MediaWiki

Wikibase/DataModel

< Wikibase

This is a living document, describing the conceptual data model behind Wikibase. It is not a specification of any concrete binding, implementation, mapping, or serialization.

The **data model of Wikibase** describes the structure of the data that is handled in Wikibase. In particular, it specifies which kind of information users can contribute to the system. On a more abstract level, the Wikibase data model provides a metamodel or ontology for describing real world entities. Such descriptions are concrete models for real world entities.

This document describes a conceptual model ("Which information do we have to support?") and does not specify how this data should be represented technically ("Which data structures should the software use?") or syntactically ("How should the data be expressed in a file?"). Separate documents describe the serialization of the Wikibase data model in JSON and in RDF (Resource Description Framework).

This specification is technical. A primer to the data model is also available that is more accessible (however, it is more ambiguous and less complete).

This document is extended by other documents to describe the data model of Wikibase extensions. The current extensions are:

- WikibaseLexeme - Lexeme Data Model.

Editorial Note: This document contains a number of "Editorial Notes". These are remarks that have been left by the editor to record some open issue or known problem. Eventually, all such notes will be addressed and removed.

History

- Latest Version (https://www.mediawiki.org/w/index.php?title=Wikibase/DataModel)
- Version 15-2-23 (https://www.mediawiki.org/w/index.php?title=Wikibase/DataModel&oldid=1422051)
- Version 15-1 (https://www.mediawiki.org/w/index.php?title=Wikibase/DataModel&oldid=1341417)
- Version 14-5 (https://www.mediawiki.org/w/index.php?title=Wikibase/DataModel&oldid=1014962)
- Version 14-4 (https://www.mediawiki.org/w/index.php?title=Wikibase/DataModel&oldid=955856)

Goals and requirements

The data model has the goal to clarify which information is stored in Wikibase. The model is extensible, but at any point in time it should document all things that are possibly stored in the system. It has two main goals:

Conceptual clarity: It should be clear *what* Wikibase can (and what it cannot) capture. It is not possible to capture all statements that one could make about the world (not even all that are important or reasonable). A balance must be found between expressive power and complexity/usability.

Technical documentation: Almost every component of Wikibase has to work with the data. To develop the software, it is therefore essential to have a common understanding of what the data is. Internally, the data can be represented quite differently (in objects, in a syntactic format, in a user interface, etc.): it is only important that each representation has a unique and unambiguous reading in terms of the data model.

There are a number of (sometimes conflicting) requirements that the data model should address in a balanced fashion:

- Coverage: the data model should be able to capture important data that occurs in Wikipedia in a natural way
- Simplicity: the data model should not be overly complex
- Extensibility: the data model should allow future extensions
- Flexibility: accessing and re-purposing data should be supported; the utility of the data should not be limited to one context
- Exchange: (parts of) the data should be exchangeable and have a clear meaning even outside the concrete system context of Wikidata
- Technical support: the data model should allow for adequate representations in existing data formats, e.g., JSON or RDF/OWL

The data model covers information that is expected to be relevant in the cause of the Wikidata project. Initially, only a part of it needs to be implemented, but it is important to ensure that the data model can also support later requirements (at least to the extent that they are in scope of the Wikidata project). Therefore, the below data model is not separated into phases.

There are also a number of things that the data model **is not supposed to do** (or that are at least beyond this document), in particular:

- Internal data structures: The data model is specified using Unified Modeling Language (UML), but this does not mean that it mandates the actual class structures to be used in implementation (in Wikidata or elsewhere). In many concrete situations, data can be stored in a more optimized way.
- Export formats: Data could be exported in many syntactic forms. Other documents will specify how this is done in each case.
- Formal semantics: This document explains what the data is intended to express, and gives concrete examples. However, it is not a completely precise specification of how to interpret this data formally: this will be given in a separate document.

Editorial Note: Provide documentation for (at least) the following bindings used by Wikibase: PHP, JavaScript, JSON, RDF. Additional bindings that may be particularly useful are Java and Python

Overview of the data model

The main purpose of Wikidata is to store data about things that are described by pages in Wikipedia (in any language). For example, one might want to store that the population of Berlin is 3,499,879. In this case, *Berlin* is the thing that is described, for example, by the article Berlin in English Wikipedia. In Wikidata, such a "thing" is represented as an **Item**. The Wikidata Item for Berlin would represent the thing that the Wikipedia article is about, not the Wikipedia article itself. Wikidata is concerned with recording facts about the subject of Wikipedia articles.

For every Item, various pieces of information are stored in Wikidata. First, there is some basic information that clarifies what the Item is about, such as the sitelink to a Wikipedia page in some language. There are also human readable labels and short descriptions that are used to help Wikidata users find the right Item. Second, there is a list of **Statements** that users have entered about the Item. Together, the information that is stored about one Item is called an **ItemDescription**.

Statements are the main approach of representing factual data, such as the population number in the above example. A Statement consists of two parts: a *claim* that something is the case (e.g., the claim "Berlin has a population of 3,499,879") and a list of *references* for that claim (e.g., a publication by the statistical office for Berlin-Brandenburg).

The reference is given by a **ReferenceRecord**, and the list of references is allowed to be empty (like in Wikipedia, editors can add Statements without a reference, which might later be improved by others who know about a suitable reference).

The claim that is made in a Statement can have various forms. The most common form is a single assignment of a **Value** to a **Property**. For example, *population* is a Property and the number *3,499,879* is a Value. Property-Value pairs can express many different claims, and Values can be numbers, dates and times, geographic coordinates, and many more. An important special case are values that are Items. For example, one could state that Berlin is the capital of Germany, where Germany has its own Item in Wikidata, that the Property *capital of* refers to. Properties are defined by users, so any Property can be created. As opposed to Items, Properties do not refer to Wikipedia pages, but they do specify a **Datatype** for the data that they (usually) store. The data stored about Properties forms a **PropertyDescription**.

The individual things that Wikidata talks about, including Items and Properties, are called **Entities**. All Entities are Values, but many kinds of Values are not Entities (examples of the latter kind include Values for numbers, strings, and geographic coordinates). This is so since Wikidata does not intend to store Statements about individual data values, such as strings or numbers (but it could store Statements about a number as a concept that is discussed on a Wikipage, in which case the number is represented by a Wikidata Item).

Property-Value pairs are not the only kind of claims that can be given in a Statement. It is also possible to say, for example, that a Property has *no* Values for the given Item. For example, one can say that a circle has *no* angles. Stating this can be relevant to distinguish it from the (common) case that the property has simply not been entered into Wikidata yet. Other things that one can say are related to classification, for example to state that Berlin is a city (i.e., "an instance of the class of all cities"). This is treated in a specific way since classification is important in many areas, e.g., in biologic taxonomies. For lack of a better name, any such basic assertion that one can make in Wikidata is called a **Snak** (which is small, but more than a byte). This term will not be relevant for using Wikidata (editors will not encounter it), but it is relevant for developers to avoid confusion with Statements or other claims.

For advanced usage, it is possible to make claims that consist of more than one Snak. For example, one might need to say that "the population of Berlin is 3,499,879, considering only the territory of the city, as estimated on 30 November 2011." Here, we have two additional Snaks that specify the territory the number refers to and the time when the measure was taken. It will be described below how exactly a claim can use additional Snaks.

How to read this document

This section explains our notation and general concepts that are used throughout this document.

Defining data structures in UML

The data structures that are specified in this document are usually described using UML class diagrams (see the Wikipedia page on UML for an introduction). We use only the following basic UML features:

- classes, represented as boxes
- abstract classes (conceptual classes that are not directly instantiated in data), represented as classes with names in *italics*
- class inheritance, represented by arrows with empty triangles as heads, pointing to the superclass
- class attributes ("member fields"), represented by "name: type" entries in classes
- associations/compositions, represented by blue lines with empty/filled diamonds on the side of the class that aggregates/composes many objects of the other class

The types of class members are either classes that are defined below, or one of the following basic datatypes:

Datatype	Explanation
String	a sequence of characters, possibly empty, where each character represents a Unicode (http://www.unicode.org/unicode/standard/) code point
integer	an integer number of arbitrarily large or small value
nonNegativeInteger	an integer number of arbitrarily large value greater than or equal to 0
decimal	a decimal number of arbitrarily large or small value, and arbitrary precision
IRI	an *absolute* Internationalized Resource Identifier according to RFC 3987; we do not consider relative IRIs
GlobalSiteIdentifier	a short string for identifying external sites, e.g., the language-related identification scheme of Wikipedia sites. (Note that this is different from BCP 47 (http://www.rfc-editor.org/rfc/bcp/bcp47.txt), e.g., there is no "en-US" in Wikipedia, just "en")
UserLanguageCode	a short string for identifying languages, based on the language preference setting of logged in Wikipedia users. (This might be more similar to BCP 47 (http://www.rfc-editor.org/rfc/bcp/bcp47.txt) but is not necessarily the same either; it is more fine-grained than a GlobalSiteIdentifier)

Numbers of arbitrarily large absolute value or precision can be represented as Strings, e.g., as described in the next section. For purposes of data access (e.g., retrieving values in numeric order), it will often be possible to approximate the value, e.g., by using a *double* value. However, technical formats such as float or double are not appropriate to represent user input accurately.

Wikidata Object Notation

UML describes data structures in a rather abstract way. To talk about concrete instances of these data structures, it is useful to have a simple serialization syntax for objects, which we call *Wikidata Object Notation* (WON). The WON is not intended to be used in implementations, but it is useful to give examples and to describe how the data model maps to other syntaxes, such as JSON or RDF.

The WON is described in this text along with the data model, and it will use exactly the same format. We give its simple grammar in BNF notation, using the following standard notation:

Construct	Syntax	Example
terminal symbols	strings in single quotes	`'PropertyDescription'`
a set of terminal symbols described in English	italic	*`a nonempty finite sequence of digits between 0 and 9`*
nonterminal symbols	boldface	**`Statement`**
zero or more	curly braces	`{` **`Statement`** `}`
zero or one	square brackets	`[` **`Statement`** `]`
alternative	vertical bar	**`Item`** `|` **`Property`**

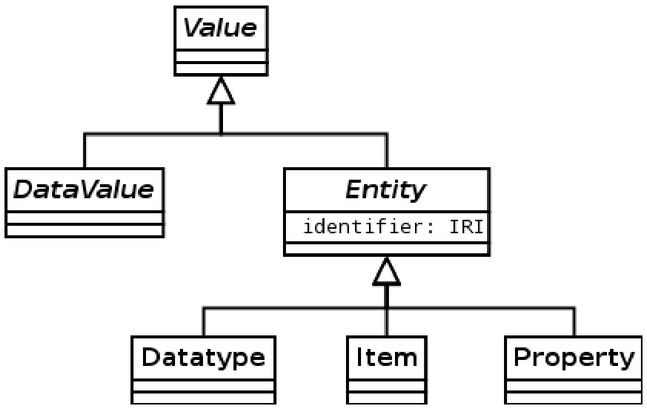
The basic datatypes that were described above can be serialized in WON as follows:

`quotedString`	:=	*a finite sequence of characters in which " and \ occur only in pairs of the form \" and \\, enclosed in a pair of " characters*
`integer`	:=	['-'] **nonNegativeInteger**
`nonNegativeInteger`	:=	*a nonempty finite sequence of digits between 0 and 9*
`decimal`	:=	**integer** ['.' **nonNegativeInteger**]
`IRI`	:=	*an IRI as defined in RFC 3987, enclosed in a pair of < and > characters*
`GlobalSiteIdentifier`	:=	*a nonempty finite sequence of Latin characters between a and z, and -*
`UserLanguageCode`	:=	*a nonempty finite sequence of Latin characters between a and z, and -*

We follow common conventions for escaping "-quoted strings, and of enclosing IRIs with < >.

Values

Values are basic objects of Wikidata, that only represent one particular thing. Items represent topics of Wikipedia pages, Properties represent the properties that Items (or other Entities) can have, DataValues represent individual values of a particular Datatype (a number, a geographic coordinate, etc.). The kinds of Values and their structure is shown in the following figure:



Various kinds of Values can be the subject of basic statements (Snaks): they are called Entities. Entities are identified in a uniform way using Uniform Resource Identifiers (URIs), or rather Internationalized Resource Identifiers (IRIs) that also allow Unicode symbols. Since an IRI is a global identifier, no two different Entities may have the same IRI. Hence, all entities can be represented by their IRI alone, without noting what kind of Entity they are. (Items have IRIs of the form `https://www.wikidata.org/entity/Qnnn` and Properties have IRIs of the form `https://www.wikidata.org/entity/Pnnn`)

```
   Value :=  DataValue | Entity
  Entity :=  Datatype | Item | Property
Datatype :=  IRI
    Item :=  IRI
Property :=  IRI
```

In contrast to Entities, DataValues are not identified by an IRI but can simply be viewed as compound values that are identified by their content. Values without an IRI can still be named internally or in exports, but the identifiers that are used in this case will usually consist in the actual content (or a hash thereof).

Note that we distinguish single Entities (e.g., an Item about Berlin) from Descriptions of Entities (e.g., the collection of information that is stored about that Item about Berlin).

Items

Items are Entities that are typically represented by a Wikipage (at least in some Wikipedia languages). They can be viewed as "the thing that a Wikipage is about," which could be an individual thing (the person Albert Einstein), a general class of things (the class of all Physicists), and any other concept that is the subject of some Wikipedia page (including things like History of Berlin).

The IRI of an Item will typically be closely related to the URL of its page on Wikidata. It is expected that Items store a shorter ID string (for example, as a title string in MediaWiki) that is used in both cases. ID strings might have a standardized technical format such as "Q1234567890" and will usually not be seen by users. The ID of an Item should be stable and not change after it has been created.

The exact meaning of an Item cannot be captured in Wikidata (or any technical system), but is discussed and decided on by the community of editors, just as it is done with the subject of Wikipedia articles now. It is possible that an Item has multiple "aspects" to its meaning. For example, the page Orca describes a species of whales. It can be viewed as a *class* of all Orca whales, and an individual whale such as Keiko would be an element of this class. On the other hand, the species Orca is also a concept about which we can make individual statements. For example, one could say that the binomial name (a Property) of the Orca species has the Value "Orcinus orca (Linnaeus, 1758)."

However, it is intended that the information stored in Wikidata is generally *about* the topic of the Item. For example, the Item for History of Berlin should store data about this history (if there is any such data), not about Berlin (the city). It is not intended that data about one subject is distributed across multiple Wikidata Items: each Item fully represents one thing. This also helps for data integration across languages: many languages have no separate article about Berlin's history, but most have an article about Berlin.

Sitelinks

An Item can be linked to pages on other wikis via sitelinks. This is used by Wikipedia to link other language versions of an article (since the different language-specific instances of Wikipedia are technically separate wikis). Note that while an Item can have multiple sitelinks to different wikis, it cannot have multiple sitelinks to the same wiki. Sitelinks can additionally have a set of "badges" associated with the page (such as "featured article"). Badges are also represented as Items.

Properties

Properties are Entities that describe a relationship between Items (or other Entities) and Values of the property. Typical properties are *population* (using numbers as values), *binomial name* (using strings as values), but also *has father* and *author of* (both using Items as values).

Like Items, Properties are identified by an IRI that will probably be closely related to their URL on Wikidata. However, the IDs will be based on a different naming scheme so that no confusion with Items is possible. For example, a typical identifier string used in a Property ID could be "P123456789". The ID of a Property should be stable and not change after it has been created.

Properties are treated differently to Items because they do not usually have a page in Wikipedia. While there is a page en:population, it does not describe the relationship between a region and its number of (human) inhabitants, but rather the noun *population*. This can be close to the property, but it can also lack important information. For example, the page en:parent describes what a parent is, but there are multiple related properties, especially *parent of* and *has parent* (which have a very different meaning). Wikipedias do not usually contain specific articles about such properties, only about the concepts that they relate to.

As another difference from Items, Properties can have a Datatype that specifies what kind of values users will normally enter for them. Note, however, that the data model does not require strict typing for Properties in Snaks (see below).

Datatypes

A Datatype is an Entity that determines the type and shape of the values that can be assigned to a Property. There are various common Datatypes, and each must be handled specifically by the software (for example, the user interface will be different depending on the type of data that is edited). Therefore, the Datatypes that are supported by Wikidata can only be extended by software developers, not by editors on the site. However, it might be possible to customize some Datatypes when using them for a Property (e.g., one might be able to say that a Property should only accept numbers without decimal digits, i.e., integers).

Most Datatypes are not *primitive* in the sense that their values consist of only one single value of a type that is commonly found in programming languages. For example, geographic coordinates are an important type of data in Wikidata, but they have an internal structure (e.g., specifying a latitude, longitude, and possibly a height).

More information about the Datatypes available in Wikidata is given in the respective section below.

DataValues

DataValues are Values that are not Entities. They represent values of a particular Datatype, such as a particular number or point in time. Details on the available DataValues and their according types is given in the respective section below.

Snaks

Snaks are the basic information structures used to describe Entities in Wikidata. They are an integral part of each Statement (which can be viewed as collection of Snaks about an Entity, together with a list of references).

Many of the Snaks are based on similar pieces of information, yet we distinguish Snaks that are intended to have a different meaning. This is useful in many places. Typically, Snaks of different meaning will be represented differently in the user interface.

```
        Snak :=  PropertySnak
PropertySnak :=  PropertyValueSnak | PropertySomeValueSnak | PropertyNoValueSnak
```

Note that currently, all Snaks are PropertySnak. Other types of Snaks that are not PropertySnak may be defined in the future.

PropertyValueSnak

A PropertyValueSnak describes that an Entity has a certain Property with a given Value. Note that it is not required that Value belongs to the Datatype that is currently given to the Property in the system. In general, the UI and API of Wikidata will only allow Values that match the given Datatype, but if the Datatype is changed, then it will not be possible to update all stored data immediately. Moreover, if the Datatype is changed back to its earlier value, it might be possible to continue using existing data that was not changed. This is the main reason for not limiting the data model to strictly typed Properties.

Please also note that the data model does not actually define a unique Datatype for each Property: it just specifies how Datatype assignments would be represented; a unique Datatype is only obtained in a closed system where every Property has a globally unique Datatype assignment.

The Wikidata Object Notation for PropertyValueSnaks is as follows:

```
PropertyValueSnak :=  'PropertyValueSnak(' Property Value ')'
```

Here and below, we omit the names of attributes (e.g., "subject") in WON, and simply encode their values positionally. We do not specify any delimiters between the arguments in this notation. It is silently assumed that whitespace is introduced to avoid ambiguities.

Example: Many basic kinds of data are naturally expressed by assigning Values to Properties. Some examples:

- Berlin (subject) has a population (property) of *3499879* (value).
- Georgia (subject) has the capital (property) Tbilisi (value).
- Gandhi (subject) was born on (property) 2 October 1869 (value).

Obviously, each Value in these statements would refer to one clearly identified object (e.g., our label "Georgia" above is surely not precise enough). We omit such details for simplicity here. Also note that Snaks do not mention the subject to which they refer (Berlin, Georgia, Gandhi); this is given by the context in which a Snak is used (typically as part of a Statement).

PropertyNoValueSnak

A PropertyNoValueSnak describes that an Entity has no values for a certain Property.

```
PropertyNoValueSnak := 'PropertyNoValueSnak(' Property ')'
```

Example: In some cases, we want to emphasize that a property value has not just been left out (or not entered yet) but that it really does not exist. Some examples:

- Circle (subject) has no angle (property).
- Mount Everest (subject) has no parent peak (property).

Such statements should only be made in cases where one could otherwise expect an incompleteness. It is not intended that Wikidata stores all things that are not the case (e.g., "The Pacific Ocean has no angle").

PropertySomeValueSnak

A PropertySomeValueSnak describes that an Entity has some value for a certain Property, without saying anything about this value. This can be used if the value of a property is unknown.

```
PropertySomeValueSnak := 'PropertySomeValueSnak(' Property ')'
```

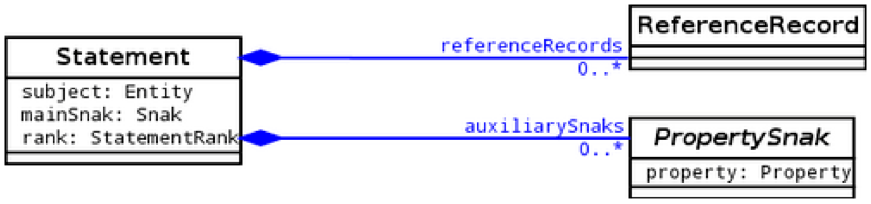
Example: The information that a property has some value can be important and useful, even if the value is not known. For example:

- Ambrose Bierce (subject) has an unknown date of death (property), yet we can be certain that he is not among the living persons.

Such statements should only be made if no concrete date is known. Wikidata does not support constraints on unknown values ("William of Ockham died in 1347 or 1348") but it does support precision on some types of data values ("William of Ockham died in the 1340s") and it does support different (possibly conflicting) values from multiple sources.

Statements

Statements describe the claim of a statement and list references for this claim. Every Statement refers to one particular Entity, called the *subject* of the Statement. There is always one *main* Snak that forms the most important part of the statement. Moreover, there can be zero or more additional PropertySnaks that describe the Statement in more detail. These qualifier Snaks (or "qualifiers" for short) store additional information that does not directly refer to the subject (e.g., the time at which the main part of the statement was valid). References are provided as a list (the order is significant in some contexts, especially for displaying a main reference). The complete structure is described as follows:



The individual components have the following meaning:

- subject: the Entity that the statement is about
- mainSnak: the main Snak of the statement
- rank: a StatementRank that will be used for simplifying the selection of Statements; see for more detail below
- referenceRecords: the list of references, see below for details
- qualifierSnaks: optional list of additional PropertySnaks that qualify the statement

`Statement :=` 'Statement(' **Entity Snak** {**PropertySnak**} {**ReferenceRecord**} **Rank** ')'

Example: A simple statement could just contain any of the Snaks in the above examples. The use of qualifier Snaks is illustrated in the following examples:

- "Obama was US Senator from Illinois from January 3, 2005 to November 16, 2008":
  - subject "Obama"
  - mainSnak of type PropertyValueSnak with property "US Senator from" and value "Illinois"
  - qualifier Snak of type PropertyIntervalSnak with property "in office" and interval "January 3, 2005 to November 16, 2008".
- "Harry Potter and the Philosopher's Stone was starring Emma Watson in the role of Hermione Granger":
  - subject "Harry Potter and the Philosopher's Stone"
  - mainSnak of type PropertyValueSnak with property "starring" and value "Emma Watson"
  - qualifier Snak of type PropertyValueSnak with property "played character", and value "Hermione Granger"
- "1.6% of people living in Austria are Turks":
  - subject "Austria"
  - mainSnak of type PropertyValueSnak with property "ethnic group", and value "Turks"
  - qualifier Snak of type PropertyValueSnak with property "percentage", and value "1.6%" (here "%" could be represented like the unit of measurement of quantities).

In each case, there are other ways to capture the respective information. Like in Wikipedia, it is left to the community to agree on uniform ways of expressing such things. Often, there are good reasons to prefer one representation over the other. For example, there are cases where a country is known to have inhabitants of some ethnic group, while the percentage of that group is not known; then the qualifier Snak could simply be omitted.

Ranks of Statements

The ranks provide a simple selection/filtering criterion in cases where there are many Statements for some property. There are three possible ranks, which have roughly the following meaning:

1. **Preferred** statements refer to the most important and most up-to-date information that should be used per default in most contexts (example: only most recent population figures for Berlin would be shown in the Wikipedia infobox for Berlin). Note that there may be multiple preferred statements. This may imply a multi-valued property (e.g. a person's children), or a disagreement (diverging population figures given by different sources).
2. **Normal** statements contain relevant information that is believed to be correct but that may be too extensive for showing it by default (example: historic population figures for Berlin for many years).
3. **Deprecated** statements that may not be considered reliable or that are even known to contain errors (example: a statement that documents a wrong population figure that was published in some historic document; in this case the

statement is not wrong – the historic document that is given as a reference really made the erroneous claim – yet the statement should not be used in most cases).

This model is intentionally left coarse and simple. The three levels translate to different treatments in data access, UI (e.g., what is displayed by default), and export (one could, e.g., have an export with only the preferred and normal Statements). The ranks may also be useful for protecting Statements from editing (e.g., by protecting only preferred and normal statements). More fine-grained rankings do not seem to have such a clear interpretation and would thus increase the UI complexity unnecessarily. Having only two ranks (or no ranks at all), on the other hand, would make it harder to cope with Statements that are not trusted, known to contain wrong claims, or simply unpatrolled (if ranks are used for protection).

Another useful concept can be constructed based on the ranks defined above: the "best rank" for the Statements about a given Property with respect to a given Item. If there is at least one Statement with *preferred* rank about the property (in the context of a given Item), the best rank for that property is *preferred*. Otherwise, the best rank is *normal*. Correspondingly, the "best Statements" about a given Property in the context of a given Item are the ones that have the best rank for that Property.

ReferenceRecords

ReferenceRecords are intended to store information about some source, represented as a set of Snaks. In the simplest case, the source can be represented by a single Snak, e.g. providing a URL. But SourceRecords can also be more complex or, e.g. consisting of Snaks representing the title, author, and publisher of a book, along with chapter and page of the cited information.

```
ReferenceRecord := 'ReferenceRecord(' Snak {Snak} ')'
```

EntityDescriptions of Items and Properties

EntityDescriptions are collections of information about an entity, and they mainly serve as data containers that can be interpreted as sets of Snaks with some further attributes (that could also be represented as Snaks, if desired). In addition, EntityDescriptions may support lexical information that can be used for displaying, searching, or referencing the respective entity.

We define **PropertyDescription** and **ItemDescription** subtypes of EntityDescription that correspond to entities of the respective type, Item and Property. In particular, all Statements of an ItemDescription must use the expected Item as the subject of their main Snak, and all Statements of a PropertyDescription must use the expected Property as the subject of their main Snak.

EntityDescriptions can contain basic lexical information. Each ItemDescriptions and PropertyDescriptions supports internationalized labels, descriptions, and aliases. The overall structure of ItemDescription and PropertyDescription can be defined as follows:

```
  EntityDescription := ItemDescription | PropertyDescription
    ItemDescription := 'ItemDescription(' Item [MultilingualTextValue] [MultilingualTextValue]
                       [MultilingualMultiTextValue] {Statement} ')'
PropertyDescription := 'PropertyDescription(' Property [MultilingualTextValue] [MultilingualTextValue]
                       [MultilingualMultiTextValue] {Statement} ')'
```

The three types of lexical information supported by ItemDescription and PropertyDescription are labels, descriptions, and aliases. Labels and descriptions are MultilingualTextValues, aliases are MultilingualMultiTextValues: for any given language, an EntityDescription may have at most one label and at most one description, but any number of aliases. Their respective purposes are:

- **label:** the main label to be used for representing the described Entity in Wikidata in various languages, e.g., *Georgia* could be an English label.
- **description:** a brief description to clarify the meaning of the label (which may be ambiguous), e.g., *a country in the Caucasus* could be an English description. The intended use is mainly disambiguation when displaying multiple entities with the same label, for selection.
- **alias:** alternative labels in various languages, used mainly for searching for items by their name.

The lexical information in EntityDescriptions may be used as unique *keys* as follows:

- For ItemDescriptions, the combination of label and description is a key for one particular language, if both label and description are defined.
- For PropertyDescriptions, the label is a key for one particular language. Since properties can be identified by their label (given a language), the label is also called the property's "name".

Planned Feature:

- For PropertyDescriptions, any alias is a key for one particular language.

Planned Feature:

The structure of PropertyDescriptions may be expanded in the future, to cover the following:

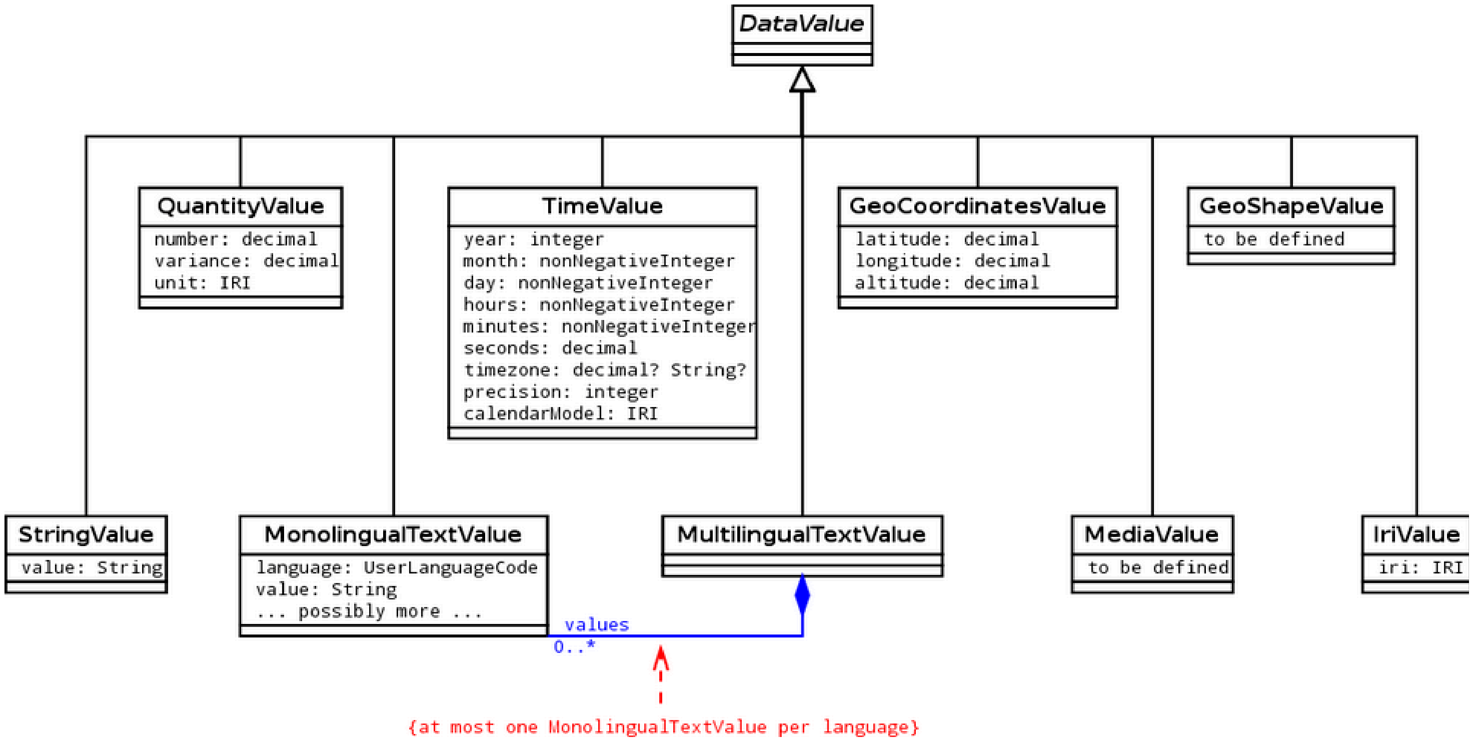
- A long description that provides more details on the meaning of the property and its proper usage. Internationalized.
- A flag to indicate that statements do not usually need a reference to be credible ("self-evident"), at least if used without qualifier Snaks. For example, it would be tedious to enter a reference for the fact that an IMDB URL is about a particular movie, yet maintenance interfaces that warn about unsourced statements should not show all statements of this kind.
- A flag to indicate that a Property of datatype MonolingualTextValue provides a label that should be used as an alias in that language, e.g., in search.
- An IRI prefix that allows values associated with the Property to be mapped to external vocabularies.
- Mappings to other vocabularies, indicating (rough or exact) equivalence of the Property with a predicate from a standard vocabulary like Dublin Core of FOAF.
- Hints on the qualifier Snaks that are typically required/given for some property. Users should generally not have to manually add qualifier Snaks but rather see empty fields for suggested Snaks if they usually belong to the property.

Note that this information could be expressed using Statements on the PropertyDescriptions, without extending the structure of PropertyDescriptions. In order to make use of such Statements, the processing software would have to have knowledge of the meaning of the properties used to make such statements.

Datatypes and their Values

Datatypes[1] are Entities that specify the format of Property Values. The set of Datatypes in Wikibase is system-defined (it can be extended, but only by developers). Every Datatype has a fixed IRI, that is also system-defined.

For every Datatype, there is one particular form of Value that is used to represent Values of that type. Wikibase distinguishes between Values that can be the subject of Snaks, called Entities, and Values that are not the subject of Snaks, called DataValues. The following is an overview of all DataValues:



`DataValue :=` **QuantityValue** | **StringValue** | **TimeValue** | **GeoCoordinateValue** | **GeoShapeValue** | **MediaValue** | **IriValue** | **MonolingualTextValue** | **MultilingualTextValue**

Quantities

- Datatype IRI: http://wikidata.org/vocabulary/quantity
- Value: QuantityValue

A QuantityValue represents a decimal number, together with information about the uncertainty interval of this number, and a unit of measurement. The decimal number is represented as a string using the lexical form of XML Schema *decimal*. The attributes are:

- amount: decimal the quantity's main value
- lowerBound: decimal the quantity's lower bound (optional)
- upperBound: decimal the quantity's upper bound (optional)
- unit: IRI or "1" to indicate the unit "unit".

`QuantityValue :=` 'QuantityValue(' **decimal** [**decimal**] [**decimal**] **IRI** ')'

The given *amount* is interpreted as the main value of the QuantityValue. The optional *lowerBound* and *upperBound* specify how far the true value of the represented quantity could deviate from the *number* in positive or negative direction. This allows to capture expressions such as *12300 +/- 50*. For many practical purposes, only the number might be used (e.g., for sorting and query answering), but the variance can provide valuable information for presentation (e.g., for selecting reasonable precision in unit conversions). If the lower and upper bound are not present, the uncertainty is unspecified.

The exact interpretation of the uncertainty interval provided with *lowerBound* and *upperBound* is unspecified. Depending on context, it may represent hard limits on the value, or the interval may just describe the 66 or 95 percentile interval of a normal distribution.

In the Wikibase UI, quantities and their bounds are input together as a string: for instance, "4~" will give an amount of 4, a lower bound of 3.5 and an upper bound of 4.5. Strings must currently match the following regular expression to be parsed in this way:

```
^\s*((?:[-+]\s*)?(?:[\d,]+\.\d*|\.?\d+)(?:[eE][-+]?\d+)?)\s*(?:([~!])|(?:\+/?-|±)\s*((?:[-+]\s*)?(?:[\d,]+\.\d*|\.?\d+)(?:[eE][-+]?
\d+)?)|)\s*$
```

The *unit* specifies a physical quantity that the number refers to. It is represented as a IRI rather than as a String, since a string like "m" might represent different units in different contexts. The value should be meaningful independently of the declaration information for its Property (from which more details about units could possibly be obtained), hence the unit is a full IRI. In practice, this IRI might be the IRI refering to an *ItemDescription* representing the desired unit, or be taken from a standard vocabulary for units, like QUDT[1] (http://qudt.org/1.1/vocab/unit).

Editorial Note: It is not clear yet how exactly the variance information is to be used to ensure "reasonable" unit conversion and display. There are special cases such as English body sizes that may need special treatment ("5 feet, 3 inches") but this should not affect the data model. Describe plans for unit conversion

Dates and times

- Datatype IRI: http://wikidata.org/vocabulary/datatype_time
- Value: TimeValue

~~The calendar model used for *saving* the data is always the proleptic Gregorian calendar according to ISO 8601, but the Calendar model used for *displaying* the data is given by the saved Calendar model.~~[2]

A TimeValue represents a point in time that might be imprecise (e.g., if only a year is given). For practical purposes (e.g., sorting values), the value will often be interpreted to be exact by filling the missing positions with more details. The structure of values of this type is as follows:

- time: timestamp in a format resembling ISO 8601, the year always being signed and having between 1 and 16 digits, e.g. +2013-01-01T00:00:00Z. The month, day and time will be set to zero if they are unknown; the *precision* field should be relied on to determine which time digits are meaningful. The Z is meaningless and the time zone should be determined from the *timezone* field. For the purposes of this description, the timestamp uses historical numbering where the year 0 is undefined and the year 1 BCE is represented as -0001; note, however, that this is not the case in all serializations (see JSON and RDF for details).
- precision: shortint. The numbers have the following meaning: 0 - billion years, 1 - hundred million years, ..., 6 - millennium, 7 - century, 8 - decade, 9 - year, 10 - month, 11 - day, ~~12 - hour, 13 - minute, 14 - second~~[3]. Note that centuries and millennia do not strictly align with the most significant digits of the timestamp: for positive timestamps, centuries and millennia begin on years 1 modulo 100/1000 and end on years 0 modulo 100/1000, and vice versa for negative timestamps. For example, the 18th century begins in the year 1701 and ends in the year 1800, and the 2nd millennium BCE begins in the year -2000 (2000 BCE) and ends in the year -1001 (1001 BCE). (On the other hand, decades align with the most significant digits: the 1980s begin in 1980 and end in 1989.)
- after: integer. If the date is uncertain, how many units before the given time could it be? the unit is given by the precision.
- before: integer. If the date is uncertain, how many units after the given time could it be? the unit is given by the precision.
- timezone: signed integer. Timezone information as an offset from UTC in minutes. For dates before the modern implementation of UTC in 1972, this is the offset of the time zone from universal time. Before the implementation of time zones, this is the longitude of the place of the event, expressed in the range −180° to 180° (positive is east of Greenwich), multiplied by 4 to convert to minutes.
- calendarmodel: URI identifying the calendar model ~~that should be used to *display* this time value. Note that time is always *saved* in proleptic Gregorian, this URI states how the value should be displayed~~[2].

~~Interpretation of dates follow ISO 8601:~~[4]

- Presently dates refer to the (possibly proleptic) Gregorian or Julian calendar, as specified by the *calendarmodel* field.[5] Any future extension to other calendars is likely to require a drastically different format for the *time* field when used with such other calendars.
- ~~There is a year number 0 that refers to the year that is commonly called 1 BC(E).~~

Examples

If you have something like "between 1846 and 1855", you can use the "before" and "after" fields of the time value:

```
time: "+00000001850-00-00T00:00:00Z",
precision: 9,
before: 4,
after: 5
```

This means the "main" value is 1850, given as a year, with a lower bound four years before and an upper bound 5 years after the "main" value (before and after are given in the unit specified by the precision value). The "main" value is what is going to be displayed per default; it will also be used for sorting query results (once we have queries).

This is a bit complicated, but should allow you to actually represent uncertain dates. We made it so you can be precise about the uncertainty.

Web resources and other IRIs

- Datatype IRI: http://wikidata.org/vocabulary/datatype_iri
- Value: IriValue

`IriValue` := 'IriValue(' **IRI** ')'

An IriValue represents an arbitrary IRI that follows RFC 3987. If the protocol part is supported by MediaWiki, a hyperlink might be displayed, but the Datatype as such does not require such protocols, and generally it is not required that all IRIs work as URLs. For example, the "tel:" protocol (RFC 3966) might also be allowed.

Geographic locations

- Datatype IRI: http://wikidata.org/vocabulary/datatype_geocoords
- Value: GlobeCoordinateValue

A coordinate is represented as:

- a latitude (decimal, no default, 9 digits after the dot and two before, signed)
- a longitude (decimal, no default, 9 digits after the dot and three before, signed)
- a precision (decimal, representing degrees of distance, defaults in the formatter to 1/3600 if the value is not a finite value greater 0 and defaults in the UI to a guessed value, 9 digits after the dot and three before, unsigned, used to save the precision of the representation)
- a coordinate system or globe (identified by an URI, defaults to http://wikidata.org/entity/Q2, i.e. Q2, the Earth, which means WGS84). Any such geodesic system must imply the globe for which it is used (and should be displayed as simply the globe in most cases).

`GlobeCoordinateValue` := 'GlobeCoordinateValue(' **decimal decimal decimal URI** ')'

Geographic shapes

- Datatype IRI: http://wikidata.org/vocabulary/datatype_geoshapes
- Value: GeoShapeValue

Editorial Note: This needs to be specified. It is likely that Wikibase will simply refer to an existing standard for representing geographic shapes here, e.g., WKT or GeoJSON.

Wikidata items

- Datatype IRI: http://wikidata.org/vocabulary/datatype_items
- Value: Item

Items in Wikibase are represented by Item as explained in the section on Values above. While not subtypes of DataValue, we list them here to define the IRI for the respective datatype. It is not planned to have user-defined properties for other types of Entities for now.

Wikidata properties

- Datatype IRI: http://wikidata.org/vocabulary/datatype_properties
- Value: Property

Item attributes in Wikibase are represented by Properties as explained in the section on Values above. While not subtypes of DataValue, we list them here to define the IRI for the respective datatype. It is not planned to have user-defined properties for other types of Entities for now.

Media

- Datatype IRI: http://wikidata.org/vocabulary/datatype_media
- Value: MediaValue

Editorial Note: Media is represented by a dedicated Datatype since Media items should be handled in a specific way. Moreover, it might be useful to have additional metadata for Media objects. To be defined.

Strings that are not translated

- Datatype IRI: http://www.w3.org/2001/XMLSchema#string
- Value: StringValue

```
StringValue := 'StringValue(' String ')'
```

Strings are represented by StringValues. All strings are considered as sequences of Unicode glyphs. As opposed to multilingual and monolingual texts, strings do not contain any language information, and are typically used directly only for strings that do not belong to a language, e.g., the post code of a UK city.

Note: Wikibase enforces that strings are at least one character long and disallows strings that match the regular expression `^\s|[\v\t]|\s$` (disallowing any strings that start or end with whitespace or contain vertical whitespace such as newlines).

Monolingual texts

- Datatype IRI: http://wikidata.org/vocabulary/datatype_monotext
- Value: MonolingualTextValue

MonolingualTextValues are Values that represent a phrase in some language. In particular, their content could also be pronounced (and be associated with pronunciation information or audio versions). The attributes of MonolingualTextValues are:

- language: UserLanguageCode
- value: String

```
MonolingualTextValue := 'MonolingualTextValue(' UserLanguageCode String ')'
```

Multilingual texts

- Datatype IRI: http://wikidata.org/vocabulary/datatype_multitext
- Value: MultilingualTextValue

```
MultilingualTextValue := 'MultilingualTextValue(' {MonolingualTextValue} ')'
```

MultilingualTextValues are Values that represent a phrase in many languages. This is different from representing individual Values for each language, since it also captures the information that all of the Values are direct translations (otherwise, if a Property has multiple MonolingualTextValues in each language, it would not be clear which values belong together). MultilingualTextValues store a list of MonolingualTextValues, but at most one for each UserLanguageCode.

Multilingual multi-texts

- Datatype IRI: http://wikidata.org/vocabulary/datatype_multimultuitext
- Value: MultilingualMultiTextValue

```
 MonolingualMultiTextValue := 'MonolingualTextValue(' UserLanguageCode {String} ')'
MultilingualMultiTextValue := 'MultilingualMultiTextValue(' {MonolingualMultiTextValue} ')'
```

MultilingualMultiTextValue are Values that represent a list of phrases in several languages. There is *no* implied relationship between the list of phrases in the different languages. MultilingualMultiTextValue store a list of MonolingualMultiTextValues.

Serializations

- Wikibase/DataModel/JSON
- Wikibase/Indexing/RDF Dump Format

Notes

1. https://www.wikidata.org/wiki/Special:ListDatatypes
2. phabricator:T88437
3. Wikidata only accepts precisions up to 11. phabricator:T57755
4. T99674
5. A proleptic calendar is a a calendar used for dates before the introduction of the calendar by applying the rules of the calendar in reverse from dates that are known with certainty. In the case of the Julian calendar, it was introduced in 45 BCE but was implemented incorrectly until 1 March 4 CE, and not enough records survive to convert dates in this range exactly. Dates in this range should be stated in the proleptic Julian calendar with

appropriate precision rather than the Julian calendar as observed in Rome at that time. Gregorian dates earlier than 15 October 1582 and Julian dates earlier than 1 March 4 CE are proleptic.

See also

- Wikidata Glossary
- Wikibase/DataModel/Primer
- [WIP] Mapping of functional traits to Wikibase Domain concepts