The OBO Flat File Format Specification, version 1.2

John Day-Richter <john.richter@aya.yale.edu>

Revision History

Revision 1.2	November 16, 2004	JD

Table of Contents

last revision: May 2nd, 2006

Abstract

The OBO flat file format is an ontology representation language. The concepts it models represent a subset of the concepts in the OWL description logic language, with several extensions for meta-data modeling and the modeling of concepts that are no supported in DL languages.

The format itself attempts to achieve the following goals:

- Human readability
- Ease of parsing
- Extensibility
- Minimal redundancy

About this file

This file is maintained in asciidoc format in the Gene Ontology cvs repository at cvs.sourceforge.net, in go-dev/docs/obo_format_spec.txt. For information about asciidoc, go to http://www.methods.co.nz/asciidoc/.

OBO Syntax

The format is similar to the tag-value format of the GO definitions file, with a few modifications. One important difference is that *unrecognized tags in any context do not necessarily generate fatal errors* (although some parsers may decide to do so; See Parser Requirements below). This allows parsers to read files that contain information not used by a particular tool.

OBO Document Structure

An OBO document is structured as follows:

```
<header>

<stanza>
<stanza>
...
```

Blank lines are ignored.

The header is an unlabeled section at the beginning of the document containing tag-value pairs. The header ends when the first stanza is encountered.

A stanza is a labeled section of the document, indicating that an object of a particular type is being described. Stanzas consist of a stanza name in square brackets, and a series of tag-value pairs, structured as follows:

```
[<Stanza name>]
<tag-value pair>
<tag-value pair>
<tag-value pair>
```

Comments

An OBO file may contain any number of lines beginning with !, at any point in the file. These lines are ignored by parsers.

Further, any line may end with a ! comment. Parsers that encounter an unescaped ! will ignore the ! and all data until the end of the line. \<newline> sequences are not allowed in ! comments (see escape characters).

Tag-Value Pairs

Tag-value pairs consist of a tag name, an unescaped colon, the tag value, and a newline:

```
<tag>: <value> {<trailing modifiers>} ! <comment>
```

The tag name is always a string. The value is always a string, but the value string may require special parsing depending on the tag with which it is associated.

In general, tag-value pairs occur on a single line. Multi-line values are possible using escape characters (see escape characters).

In general, each stanza type expects a particular set of pre-defined tags. However, a stanza may contain *any* tag. If a parser does not recognize a tag name for a particular stanza, no error will be generated. This allows new experimental tags to be added without breaking existing parsers. See handling unrecognized tags for specifics.

Trailing Modifiers

Any tag-value pair may be followed by a trailing modifier. Trailing modifiers have been introduced into the OBO 1.2 Specification to allow the graceful addition of new features to existing tags.

A trailing modifier has the following structure:

{<name>=<value>, <name=value>, <name=value>}

That is, trailing modifiers are lists of name-value pairs.

Parser implementations may choose to decode and/or round-trip these trailing modifiers. However, this is NOT required. A parser may choose to ignore or strip away trailing modifiers.

For this reason, trailing modifiers should only include information that is optional or experimental.

Trailing modifiers may also occur within dbxref definitions (see dbxref formmatting).

Escape characters

Tag names and values may contain the following escape characters:

\n	newline
\W	single space
\t	tab
\:	colon
\,	comma
\"	double quote

\\	backslash
\(	open parenthesis
\)	close parenthesis
\[	open bracket
\]	close bracket
{	open brace
}	close brace
\<newline>	<no value>

Escaped characters should only be used when a literal character is needed (that is, a character that the parser should not interpret as having a special meaning when parsing). Some tag values may contain unescaped colons, brackets, quotes, etc. that have meaning in decoding the tag value. Unescaped spaces between the seperator colon and the start of the value tag are discarded.

OBO Parser implementations may support only these escape characters, or they may assume that ANY character following a backslash is an escaped character. Parsers that choose the latter approach will translate \a and \? to "a" and "?" respectively.

Built-in OBO Semantics

Document Header Tags

Required tags:

format-version	Gives the obo specification version that this file
	uses. This is useful if tag semantics change from
	one obo specification version to the next.

Optional tags:

data-version	Gives the version of the current ontology.
version	Deprecated. Use data-version instead.
date	The current date in dd:MM:yyyy HH:mm format.
saved-by	The username of the person to last save this file.
	The meaning of "username" is entirely up to the
	application that generated the file.
auto-generated-by	The program that generated the file.

subsetdef	A description of a term subset. The value for this tag
	should contain a subset name, a space, and a quote
	enclosed subset description, as follows:
	subsetdef: GO_SLIM "GO Slim"
import	A url pointing to another obo document. The contents
	of the target document will be appended to this
	document at parse time. If the target document also
	contains import statements, they will be
	resolved. This tag replaces the "typeref" tag from
	earlier versions of the OBO spec.
synonymtypedef	A description of a user-defined synonym type. The
	value for this tag should contain a synonym type name,
	a space, a quote enclosed description, and an optional
	scope specifier, as follows:
	synonymtypedef: UK_SPELLING "British spelling"
	EXACT
	The scope specifier indicates the default scope for
	any synonym that has this type. See the synonym
	section of tags in a term stanza for more
	information on the scope specifier.
idspace	A mapping between a "local" ID space and a
	"global" ID space. The value for this tag should
	be a local idspace, a space, a URI, optionally
	followed by a quote-enclosed description, like this:
	idspace: GO urn:lsid:bioontology.org:GO: "gene ontology terms"
default-relationship-id	-prefix
	Any relationship lacking an ID space will be
	prefixed with the value of this tag. For example:
	default-relationship-id-prefix: OBO_REL

	The above will make sure that all relations
	refered to in the current file come from
	the OBO relations ontology, unless otherwise
	specified.
	The scope of this tag is within the current
	file only. See also id-mapping, below
id-mapping	maps a Term or Typedef ID to another Term or
	Typedef ID. The main reason for this Tag is to
	increase interoperability between different
	OBO ontologies.
	id-mapping: part_of OBO_REL:part_of
	This maps all cases of the unqualified
	relationship "part_of" to the ID
	OBO_REL:part_of defined in the OBO relations
	ontology
	The scope of this tag is within the current
	file only. Note that the
	default-relationship-id-prefix tag takes
	precedence over this tag
remark	General comments for this file. This tag is
	differentiated from a "!" comment in that the contents
	of a remark tag are guaranteed to be preserved by a
	parser.

Stanzas

At present, every OBO stanza always begins with an "id" tag. The value of the "id" tag announces the object to which the rest of the tags in the stanza refer. Normal, non-anonymous ids have global scope. An object has the same id in every file, and in every namespace.

The id tag may be optionally followed by an is_anonymous tag. If the value of is_anonymous is true, the object is anonymous. The id of an anonymous object is not fixed; if the ontology is parsed and then reserialized, the id may change. Anonymous ids have local scope; they are only valid in the file from which they were loaded. The same anonymous id in two different files refers to *a different object* in each file.

Any given stanza *does not* have to contain all the required tags. A file (or collection of files) may contain multiple stanzas that describe different aspects of an object. A "required" tag must be specified at least once for each object in a given set of files. This makes it possible for optional information to be stored in a seperate file, and only loaded when necessary.

This means that parsers must wait until the end of the parse batch to check whether required information is missing. Multiple descriptions may produce parse errors if:

1. A stanza contains tags that contradict a previous stanza (ie one term description gives a different term name than another description)

2. A parser has processed all the files in a batch, but an object is still missing some required value (such as a term name).

There are currently three supported stanza types: [Term], [Typedef], and [Instance]. Parsers/serializers will round-trip (successfully load and save) unrecognized stanzas.

Legal IDs and Special Identifiers

Any string is a legal id, as long as it is not one of the built in identifiers. Four of these are defined by the OBO spec:

obo:TYPE	Refers to any type
obo:TERM	Refers to any term
obo:TERM_OR_TYPE	Refers to any term or type
obo:INSTANCE	Refers to any instance

The others are primitive types specified by XML-Schema [http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html].

These are the only XML-Schema primitives supported by obo.

xsd:simpleType	Indicates any primitive type (abstract)
xsd:string	A string
xsd:integer	Any integer
xsd:decimal	Any real number
xsd:negativeInteger	Any negative integer
xsd:positiveInteger	Any integer > 0
xsd:nonNegativeInteger	Any integer >= 0
xsd:nonPositiveInteger	Any integer < 0
xsd:boolean	True or false
xsd:date	An XML-Schema date

These reserved identifiers are ONLY legal for domain and range constraints or as a datatype specifier in a

property_value statement. They may have special uses within applications that use OBO. See typedef tags and instance tags for more information.

Many tags require one or more object ids as their target values. It is up to the parser implementation to decide whether it is legal for a tag to reference an id for an object that has not been loaded. See dangling references for information on how to handle dangling references.

Tags in a [Term] Stanza

Required tags:

id	The unique id of the current term.
name	The term name. Any term may have only ONE name defined. If
	multiple term names are defined, it is a parse error.

Optional tags:

is_anonymous	Whether or not the current object has an anonymous id
alt_id	Defines an alternate id for this term. A term may have any
	number of alternate ids.
def	The definition of the current term. There must be zero or one
	instances of this tag per term description. More than one
	definition for a term generates a parse error. The value of
	this tag should be the quote enclosed definition text,
	followed by a dbxref list containing dbxrefs that describe the
	origin of this definition (see dbxref formatting
	for information on how dbxref lists are encoded). An example
	of this tag would look like this:
	definition: "The breakdown into simpler components of (+)-camphor, a bicyclic monoterpene ketone." [UM-BBD:pathway "", http://umbbd.ahc.umn.edu/cam/cam_map.html ""]
comment	A comment for this term. There must be zero or one instances
	of this tag per term description. More than one comment for a
	term generates a parse error.
subset	This tag indicates a term subset to which this term
	belongs. The value of this tag must be a subset name as
	defined in a subsetdef tag in the file header. If the value of

	this tag is not mentioned in a subsetdef tag, a parse error
	will be generated. A term may belong to any number of subsets.
synonym	This tag gives a synonym for this term, some xrefs to
	describe the origins of the synonym, and may indicate a
	synonym category or scope information.
	The value consists of a quote enclosed synonym
	text, an optional scope identifier, an optional synonym type
	name, and an optional dbxref list, like this:
	synonym: "The other white meat" EXACT MARKETING_SLOGAN [MEAT:00324, BACONBASE:03021]
	The synonym scope may be one of four values: EXACT, BROAD,
	NARROW, RELATED. If the first form is used to specify a
	synonym, the scope is assumed to be RELATED.
	The synonym type must be the id of a synonym type defined by a
	synonymtypedef line in the header. If the synonym type has a
	default scope, that scope is used regardless of any scope
	declaration given by a synonym tag.
	The dbxref list is formatted as specified in
	dbxref formatting. A term may have any number of
	synonyms.
exact_synonym	Deprecated. An alias for the synonym tag with the scope
	modifier set to EXACT.
narrow_synonym	Deprecated. An alias for the synonym tag with the scope
	modifier set to NARROW.
broad_synonym	Deprecated. An alias for the synonym tag with the scope
	modifier set to BROAD.
xref	A dbxref that describes an analagous term in another
	vocabulary (see dbxref formatting for information
	about how the value of this tag must be formatted). A term may
	have any number of xrefs.

xref_analog	Deprecated. An alias for the xref tag.
xref_unk	Deprecated. An alias for the xref tag.
is_a	This tag describes a subclassing relationship between one term
	and another. The value is the id of the term of which this
	term is a subclass. A term may have any number of is_a
	relationships.
	Parsers which support trailing modifiers may optionally parse
	the following trailing modifier tags for is_a:
	namespace <any namespace id>
	derived true OR false
	The "namespace" modifier allows the is_a relationship to be
	assigned it's own namespace (independant of the namespace of
	the superclass or subclass of this is_a relationships).
	The derived modifier indicates that the is_a relationship was
	not explicitly defined by a human ontology designer, but was
	created automatically by a reasoner, and could be re-derived
	using the non-derived relationships in the ontology.
	This tag previously supported the "completes" trailing
	modifier. This modifier is now deprecated. Use the
	intersection_of tag instead.
intersection_of	This tag indicates that this term represents the intersection
	of several other terms. The value is either a term id, or a
	relationship type id, a
	space, and a term id. For example:
	intersection_of: GO:00001
	intersection_of: part_of GO:00002
	This means that the term is a subclass of any term that is
	both a subclass of GO:00001 and has a part_of relationship to
	GO:00002.

	If any intersection_of tags are specified for a term, at least
	2 intersection_of tags need to be present or it is a parse
	error. The full intersection for the term is the set of ALL
	ids specified by ALL intersection_of tags for that term.
	This tag may not be applied to relationship types.
	Parsers which support trailing modifiers may optionally parse
	the following trailing modifier tag for disjoint_from:
	namespace <any namespace id>
union_of	This tag indicates that this term represents the union of
	several other terms. The value is the id of one of the other
	terms of which this term is a union.
	If any union_of tags are specified for a term, at least 2
	union_of tags need to be present or it is a parse error. The
	full union for the term is the set of ALL ids specified by ALL
	union_of tags for that term.
	This tag may not be applied to relationship types.
	Parsers which support trailing modifiers may optionally parse
	the following trailing modifier tag for disjoint_from:
	namespace <any namespace id>
disjoint_from	This tag indicates that a term is disjoint from another,
	meaning that the two terms have no instances or subclasses in
	common. The value is the id of the term from which the current
	term is disjoint. This tag may not be applied to relationship
	types.
	Parsers which support trailing modifiers may optionally parse
	the following trailing modifier tag for disjoint_from:
	namespace <any namespace id>
	derived true OR false

	The "namespace" modifier allows the disjoint_from relationship
	to be assigned it's own namespace.
	The derived modifier indicates that the disjoint_from
	relationship was not explicitly defined by a human ontology
	designer, but was created automatically by a reasoner, and
	could be re-derived using the non-derived relationships in the
	ontology.
relationship	This tag describes a typed relationship between this term and
	another term. The value of this tag should be the relationship
	type id, and then the id of the target term. The relationship
	type name must be a relationship type name as defined in a
	typedef tag stanza. The [typedef] must either occur in a
	document in the current parse batch, or in a file imported via
	an import header tag. If the relationship type name is
	undefined, a parse error will be generated. If the id of the
	target term cannot be resolved by the end of parsing the
	current batch of files, this tag describes a "dangling
	reference". See "Parser Requirements" for information about
	how a parser may handle dangling references. If a relationship
	is specified for a term with an "is_obsolete" value of true, a
	parse error will be generated. If a relationship target is a
	term which is obsolete, a parse error will be generated.
	Parsers which support trailing modifiers may optionally parse
	the following trailing modifier tags for relationships:
	not_necessary true OR false
	inverse_necessary true OR false
	namespace <any namespace id>
	derived true OR false
	cardinality any non-negative integer
	maxCardinality any non-negative integer
	minCardinality any non-negative integer
	The "necessary" modifier allows a relationship to be marked as
	"not necessarily true". The "inverse_necessary" modifier allows
	the inverse of a relationship to be marked "necessarily

	true".
	The "namespace" modifier allows the relationship to be
	assigned it's own namespace (independant of the namespace of
	the parent, child, or type of the relationship).
	The derived modifier indicates that the relationship was
	not explicitly defined by a human ontology designer, but was
	created automatically by a reasoner, and could be re-derived
	using the non-derived relationships in the ontology.
	The various cardinality constraints specify the number of
	relationships of a given type that may be defined for instances
	of this term.
	This tag previously supported the "completes" trailing
	modifier. This modifier is now deprecated. Use the
	intersection_of tag instead.
is_obsolete	Whether or not this term is obsolete. Allowable values are
	"true" and "false" (false is assumed if this tag is not
	present). Obsolete terms must have NO relationships, and no
	defined is_a, inverse_of, disjoint_from, union_of, or
	intersection_of tags.
replaced_by	Gives a term which replaces an obsolete term. The value is the
	id of the replacement term. The value of this tag can safely
	be used to automatically reassign instances whose instance_of
	property points to an obsolete term.
	The replaced_by tag may only be specified for obsolete terms,
	and may only be specified once per term. This tag cannot be
	used in conjunction with the consider tag.
consider	Gives a term which may be an appropriate substitute for an
	obsolete term, but needs to be looked at carefully by a human
	expert before the replacement is done.
	This tag may only be specified for obsolete terms. A single
	obsolete term may have many consider tags. This tag

	cannot be used in conjunction with replaced_by.
use_term	Deprecated. Equivalent to consider.
builtin	Whether or not this term or relation is builtin to
	the obo format. Allowable values are "true" and
	"false" (false assumed as default). Rarely used.
	One example of where this is used is the Obo
	relations ontology, which provides a stanza for the
	"is_a" relation, even though this relation is
	axiomatic to the language.

Dbxref Formatting

Dbxref definitions take the following form:

```
<dbxref name> {optional-trailing-modifier}
```

or

```
<dbxref name> "<dbxref description>" {optional-trailing-modifier}
```

By convention, the dbxref name is a colon separated key-value pair, but this is not a requirement. If provided, the dbxref description is a string of zero or more characters describing the dbxref. An example of a dbxref would be:

```
GO:ma "Sprung whole from the head of Michael, like Athena"
```

Dbxref lists are used when a tag value must contain several dbxrefs. Dbxref lists take the following form:

```
[<dbxref definition>, <dbxref definition>, ...]
```

The brackets may contain zero or more comma separated dbxref definitions. An example of a dbxref list would be:

```
[GO:ma, GO:midori "Midori was drinking and came up with this", GO:john {name
```

Note that the trailing modifiers (like all trailing modifiers) do not need to be decoded or round-tripped by parsers; trailing modifiers can always be optionally ignored. However, all parsers must be able to gracefully ignore trailing modifiers. It is important to recognize that lines which accept a dbxref list may have a trailing modifier for each dbxref in the list, and ANOTHER trailing modifier for the line itself.

Tags in [Typedef] Stanza:

[Typedef] stanzas support almost all the same tags as a [Term] stanza.

The following tags are not allowed in a [Typedef] stanza:

- union_of
- intersection_of
- disjoint_from

The following additional tags are *only* allowed in a Typedef stanza:

domain	The id of a term, or a special reserved identifier,
	which indicates the domain for this relationship type. If a property P has domain D, then any term T that has a relationship of type P to another term
	is a subclass of D. Note that this does NOT mean that the domain restricts which classes of terms
	can have a relationship of type P to another term.
	Rather, it means that any term that has a relationship of type P to another term is
	by definition a subclass of D.
range	The id of a term, or a special reserved identifier,
	which indicates acceptable range for this
	relationship type. If a property P has range R, then any term T that is the target of a relationship of type P is a subclass of R. Note that this does NOT mean that the range restricts which classes of terms can be the target of relationships of type P. Rather, it means that any term that is the target of a relationship of type P is *by definition* a subclass of R.
inverse_of	The id of another relationship type that is the
	inverse of this relationship type. If relation A is the
	inverse_of type B, and instance X has relationship A
	to instance Y, then it is implied that instance Y has
	relation B to instance X. Note that this
	applies at the instance level. If a particular
	relationship tag has a "true"
	trailing qualifier then the inverse applies at
	the term level.
transitive_over	The id of another relationship type that this
	relationship type is transitive over. If P is

	transitive_over Q, and the ontology has X P Y
	and Y Q Z then it follows that X P Z (term/type
.	level).
is_cyclic	Whether or not a cycle can be made from this
	relationship type. If a relationship type is
	non-cyclic, it is illegal for an ontology to contain a
	cycle made from user-defined OR implied relationships
	of this type. Allowed values: true or false
is_reflexive	Whether this relationship is reflexive. All reflexive
	relationships are also cyclic. Allowed values:
	true or false. Term/type level.
is_symmetric	Whether this relationship is symmetric. All symmetric
	relationships are also cyclic.
	Allowed values: true or false. Term/type level.
is_anti_symmetric	Whether this relationship is anti-symmetric.
	Allowed values: true or false. Term/type level.
is_transitive	Whether this relationship is transitive
	Allowed values: true or false. Term/type level.
is_metadata_tag	Whether this relationship is a metadata tag. Properties that are marked as metadata tags are used to record object metadata. Object metadata is additional information about an object that is useful to track, but does not impact the definition of the object or how it should be treated by a reasoner. Metadata tags might be used to record special term synonyms or structured notes about a term, for example.

Tags in an [Instance] Stanza

Required tags:

id	The unique id of the current term.
name	The instance name. Any instance may have only ONE name
	defined.

instance_of	The term id that gives the class of which this is an
	instance.

Optional tags

property_value	This tag binds a property to a value in this instance. The
	value of this tag is a relationship type id, a space, and a
	value specifier. The value specifier may have one of two
	forms; in the first form, it is just the id of some other
	instance, relationship type or term. In the second form, the
	value is given by a quoted string, a space, and datatype
	identifier. See Legal IDs and special identifiers
	for more information on legal datatype identifiers.
	[Instance]
	id: john
	name: John Day-Richter
	instance_of: boy
	property_value: married_to heather
	property_value: shoe_size "8" xsd:positiveInteger

The following optional tags are also allowable for instances. They have exactly the same syntax and semantics as defined in tags in a term stanza:

-is_anonymous -namespace -alt_id -comment -xref -synonym -is_obsolete -replaced_by -consider

The replaced_by and consider tags are also allowable for obsolete instances, but they must refer to another instance, rather than another term, to use as a replacement.

Built-In Objects

By default, every obo ontology contains the following objects:

```
[Typedef]
id: is_a
name: is_a
range: obo:TERM_OR_TYPE
domain: obo:TERM_OR_TYPE
definition: The basic subclassing relationship [obo:defs]

[Typedef]
id: disjoint_from
name: disjoint_from
range: obo:TERM
domain: obo:TERM
definition: Indicates that two classes are disjoint [obo:defs]
```

```
[Typedef]
id: instance_of
name: instance_of
range: obo:TERM
domain: obo:INSTANCE
definition: Indicates the type of an instance [obo:defs]

[Typedef]
id: inverse_of
name: inverse_of
range: obo:TYPE
domain: obo:TYPE
definition: Indicates that one relationship type is the inverse of another \
            [obo:defs]

[Typedef]
id: union_of
name: union_of
range: obo:TERM
domain: obo:TERM
definition: Indicates that a term is the union of several others [obo:defs]

[Typedef]
id: intersection_of
name: intersection_of
range: obo:TERM
domain: obo:TERM
definition: Indicates that a term is the intersection of several others \
            [obo:defs]
```

Parsers and Serializers

General Behavior

All parsers should be capable of failing gracefully and generating errors explaining the failure. Parsers may optionally be capable of generating warnings, if the file being read contains non-fatal errors.

Handling Unrecognized Tags

A parser may do one of three things when an unrecognized tag is found: - FAIL - Report a fatal error and terminate parsing - WARN - Report a warning, but continue parsing and ignore the unrecognized tag - WARN_AND_RECORD - Report a warning, but record the unrecognized tag for later serialization - IGNORE - Silently ignore the unrecognized tag - RECORD - Record the unrecognized tag for later serialization (RECOMMENDED)

Non-roundtripping Header Tags

The following optional header tags need not survive round-tripping:

- format-version

- version
- date
- saved-by
- auto-generated-by

The do not need to be round tripped, because the correct values will change when the file is saved.

Dangling references

There are several options when a dangling reference is encountered

- FAIL - Report a fatal error and terminate parsing
- WARN_AND_IGNORE - Report a fatal error and ignore the dangling reference
- WARN_AND_READ - Report a warning and read in the dangling reference, storing it in a form suitable for round-tripping
- READ - Silently read and store the dangling relationship (recommended)

Serializer Conventions

Any parser should be able to read correctly formatted files in any layout. However, it is suggested that serializers obey the following conventions to ensure consistency.

General conventions

- Within a single file, all tags relating to a single entity should appear in the same stanza (thereby minimizing the total number of stanzas and keeping all tags regarding a single entity in the same place)
- In any case where the correct ordering of tags is ambiguous (for example, if there are two tags with the same name, or the ordering is not given in this document), tags should be ordered alphabetically, first on the tag name, then on the tag value.

Stanza conventions

All new stanza declarations should be preceded by a blank line. [Typedef] stanzas should appear before [Term] stanzas, and [Instance] stanzas should appear after [Term] stanzas. All other stanza types should appear after [Instance] stanzas, in alphabetical order on the stanza name.

Header tags:

Header tags should appear in the following order: format-version, data-version, date, saved-by,

auto-generated-by, import, subsetdef, synonymtypedef, default-namespace, remark.

Ordering Term & Typedef stanzas

Term, Typdef, and Instance stanzas should be serialized in alphabetical order on the value of their id tag.

Ordering Term & Typedef tags

Term tags should appear in the following order: *id* is_anonymous *name* namespace *alt_id* def *comment* subset *synonym* xref *is_a* intersection_of *union_of* disjoint_from *relationship* is_obsolete *replaced_by* consider

Typedef tags should appear in the following order: *id* is_anonymous *name* namespace *alt_id* def *comment* subset *synonym* xref *domain* range *is_anti_symmetric* is_cyclic *is_reflexive* is_symmetric *is_transitive* is_a *inverse_of* transitive_over *relationship* is_obsolete *replaced_by* consider

Instance tags should appear in the following order: *id* is_anonymous *name* namespace *alt_id* comment *synonym* xref *instance_of* property_value *is_obsolete* replaced_by * consider

If the same tag appears multiple times in a stanza, the tags should be ordered alphabetically on the tag value.

Dbxref lists

Values in dbxref lists should be ordered alphabetically on the dbxref name.

Changes in version 1.2

May 2nd, 2006 revision

- Added is_metadata_tag tag for properties
- Revised bad definitions of range and domain from old, closed-world definitions to correct open-world definitions

March 14th, 2006 revision

- added builtin
- clarified meaning of relation property tags (is_transitive, is_symmetric, is_anti_symmetric, is_reflexive) - these tags apply at the type level

February 9th, 2006 revision

- Added transitive_over

- Added is_anti_symmetric
- Specified allowed values (true/false) for typedef boolean tags

November 9th, 2005 revision

- Fixed out-of-date "last revised" date in this document
- Added id-mapping tag
- Added default-relationship-id-prefix tag
- Added idspace tag
- Fixed some mistakes in tag examples

June 30, 2005 revision

- Deprecated xref_analog tag. All xrefs are now of the same type.

June 23, 2005 revision

- Changed name of UNSPECIFIED scope modifier to RELATED

June 6, 2005 revision

- Changed name of propertyValue tag to property_value for consistency.
- Fixed some omissions and typos in section Ordering Term & Typedef tags

February 2, 2005 revision

- Added namespace as an allowable instance tag
- Changed name of xref_unk tag to xref. Deprecated xref_unk

January 31, 2005 revision

- Changed property_value tag syntax with quoted values. Now a primitive datatype identifier is required

following the quoted value. The datatype identifier gives the datatype of the quoted value. (see special identifiers and instance tags)

- Removed xsd:duration and xsd:number from primitive type listing (see special identifiers)
- Added section numbers for revision information
- Added About this file section

Initial 1.2 release

- Nested trailing modifier values are no longer allowed. Trailing modifiers are now a list of simple name value pairs. (see [S.1.4]).
- Added synonymtypedef header tag (see [S.2.1]).
- Deprecated typeref tag. Use import instead (see [S.2.1])
- Added reserved identifiers for specifying non-term range and domain values (see [S.2.2.1], [S.2.2.4])
- Deprecated exact_synonym, narrow_synonym, broad_synonym. Use synonym tag instead (see [S.2.2.2]).
- Added additional parsing options for synonyms (see [S.2.2.2]).
- Deprecated use_term tag. Use "consider" instead (see [S.2.2.2]).
- The "completes" trailing modifier for isa is now deprecated. Use intersection_of instead. (see [S.2.2.2])
- Added intersection_of, union_of, inverse_of, and disjoint_from tags. (see [S.2.2.2] and [S.2.2.4])
- Added "derived" trailing modifier to is_a, inverse_of, disjoint_from and relationship. (see [S.2.2.2] and [S.2.2.4])
- Added instances (see [S.2.2.5])