> ```
>
> The context in which these two *terms* can be used is when specifying a cardinality, so a `Cardinality` *class* would be defined:
>
> ```
> https://example.com/events/Cardinality
> ```
>
> The *type* of `SinglyOccuring` and `MultiplyOccuring` would be `Cardinality`, and the *range* of the `cardinality` *property* would be the `Cardinality` *class*. Having a *property* and the *class* that serves as its *range* only differing in capitalisation is a common idiom.
>
> A second option is to use two *strings* to represent the possible cardinalities, perhaps "1" and "unbounded". A *datatype* would then be defined whose *lexical space* consisted of just these two *strings*, and the *datatype* given a name like:
>
> ```
> https://example.com/events/Cardinality
> ```
>
> As in the first option, the *range* of the `cardinality` *property* would be the `Cardinality` *class*.

> A third and likely preferable option would be to name the `cardinality` *property* differently, say `canOccurMultiply`, so that its *range* could be a standard boolean *datatype* like `xsd:boolean`.

> *Note* — This standard has already defined one *property term*, namely the `rdf:type` *property term* in §5.1.1. The *type* of a *term* is the *class* which denotes the context in which it can be used. Therefore the *range* of `rdf:type` is `rdfs:Class`, as shown in the *property* definition table in §5.1.1.

Standards which define *property terms* SHOULD specify their *range*, and MUST do so if third parties are permitted to define their own *terms* and if it is RECOMMENDED or REQUIRED that these third parties document or otherwise make available the information represented by the *property term*.

> *Note* — This is the same wording that is used in §5.2 to specify when a *property term* MUST be defined. In circumstances where a *property term* MUST be defined, its *range* MUST also be defined.

The *range* of a *property term* is itself a *property* which is defined as follows:

Property definition

Name	`http://www.w3.org/2000/01/rdf-schema#range`
Type	`http://www.w3.org/1999/02/22-rdf-syntax-ns#Property`
Range	`http://www.w3.org/2000/01/rdf-schema#Class`

> *Note* — The *range* of the `rdfs:range` *property* is defined above to be `rdfs:Class`, although the *property value* of an `rdfs:range` *property* can be either a *class name* or a *datatype name*. This works because `rdfs:Datatype` is defined as a *subclass* of `rdfs:Class`, and therefore a *datatype name* can be used where a *class name* is required.

> *Editorial note* — We may need to introduce the concepts of the **domain** of a *property term*, currently in our Vocabularies policy. Careful consideration will be needed before the *domain* is introduced to ensure it does not cause forwards compatibility problems if new uses are found for the *property*.

5.2.2 Required properties

A *property* which MUST be provided when a third party defining a new *term* with some particular *type* is called a **required property**.

> *Example* — The notion of a *datatype* is defined in §6 of this standard, and is common to many FHISO standards. *Datatypes* are identified by a *term* known as their *datatype name*, and any party defining a *datatype* for use with FHISO standards is REQUIRED to specify its *pat-*

> *tern*, *supertype* if any, and whether it is an *abstract datatype*. These pieces of information are specified via three *properties* called `types:pattern`, `types:nonTrivialSupertype` and `types:isAbstract`. These three *properties* are therefore the *required properties* for *datatypes*. In fact, *datatypes* have a fourth *required property* which is their *type*: i.e. a statement that the *term* is a *datatype*.

The *type* of a new *term* being defined is a *class*, and therefore the list of the *property names* of the *required properties* of a *term* defined with that *type* is a *property* of that *class*. The *property* representing the *required properties* of a *class* is defined as follows:

Property definition

Name	`https://terms.fhiso.org/types/requiredProperty`
Type	`http://www.w3.org/1999/02/22-rdf-syntax-ns#Property`
Range	`http://www.w3.org/1999/02/22-rdf-syntax-ns#Property`

> *Editorial note* — This data model does not provide a convenient mechanism for the *property value* to be a list. Therefore, instead of one `requiredProperties` *property* whose value is a list of *property names*, *classes* will normally have multiple `requiredProperty` *properties* each of whose value is a single *property name*.

The *required properties* of a *class* SHALL include all the *required properties* of each *superclass* of the *class*.

> *Note* — The `rdf:type` is a *required property* of `rdfs:Resource` and all *classes* are a *subclass* of `rdfs:Resource`, thus `rdf:type` is a *required property* of every *class*.

6 Datatypes

> *Editorial note* — The concepts related to *datatypes* were originally defined in the CEV Concepts draft. This section and its subsections have been moved here to be more generally usable.

A **datatype** is a *term* which serves as a formal description of the values that are permissible in a particular context. Being a *term*, a *datatype* is identified by a *term name* which is an IRI. The *term name* of a *datatype* is also referred to as its **datatype name**.

A *datatype* has a **lexical space** which is the set of *strings* which are interpreted as valid values of the *datatype*. The definition of a *datatype* SHALL state how each string in its *lexical space* maps to a logical value, and state the semantics associated with of those values.

> *Note* — This definition of a *datatype* is sufficiently aligned with XML Schema's notion of a simple type, as defined in [XSD Pt2], that XML Schema's simple types can be used as *datatypes* in this standard. Best practice on how to get an IRI for use as the *term name* of

> XML Schema types can be found in [SWBP XSD DT]. Similarly, this standard's definition of a *datatype* is very similar to the definition of a datatype in [RDF Concepts], and RDF datatypes can be used as *datatypes* in this standard.

> *Example* — XML Schema defines an integer type in §3.4.13 of [XSD Pt2] which is well-suited for use in this standard. FHISO uses this type where integer values occur. It discussed in §6.6.3 of this standard.

The mapping from lexical representations to logical values need not be one-to-one. If a *datatype* has multiple lexical representations of the same logical value, a *conformant* application MUST treat these representations equivalently and MAY change a *string* of that *datatype* to be a different but equivalent lexical representation.

> *Note* — This allows applications to store such *strings* internally using as an entity (such as a database field or a variable) of some appropriate type without retaining the original lexical representation.

> *Example* — The XML Schema `integer` *datatype* used in the previous example is one where the mapping from lexical representation to value is many-to-one rather than one-to-one. This is due to *lexical space* including strings with a leading + sign as well as superfluous leading 0s, and means that "`00137`", "`+137`" and "`137`" all represent the same underlying value: the number one hundred and thirty-seven. Because *conformant* applications MAY convert strings between equivalent lexical representations, they MAY store them in a database in an integer field and regenerate *strings* in a canonical representation.

Strings outside the *lexical space* of a *datatype* MUST NOT be used where a *string* of that *datatype* is required. If an application encounters any such *strings*, it MAY remove them from the dataset or MAY convert them to a valid value in an implementation-defined manner. Any such conversion that is applied automatically by an application MUST either be locale-neutral or respect any locale given in the dataset.

> *Example* — XML Schema defines a `date` type in §3.3.9 of [XSD Pt2] which has a *lexical space* based on [ISO 8601] dates. If, in a dataset that is somehow identified as being written in German, an application encountering the *string* "`8 Okt 2000`" in a context where an XML Schema `date` is expected, it MAY convert this to "`2000-10-08`". However an application encountering the *string* "`8/10/2000`" MUST NOT conclude this represents 8 October or 10 August unless the document includes a locale that uniquely determines the date format. In this case, information that the document is in English is not sufficient as different English-speaking countries have different conventions for formatting dates.

This standard uses the `rdfs:Datatype` *class* as the *class* of *datatypes*, defined as follows:

Class definition

Name	`http://www.w3.org/2000/01/rdf-schema#Datatype`
Type	`http://www.w3.org/2000/01/rdf-schema#Class`
Superclass	`http://www.w3.org/2000/01/rdf-schema#Class`

Required properties	`http://www.w3.org/1999/02/22-rdf-syntax-ns#type`
	`https://terms.fhiso.org/types/pattern`
	`https://terms.fhiso.org/types/nonTrivialSupertypeCount`
	`https://terms.fhiso.org/types/isAbstract`

> *Note* — The `rdfs:Datatype` *term* is defined in §2.4 of [RDF Schema].

> *Note* — The *class* of *datatypes*, `rdfs:Datatype`, is defined here to be a *subclass* of the *class* of all *classes*, `rdfs:Class`. This may appear counter-intuitive as new *classes* are normally defined to be a *subclass* only of `rdfs:Resource`, the *universal superclass*. The reason for doing this is partly for compatibility with its definition in [RDF Schema], but the reasons [RDF Schema] took this unusual decision are also valid here.
>
> Making `rdfs:Datatype` a *subclass* of `rdfs:Class` says that a *datatype name* MAY be used where a *class name* is expected. In many situations this is desirable. For example, the *range* of a *property* is, in general, a *class name*, but frequently a *datatype name* will be used: for example, the *range* of `types:isAbstract` is the `xsd:boolean` *datatype*. By making `rdfs:Datatype` a *subclass* of `rdfs:Class`, the *range* of `rdfs:range` can be `rdfs:Class`.

6.1 Patterns

A party defining a *datatype* SHALL specify a **pattern** for that *datatype*. This is a regular expression which provides a constraint on the *lexical space* of the *datatype*. Matching the *pattern* might not be sufficient to validate a *string* as being in the *lexical space* of the *datatype*, but parties defining a *datatype* MUST ensure that all *strings* in the *lexical space* match the *pattern*, even if some *strings* outside the *lexical space* also match the *pattern*.

> *Note* — Patterns are included in this standard to provide a way for an application to find out about the *lexical space* of a unfamiliar *datatype* through *discovery*.

> *Example* — The XML Schema `date` type mentioned in a previous example has the following *pattern* (here split onto two lines for readability — the second line is an optional timezone which the XML Schema `date` type allows).
>
> ```
> -?([1-9][0-9]{3,}|0[0-9]{3})-(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])
> (Z|(\+|-)((0[0-9]|1[0-3]):[0-5][0-9]|14:00))?
> ```
>
> This *pattern* matches *strings* like “`1999-02-31`”. Despite matching the *pattern*, this *string* is not part of the *lexical space* of this `date` type as 31 February is not a valid date.

The *property term* representing the *pattern* of a *datatype* is defined as follows:

Property definition

Name	`https://terms.fhiso.org/types/pattern`
Type	`http://www.w3.org/1999/02/22-rdf-syntax-ns#Property`
Range	`https://terms.fhiso.org/types/Pattern`

> *Note* — The `types:Pattern` *datatype* used as the *range* of this property is defined in a separate [FHISO Patterns] standard which defines the dialect of regular expressions which FHISO supports.

> *Editorial note* — We added [FHISO Patterns] after adding most of the *pattern* examples in this and other current draft standards, and have not yet reviewed them to ensure they all match that regular expression syntax.

> *Editorial note* — This standard does not use `xsd:pattern` as the *property term*, even though it is used as a predicate in OWL 2. Its use would pose a difficulty because none of the relevant W3C specifications indicate what the `rdfs:domain` of `xsd:pattern` is supposed to be. Possibly it is an `owl:Restriction`, which would be incompatible with this use. Using `xsd:pattern` would also require us to use precisely the form of regular expression defined in Appendix G of [XSD Pt2].

A *datatype* with a *pattern* other than "`.*`" is known as a **structured datatype**, while one with a *pattern* of "`.*`" is known as an **unstructured datatype**.

6.2 Subtypes

A *datatype* MAY be defined as a **subtype** of one or more other *datatype* which are referred to as its **supertypes**. This is used to provide a more specific version of a more general *datatype*. If an application is unfamiliar with the *subtype* it MAY process it as if it were one of its *supertypes*. The *subtype* MUST be defined in such a way that at most this results in some loss of meaning but does not introduce any false implications about the dataset.

> *Editorial note* — Would it be a useful simplification if this definition said something along the following lines? If a *datatype* has more than one *supertype* which are not *abstract datatypes*, one of these *supertypes* SHALL be the *subtype* of all of the others. This is similar to Java's rule on inheritance: you can multiply inherit interfaces (here *abstract datatypes*) but only singly inherit a class (here *datatypes* other than *abstract datatypes*).

The *lexical space* of the *subtype* SHALL be a subset of the *lexical space* of the *supertype*.

> *Note* — It is the *lexical space* of the *subtype* that is required to be a subset of the *lexical space* of the *supertype*. The set of *strings* that match the *pattern* of the *subtype* might not necessarily be a subset of that of the *supertype*. This is because the *pattern* is permitted to match *strings* outside the *lexical space*, as in the example of the date "`1999-02-31`".

> *Editorial note* — This section needs an example, but not one involving *language-tagged datatypes* as they have yet to be defined. Currently the only uses of *subtypes* as with *language-tagged datatypes*, or involve the rather arcane ultimate *supertypes*, `xsd:anyAtomicType`. It is anticipated that dates will provide a good example, as we expect to need several *subtypes* of `AbstractDate`, but FHISO has yet to specify how dates are handled in this data model.

> *Note* — The concept of a *subtype* in this standard corresponds to XML Schema's concept of derivation of a simple type by restriction per §3.16 of [XSD Pt1]. XML Schema does not have concept compatible with this standard's notion of an *abstract datatype*, as in XML Schema only complex types can be abstract and complex types are not *strings*. If it is desirable to describe a FHISO *abstract datatype* in XML Schema, it should be defined as a normal simple type, with the information that it is abstract conveyed by another means.

All *datatypes* are implicitly a *subtype* of the `xsd:anyAtomicType` *abstract datatype* defined to be the *universal supertype* in §6.6.6.

> *Editorial note* — The following paragraph is duplicated in §5.1.3.

The notion of a *subtype* is transitive, meaning that if a *datatype* is a *subtype* of a second *datatype*, and that second *datatype* is a *subtype* of a third *datatype*, then the first *datatype* is a *subtype* of the third. The notion of a *subtype* is also reflexive, meaning that a *datatype* is by definition a *subtype* of itself. The notion of a *supertype* is similarly transitive and reflexive.

6.2.1 Non-trivial supertypes {non-trivial-types}

The **trivial supertypes** of a *datatype* are certain *supertypes* whose status as a *supertype* of the *datatype* is implied by the data model defined in this standard. The *trivial supertypes* of a *datatype* include the *datatype* itself and the *universal supertype*, `xsd:anyAtomicType`. A *supertype* which not a *trivial supertype* is called a **non-trivial supertype**.

> *Note* — *Unions of datatypes*, as defined in §6.5, are also *trivial supertypes*.

The *property term* representing a *non-trivial supertype* of a *datatype* is defined as follows:

Property definition

Name	`https://terms.fhiso.org/types/nonTrivialSupertype`
Type	`http://www.w3.org/1999/02/22-rdf-syntax-ns#Property`
Range	`http://www.w3.org/2000/01/rdf-schema#Datatype`

> *Editorial note —* An earlier unpublished draft of this standard reused the `rdfs:subClassOf` *property* to represent the *supertype* of a *datatype*. This introduced a fairly obscure incompatibility with RDF. RDF only requires that the value space of a *subtype* is a subset of the value space of the *supertype*: it says nothing about their *lexical spaces*. Thus in RDF it would be possible for `xsd:boolean` to be a *subclass* of `xsd:integer` if the boolean values "true" and "false" are considered to be identical to the integer values 1 and 0, respectively (though in fact they're not). This is despite the *strings* "`true`" and "`false`" being part of *lexical space* of `xsd:boolean` but not of `xsd:integer`. This means a stronger relationship is needed which constrains both the *lexical space* and the *value space*. This is what `types:nonTrivialSupertype` provides. This standard explicitly does not state whether `types:nonTrivialSupertype` is an `rdfs:subPropertyOf` `rdfs:subClassOf`.

The `types:nonTrivialSupertype` *property* MUST NOT be used to record a *trivial supertypes* of the *datatype*.

A `types:nonTrivialSupertype` *property* MUST be used to record every *non-trivial supertype* of a *datatype* which is not implied by the transitivity of `types:nonTrivialSupertype` and the other `types:nonTrivialSupertype` *properties* present.

> *Example —* Suppose a hypothetical standard defines three *datatypes*, `DateTime`, `Date`, and `Year`. If the standard specifies that `Year` has a `types:nonTrivialSupertype` *property* with *property value* `Date`, and that `Date` has a `types:nonTrivialSupertype` *property* with *property value* `DateTime`, it is not necessary for the standard to record that `Year` has a second `types:nonTrivialSupertype` *property* with *property value* `DateTime` as this is implied by the other two. Nevertheless, the standard MAY do so.

As a way of checking for data integrity during *discovery*, an additional *property* is provided representing the number of *non-trivial supertypes* of the *datatype* that are either recorded using `types:nonTrivialSupertype` *properties* or are implied by them via transitivity:

Property definition

Name	`https://terms.fhiso.org/types/nonTrivialSupertypeCount`
Type	`http://www.w3.org/1999/02/22-rdf-syntax-ns#Property`
Range	`http://www.w3.org/2001/XMLSchema#integer`

> *Editorial note* — Should this have a *range* of `xsd:nonNegativeInteger` instead?

This `types:nonTrivialSupertypeCount` *property* is a *required property* of `rdfs:Datatype`, and must be specified (with a value of "0") even if there are no *non-trivial supertypes*.

An application which finds out about a *datatype* through *discovery* MUST NOT assume it knows the *supertypes* of the *datatype* unless it has verified that the number of *non-trivial supertypes* specified with the `types:nonTrivialSupertype` *property* or implied by the transitivity of that *property* is equal to the value of the `types:nonTrivialSupertypeCount` *property*.

> *Editorial note* — These two *properties* are likely to be changed in a future draft. A cleaner implementation would be to have a single `types:supertypes` *property* which is a list of the *non-trivial supertypes* of the *datatype*, however at the moment the data model does not support list-valued properties. This is a recognised deficiency in the current data model that is likely to be changed, but which requires considerable work.
>
> The reason why a single list-valued *property* is inherently safe whereas a collection of a *properties* is not is that the list-valued property can be made a *required property* which MUST be present exactly once. If it is not, an application knows that is missing and will not assume it properly understands the *datatype*. However if one of several `types:nonTrivialSupertype` *properties* goes missing, this might go unnoticed. This is particular relevant if the *properties* have been processed by RDF applications, as the RDF philosophy is that RDF triples can be taken in isolation and that removing one or more RDF triples merely loses information rather than altering the meaning of something. It is therefore quite conceivable that an RDF triple encoding a *property* might go missing.
>
> In [CEV Concepts], a missing `types:nonTrivialSupertype` might result in a *datatype* being incorrectly thought not to conform to the *range* of some *citation element*, which might result in a valid *citation element* being discarded. The importance of avoiding this is the reason why the current draft includes a `types:nonTrivialSupertypeCount` as a check.

In the *datatype* definition tables in this standard, a single *supertype* row is given which is understood to contain a complete list of all *non-trivial supertypes* and no *trivial supertypes*.

> *Editorial note* — A future version of this standard needs to address what changes may be made to an existing *datatype* hierarchy. Specifically, can a new *non-trivial supertype* be injected into an existing hierarchy? Doing so changes the number of *non-trivial supertypes* a *datatype* has, so at present it would break third-party *subtypes*. A related question is whether a third party can inject their own *non-trivial supertype* into your *datatype* hierarchy. Probably they should not be allowed to, and most use cases where this might be needed can hopefully be accommodated with a *union of datatypes*.

6.3 Abstract datatypes

A *datatype* MAY be defined to be a **abstract datatype**. An *abstract datatype* is one that MUST only be used as a *supertype* of other types. A *string* MUST NOT be declared to have a *datatype* which is an *abstract datatype*. *Abstract datatypes* SHALL specify a *pattern* and SHALL have a *lexical space*.

> *Note* — The *lexical space* of an *abstract datatype* and any *pattern* defined on it serve to restrict the *lexical space* of all its *subtypes*. If no such restriction is desired, the *lexical space* may be defined as the space of all *strings*.

The *property* that represents whether or not a *datatype* is an *abstract datatype* defined as follows:

Property definition

Name	`https://terms.fhiso.org/types/isAbstract`
Type	`http://www.w3.org/1999/02/22-rdf-syntax-ns#Property`
Range	`http://www.w3.org/2001/XMLSchema#boolean`

> *Editorial note* — Are *abstract datatypes* a necessary part of our data model at all? They were introduced to allow an `AbstractDate` *datatype*, but is it necessary for this datatype to be an *abstract datatype*?

6.4 Language-tagged datatypes

A **language-tagged datatype** is a *datatype* whose values are *language-tagged strings* consisting of both a *string* from the *lexical space* of the *datatype* and a *language tag* to identify the language in which that particular *string* is written.

Language-tagged datatypes SHOULD be used whenever a *datatype* is needed to represent textual data that is in a particular language or script and which cannot automatically be translated or transliterated as required, and SHOULD NOT be used otherwise.

> *Example* — In a context where a year Anno Domini is required, a *language-tagged datatype* SHOULD NOT be used, and the *lexical space* of the *datatype* should encompass *strings* like, say, “2015”. Even though an application designed for Arabic researchers might need to render this year as “٢٠١٥” using Eastern Arabic numerals, this conversion can be done entirely in the application’s user interface, so a *language-tagged datatype* is not required and SHOULD NOT be used.

> *Example* — The [CEV Vocabulary] defines a *datatype* for representing the names of authors and other people, which has the following *term name*:
>
> `https://terms.fhiso.org/sources/AgentName`

> A person's name is rarely translated in usual sense, but may be transliterated. For example, the name of Andalusian historian صاعد الأندلسي might be transliterated "Ṣā'id al-Andalusī" in the Latin script. Because machine transliteration is far from perfect, a *language-tagged datatype* SHOULD be used to allow an application to store both names.
>
> An author's names may also be respelled to conform to the spelling and grammar rules of the reader's language. An Englishman named Richard may be rendered "Rikardo" in Esperanto: the change of the "c" to a "k" being to conform to Esperanto orthography, while the final "o" marks it as a noun. The respelling would be tagged `eo`, the language code for Esperanto.

Language-tagged datatypes SHALL define a *pattern*, just as other *datatypes* do.

> *Note* — Because the *language tag* is not part of the *lexical space* of the *datatype*, and is not embedded in the *string*, a *pattern* cannot be used to constrain the *language tag*.

A *datatype* that is not a *language-tagged datatype* is called a **non-language-tagged datatype**.

> *Note* — This means the classification of *datatypes* as *language-tagged* or *non-language-tagged* is orthogonal to their classification as *structured* or *unstructured*. It is anticipated that most *non-language-tagged datatypes* will be *structured datatype*.

> *Example* — The `AgentName` datatype from the previous example is a microformat which is constrained by a *pattern* meaning it is a *structured datatype*, but it is also a *language-tagged datatype* as names can be translated and transliterated.

A *language-tagged datatypes* MAY be used as a *supertype* of a *datatype*. All *subtypes* of a *language-tagged datatype* SHALL also be *language-tagged datatypes*.

> *Editorial note* — An earlier unpublished draft of this standard also said that the *subtypes* of a *non-language-tagged datatypes* (other than `xsd:anyAtomicType`) were REQUIRED to be *non-language-tagged*, with an exception for *subtypes*. This requirement has been dropped to allow *unions* to be defined which contain a mixture of *language-tagged datatypes* and *non-language-tagged datatypes*.

All *language-tagged datatypes* are implicitly a *subtype* of the `rdf:langString` *datatype* defined in §6.6.5.

> *Note* — There is no need for a *property* stating whether or not a *datatype* is a *language-tagged datatype* because this information is conveyed using the `types:nonTrivialSupertype` *property*.

6.5 Unions of datatypes

A **union of datatypes** is an *abstract datatype* which is defined in terms of a unordered list of two or more distinct *datatypes* called its **constituent datatypes**. The *constituent datatypes* MUST NOT themselves be *unions of datatypes*. The *lexical space* of a *union of datatypes* is the union of the *lexical spaces* of each *constituent datatype*.

> *Note* — There is no requirement that the *lexical spaces* of each constituent *datatype* be disjoint.

Like any other *datatype*, a *union of datatypes* is a *term* with a *term name*. It MUST also specify a *pattern*.

> *Editorial note* — The following example describes a formalism for dates which has not yet been agreed nor even properly discussed. It is likely to change.

> *Example* — FHISO plans to define a *datatype* for representing dates which has the following *datatype name*:
>
> `https://terms.fhiso.org/dates/Date`
>
> It is a *union of datatypes* with the following two *constituent datatypes*:
>
> `http://www.w3.org/1999/02/22-rdf-syntax-ns#langString`
> `https://terms.fhiso.org/dates/AbstractDate`
>
> The former is the *language-tagged datatype* defined in §6.6.5 and is used to record dates that are described in a way that cannot be converted to a structured form without loosing information. The latter is an *abstract datatype* which serves as the *supertype* for various *structured datatypes* for dates.
>
> Because the `rdf:langString` *constituent datatype* is an *unstructured datatype*, every possible *string* is part of that of the *lexical space* of that *datatype*, and therefore also part of the *lexical space* of the *union of datatypes*. This means the *pattern* specified for the *union of datatypes* MUST allow every possible *string*, and so SHOULD be "`.*`".

> *Editorial note* — In the second draft of [CEV Concepts], which is where they were previously defined, *unions of datatypes* were not themselves *datatypes* as they lacked a *term name* to identify them, did not have a *pattern*, and could not be used as a *subtype* or *supertype*. As that draft noted, this is just a matter of nomenclature, and it seems more useful to make them proper *datatypes* in their own right.

A *union of datatypes* MAY contain *language-tagged datatypes*, *non-language-tagged datatypes*, or a mixture of both.

Each *constituent datatype* of a *union of datatypes* is a *subtype* of the *union of datatypes*. Whenever a *union of datatypes* is *supertype* of some other *datatype* it is defined to be a *trivial datatype*.

> *Note* — This means that every *datatype* has an unbounded set of *trivial supertypes* because every possible *union of datatypes* which has the *datatype* as a *constituent datatype* is a *supertype* of it. The set of *non-trivial supertypes* remains finite.

A *datatype* SHALL be a *supertype* of a *union of datatypes* if and only if it is a *supertype* of every *constituent datatype* of the *union of datatypes*.

> *Note* — Because the set of *supertypes* of each *constituent datatype* is unbounded, the set of *supertypes* of a *union of datatypes* is also unbounded as it contains every *union of datatypes* whose set of *constituent datatypes* is a superset of its own. The set of *non-trivial supertypes* remains finite.

> *Example* — In previous example, neither `rdf:langString` nor `AbstractDate` has any *non-trivial supertypes*, and therefore neither does the `Date` *union of datatypes*.

> *Example* — In a *union of datatypes* whose *constituent datatypes* are all *language-tagged datatypes*, each *constituent datatype* is a *subtype* of `rdf:langString` and therefore `rdf:langString` is a *non-trivial supertype* of the *union of datatypes*. This means the *union of datatypes* is classified as a *language-tagged datatype*.

The *class* of *unions of datatypes* is defined as follows:

Class definition

Name	`http://www.w3.org/2000/01/rdf-schema#Union`
Type	`http://www.w3.org/2000/01/rdf-schema#Class`
Superclass	`http://www.w3.org/2000/01/rdf-schema#Datatype`
Required properties	`http://www.w3.org/1999/02/22-rdf-syntax-ns#type`
	`https://terms.fhiso.org/types/pattern`
	`https://terms.fhiso.org/types/nonTrivialSupertypeCount`
	`https://terms.fhiso.org/types/isAbstract`
	`https://terms.fhiso.org/types/constituentDatatypeCount`

> *Note* — The main reason for defining a *class* for *unions of datatypes* is so that applications can distinguish *unions of datatypes* from other *datatypes* in order to check the number of *non-trivial supertypes* a *datatype* has, and whether this matches the number given in the `types:nonTrivialSupertypeCount` *property*. It also allows `types:constituentDatatypeCount` to be defined as a *required property*.

The *property* which denotes a *constituent datatype* of a *union of datatypes* is defined as follows:

Property definition

Name	`https://terms.fhiso.org/types/constituentDatatype`
Type	`http://www.w3.org/1999/02/22-rdf-syntax-ns#Property`
Range	`http://www.w3.org/2000/01/rdf-schema#Datatype`

As a way of checking for data integrity during *discovery*, an additional *property* is provided representing the number of *constituent datatypes* of the *union of datatype*:

Property definition

Name	`https://terms.fhiso.org/types/constituentDatatypeCount`
Type	`http://www.w3.org/1999/02/22-rdf-syntax-ns#Property`
Range	`http://www.w3.org/2001/XMLSchema#integer`

> *Editorial note —* Should this have a *range* of `xsd:nonNegativeInteger` instead?

> *Editorial note —* These two *properties* are likely to be changed in a future draft. Much as with the two *properties* for recording *supertypes* given in §6.2, a cleaner implementation would be to have a single `types:unionOf` *property* which is a list of the *constituent dataptyes* of the *union of datatypes*, however at the moment the data model does not support list-valued properties. This is a recognised deficiency in the current data model that is likely to be changed, but which requires considerable work.
>
> If and when list-valued *properties* are added to the data model, it may be that the `owl:unionOf` *property* defined in OWL should be reused instead of inventing our own *property*.

6.6 Standard datatypes

This standard recommends the use of the `xsd:string`, `xsd:boolean`, `xsd:integer` and `xsd:anyURI` *datatypes* defined in [XSD Pt2] to represent *strings*, *booleans*, *integers* and IRIs, respectively. They are described in the following subsections.

> *Note* — XML Schema does not give its types IRIs, but it does give them `id`s, and following the best practice advice given in §2.3 of [SWBP XSD DT] gives them IRIs like this:
>
> ```
> http://www.w3.org/2001/XMLSchema#integer
> ```
>
> These types are also recommended for use in RDF by §5.1 of [RDF Concepts]. RDF requires all *datatypes* to be identified by an IRI, and IRIs such as the one above are used for XML Schema *datatypes*.

This section also contains a summary of the `rdf:langString` *datatype* which is used heavily by FHISO technologies.

> *Note* — The *datatypes* described in this section are not intended to be an exhaustive list of *datatypes* usable with FHISO technologies, but rather is a list of the most common ones. Other XML Schema *datatypes* MAY also be suitable, as MAY *datatypes* from other third-party standards. Other FHISO standards will define additional *datatypes*.

6.6.1 The `xsd:string` datatype

Some FHISO standards make limited use of the `xsd:string` *datatype* defined in §3.3.1 of [XSD Pt2]. This is an *unstructured non-language-tagged datatype* which has the following properties:

Datatype definition	
Name	`http://www.w3.org/2001/XMLSchema#string`
Type	`http://www.w3.org/2000/01/rdf-schema#Datatype`
Pattern	`.*`
Supertype	*No non-trivial supertypes*
Abstract	`false`

It is a general-purpose *datatype* whose lexical space is the space of all *strings*; however it is not a *language-tagged datatype* and therefore it SHOULD NOT be used to contain text in a human-readable natural language.

> *Note* — This type is not the ultimate *supertype* of all *non-language-tagged* datatypes. This is because many other XML Schema *datatypes*, including `xsd:boolean` and `xsd:integer` are not defined as *subtypes* of `xsd:string` in XML Schema.

Use of this *datatype* is generally NOT RECOMMENDED: data that is in a human-readable form SHOULD use a *language-tagged datatype*, while data that is not human-readable SHOULD use a *structured datatype*.

If an application encounters a *string* with the `xsd:string` *datatype* in a context where a *language-tagged string* would be permitted, the application MAY change the *datatype* to `rdf:langString` and assign the *string* a *language tag* of und, meaning an undetermined language.

> *Note* — The `xsd:string` *datatype* is included in this standard in order to align this data model more closely with the RDF data model, and in particular the [CEV RDFa] bindings which use this *datatype* as the default when no *language tag* is present. The above rule allowing conversion to `rdf:langString` means that applications MAY ignore the `xsd:string` *datatype*.

6.6.2 The `xsd:boolean` datatype

A **boolean** is a *datatype* with precisely two logical values: **true** and **false**. FHISO standards represent *booleans* using the `xsd:boolean` *datatype* defined in §3.3.2 of [XSD Pt2]. This is a *structured non-language-tagged datatype* which has the following properties:

Datatype definition

Name	`http://www.w3.org/2001/XMLSchema#boolean`
Type	`http://www.w3.org/2000/01/rdf-schema#Datatype`
Pattern	`true|false|1|0`
Supertype	*No non-trivial supertypes*
Abstract	`false`

The *lexical space* of this *datatype* includes four different *strings* so that the two logical values of the *datatype* each have two alternative lexical representations. The value *true* MAY be represented by either "`true`" or "1", while the value *false* MAY be represented by either "`false`" or "0". *Conformant* applications SHALL NOT attach any significance to which of the alternative lexical representations is used, and MAY replace any instance of "1" in a *boolean string* with "`true`", or "0" with "`false`", but not vice versa. Where possible, the numeric representations, "0" and "1", SHOULD NOT be used.

> *Note* — The numeric representations are allowed because `xsd:boolean` allows them, and alignment with the XML Schema *datatype* is desirable as it is widely used in third-party standards. Appendix E.4 of [XSD Pt2] defines the alphabetic representations, "`true`" and "`false`", to be the canonical forms of the datatype, and this standard does similarly.

> *Note* — Even though the preferred forms of the allowed values of `xsd:boolean` are "`true`" and "`false`", which are in English, it is not a *language-tagged datatype* because the values MUST NOT be present in translated form. A Romanian dataset, for example, would still use the value "`false`" rather than translating it as "`adevărat`".

6.6.3 The `xsd:integer` datatype

FHISO uses the `xsd:integer` *datatype* defined in §3.4.13 of [XSD Pt2] to represent integers. It MUST NOT be used for values which are typically but not invariably integers.

> *Example* — Quantities that are invariably integer-valued do not occur all that frequently in genealogy. The page number of material being cited is normally an integer, but not invariably as a page number of a colour plate might be "facing p. 102" and prefatory pages are often numbered with Roman numerals. For this reason, page numbers should not be represented with integers. House numbers are similar, as it is not uncommon to find houses with numbers like "12A" in some countries.

> *Example* — The number of children born to a couple is an example of a value which is integer-valued. The number might be unknown or might only be known within certain bounds, but ultimately it is an integer: a couple cannot have 2.4 children.

This *datatype* can represent arbitrarily large integers, but unless otherwise stated, applications MAY opt not to support values greater than 2 147 483 647 or less than −2 147 483 648. In the event an unsup-

ported value is encountered, an implementation MAY handle it in an implementation-defined manner, but MUST NOT convert it to a different integer.

> *Note* — This permits applications to represent an `xsd:integer` as a signed 32-bit integer except where otherwise noted.

The *lexical space* of this *datatype* is the space of all *strings* consisting of a finite-length sequence of one or more decimal digits (U+0030 to U+0039, inclusive), optionally preceded by a + or - sign (U+002B or U+002D, respectively).

> *Example* — Thus the *string* "`137`" is within the *lexical space* of this *datatype*, but "`20.000`" and "`四十二`" are not, despite being normal ways of representing integers in certain cultures.

This *datatype* has several alternative representations of the same logical integer value. This arises because leading zeros are permitted, the + sign is optional, and the value `-0` is permitted. Applications MAY remove any leading + sign and any leading zeros preceding a non-zero digit, and MAY rewrite `-0` as 0.

> *Note* — This ensures that applications do not need to preserve the original lexical form of an integer, only its value.

This is a *structured non-language-tagged datatype* which has the following properties:

Datatype definition

Name	`http://www.w3.org/2001/XMLSchema#integer`
Type	`http://www.w3.org/2000/01/rdf-schema#Datatype`
Pattern	`[+-]?[0-9]+`
Supertype	*No non-trivial supertypes*
Abstract	`false`

> *Editorial note* — Its *supertype* is actually `xsd:decimal`, but this is not a supported *datatype* in this standard.

> *Note* — [XSD Pt2] also provides a range of signed and unsigned *datatypes* for integers represented in a specified number of bytes. The *datatypes* are `xsd:byte`, `xsd:short`, `xsd:int`, `xsd:long` and their unsigned equivalents. FHISO discourage the use of all of these *datatypes* in conjunction with FHISO standards as there very few genealogical contexts where an integer is required but where the value can be guaranteed always to fit in these fixed sized *datatypes*.

> *Editorial note* — This draft does not include specific guidance on the use of `xsd:positiveInteger` and `xsd:nonNegativeInteger`.

6.6.4 The `xsd:anyURI` datatype

FHISO uses the `xsd:anyURI` *datatype* defined in §3.3.17 of [XSD Pt2] to represent *strings* which are valid IRIs.

> *Note* — Despite the name of this *datatype* it is used to represent any IRI, not just those which are valid URIs. This misleading naming arose because XML Schema 1.0 did restrict the *datatype* to just URIs as IRIs were yet to be standardised. XML Schema 1.1 broadened the definition to include IRIs and FHISO uses this broader definition of the *datatype*.

Formally this is an *unstructured datatype* with no restrictions imposed on its *lexical space*; nevertheless, this *datatype* SHOULD only be used with *strings* which match the `IRI-reference` production in §2.2 of [RFC 3987] which matches both absolute and relative IRIs.

> *Note* — FHISO are following the definition in §3.3.17 of [XSD Pt2] in making this an *unstructured type*. XML Schema does this because the rules for validating an IRI are complex, subject to frequent updates, and dependent on IRI scheme.

Datatype definition

Name	`http://www.w3.org/2001/XMLSchema#anyURI`
Type	`http://www.w3.org/2000/01/rdf-schema#Datatype`
Pattern	`.*`
Supertype	*No non-trivial supertypes*
Abstract	`false`

6.6.5 The `rdf:langString` datatype

The `rdf:langString` *datatype* defined in §2.5 of [RDFS] is used as the general-purpose *unstructured language-tagged datatype*. No constraints are placed on the *lexical space* of this *datatype*; the only restriction placed on the use or semantics of this *datatype* is that it SHOULD contain text in a human-readable form.

Any *language-tagged datatype* that is not defined to be a *subtype* of some other *datatype* SHALL implicitly be considered to be a *subtype* of the `rdf:langString` *datatype*.

> *Note* — Together with the requirement in §6.4 that *language-tagged datatypes* MUST NOT be *subtypes* of *non-language-tagged datatypes*, this ensures that `rdf:langString` is the ultimate *supertype* of all *language-tagged datatypes*.

This *datatype* has the following properties:

Datatype definition

Name	`http://www.w3.org/1999/02/22-rdf-syntax-ns#langString`
Type	`http://www.w3.org/2000/01/rdf-schema#Datatype`
Pattern	`.*`
Supertype	*No non-trivial supertypes*
Abstract	`false`

> *Note* — Although this type is formally defined in the RDF Schema specification, this standard requires no knowledge of RDF; an implementer may safely use this *datatype* using just the information given in this section, and without reading [RDF Schema].

6.6.6 The `xsd:anyAtomicType` datatype

The `xsd:anyAtomicType` *datatype* defined in defined §3.2.2 of [XSD Pt2] is used as the **universal supertype** of all *datatypes*.

This *datatype* has the following properties:

Datatype definition

Name	`http://www.w3.org/2001/XMLSchema#anyAtomicType`
Type	`http://www.w3.org/2000/01/rdf-schema#Datatype`
Pattern	`.*`
Supertype	*No non-trivial supertypes*
Abstract	`true`

> *Note* — The `xsd:anyAtomicType` *datatype* is defined §3.2.2 of [XSD Pt2]. That standard does not define it as an *abstract datatype* as XML Schema's notion of abstract types does not extend to simple types. Nevertheless, `xsd:anyAtomicType` is treated specially by XML Schema in a way that is similar to this standard's definition of an *abstract datatype*. It is also not considered an "RDF-compatible XSD type" in §5.1 of [RDF Concepts] which means it SHOULD NOT be used as a *datatype* in RDF; again, this is similar to this standard's notion of an *abstract datatype*.

Any *non-language-tagged datatype* not defined to be a *subtype* of any other *datatype* SHALL implicitly be considered to be a *subtype* of the `xsd:anyAtomicType` *datatype*.

> *Editorial note* — In RDF, `xsd:anyAtomicType` is a *subclass* of `rdfs:Literal`. So is `rdf:langString`. This standard does not explicitly say this as FHISO's data model currently has no need for the `rdfs:Literal` *class*.

7 References

7.1 Normative references

[ISO 10646]

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

[FHISO Patterns]

FHISO (Family History Information Standards Organisation). *The Pattern Datatype*. First public draft.

[RDFS]

W3C (World Wide Web Consortium). *RDF Schema 1.1*. W3C Recommendation, 2014. (See https://www.w3.org/TR/rdf-schema.)

[RFC 2119]

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

[RFC 3987]

IETF (Internet Engineering Task Force). *RFC 3987: Internationalized Resource Identifiers (IRIs).* Martin Duerst and Michel Suignard, eds., 2005. (See https://tools.ietf.org/html/rfc3987.)

[RFC 5646]

IETF (Internet Engineering Task Force). *RFC 5646: Tags for Identifying Languages.* Addison Phillips and Mark Davis, eds., 2009. (See https://tools.ietf.org/html/rfc5646.)

[RFC 7230]

IETF (Internet Engineering Task Force). *RFC 7230: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing.* Roy Fielding and Julian Reschke, eds., 2014. (See https://tools.ietf.org/html/rfc7230.)

[RFC 7231]

IETF (Internet Engineering Task Force). *RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content.* Roy Fielding and Julian Reschke, eds., 2014. (See https://tools.ietf.org/html/rfc7231.)

[Triples Discovery]

FHISO (Family History Information Standards Organisation). *Simple Triples Discovery Mechanism*. First public draft.

[UAX 15]

The Unicode Consortium. “Unicode Standard Annex 15: Unicode Normalization Forms” in *The Unicode Standard, Version 8.0.0.* Mark Davis and Ken Whistler, eds., 2015. (See http://unicode.org/reports/tr15/.)

[XML]

W3C (World Wide Web Consortium). *Extensible Markup Language (XML) 1.1*, 2nd edition. Tim

Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, and John Cowan eds., 2006. W3C Recommendation. (See https://www.w3.org/TR/xml11/.)

7.2 Other references

[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.

[CEV Concepts]

FHISO (Family History Information Standards Organisation). *Citation Elements: General Concepts“. Third public draft. See https://fhiso.org/TR/cev-concepts.

[CEV RDFa]

FHISO (Family History Information Standards Organisation). *Citation Elements: Bindings for RDFa*. Third public draft. (See https://fhiso.org/TR/cev-rdfa-bindings.)

[CEV Vocabulary]

FHISO (Family History Information Standards Organisation). *Citation Elements: Vocabulary*. Exploratory draft.

[GEDCOM]

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

[IANA Lang Subtags]

IANA (Internet Assigned Numbers Authority). *Language Subtag Registry*. Online data file. (See http://www.iana.org/assignments/language-subtag-registry.)

[ISO 639-1]

ISO (International Organization for Standardization). *ISO 639-1:2002. Codes for the representation of names of languages — Part 1: Alpha-2 code*. 2002.

[ISO 639-2]

ISO (International Organization for Standardization). *ISO 639-2:1998. Codes for the representation of names of languages — Part 2: Alpha-3 code*. 1998. (See http://www.loc.gov/standards/iso639-2/.)

[ISO 639-3]

ISO (International Organization for Standardization). *ISO 639-3:2007. Codes for the representation of names of languages — Part 3: Alpha-3 code for comprehensive coverage of languages*. 2007.

[ISO 639-5]

ISO (International Organization for Standardization). *ISO 639-5:2007. Codes for the representation of names of languages — Part 5: Alpha-3 code for language families and groups*. 2008.

[ISO 3166-1]

ISO (International Organization for Standardization). *ISO 3166-1:2006. Codes for the repre-*

sentation of names of countries and their subdivisions – Part 1: Country codes. 2006. (See https://www.iso.org/iso-3166-country-codes.html.)

[ISO 15924]
ISO (International Organization for Standardization). *ISO 15924:2004. Codes for the representation of names of scripts.* 2004.

[N-Triples]
W3C (World Wide Web Consortium). *RDF 1.1 N-Triples.* David Becket, 2014. W3C Recommendation. (See https://www.w3.org/TR/n-triples/.)

[RDF Concepts]
W3C (World Wide Web Consortium). *RDF 1.1 Concepts and Abstract Syntax.* Richard Cyganiak, David Wood and Markus Lanthaler, eds., 2014. W3C Recommendation. (See https://www.w3.org/TR/rdf11-concepts/.)

[RDF Schema]
W3C (World Wide Web Consortium). *RDF Schema 1.1*. Dan Brickley and R. V. Guha, eds., 2014. W3C Recommendation. (See https://www.w3.org/TR/rdf-schema.)

[RFC 4122]
IETF (Internet Engineering Task Force). *A Universally Unique IDentifier (UUID) URN Namespace*. P. Leach, M. Mealling and R. Salz, ed., 2005. (See https://tools.ietf.org/html/rfc4122.)

[RFC 4648]
IETF (Internet Engineering Task Force). *RFC 4648: The Base16, Base32, and Base64 Data Encodings*. S. Josefsson, ed., 2006. (See https://tools.ietf.org/html/rfc4648.)

[RFC 7159]
IETF (Internet Engineering Task Force). *RFC 7159: The JavaScript Object Notation (JSON) Data Interchange Format*. T. Bray, ed., 2014. (See https://tools.ietf.org/html/rfc7159.)

[RFC 7469]
IETF (Internet Engineering Task Force). *Public Key Pinning Extension for HTTP*. C. Evans, C. Palmer and R. Sleevi, ed., 2015. (See https://tools.ietf.org/html/rfc7469.)

[SWBP XSD DT]
W3C (World Wide Web Consortium). *XML Schema Datatypes in RDF and OWL*. Jeremy J. Carroll and Jeff Z. Pan, eds., 2006. W3C Working Group Note. (See https://www.w3.org/TR/swbp-xsch-datatypes/.)

[UN M.49]
United Nations, Statistics Division. *Standard Country or Area Codes for Statistical Use*, revision 4. United Nations publication, Sales No. 98.XVII.9, 1999.

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

[XSD Pt1]

W3C (World Wide Web Consortium). *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures*. Shudi Gao (高殊镝), C. M. Sperberg-McQueen and Henry S. Thompson, ed., 2012. W3C Recommendation. (See https://www.w3.org/TR/xmlschema11-1/.)

[XSD Pt2]

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