

# RDF Facets v2.1

## Bringing the Semantic Web to Traditional Web Search Engines

May 27<sup>th</sup>, 2008

Patrick Stickler  
patrick.stickler@nokia.com

### Overview

RDF uses URIs. URIs tend to cause problems with most token keyword based search engines due to special characters which are allowed in URIs but typically ignored/discarded/disallowed in keywords by most web search engines. And unlike simple token keywords, RDF knowledge is expressed as structured property+value pairs, therefore there must be some means of expressing property+value pairs as single, unified string keywords escaped in a manner acceptable to a broad range of web search engines.

Furthermore, RDF provides for arbitrary datatypes, and so reliable/meaningful comparisons of values can be difficult in practice, since while we want a fairly flexible and future proof metadata framework for expressing knowledge, most search engines support only a very basic set of actual datatypes.

A solution that is flexible and scalable, supporting the evolution, extension, and adoption of various vocabularies, must be able to deal with arbitrary URIs, property+value pairs, datatyped values, and language qualifications without major retooling of the underlying infrastructure every time vocabularies and ontologies change. Ideally, a system would be automatically configured and updated based on knowledge defined in RDF itself.

The following is a description of "RDF Facets", an intermediate, simplified, and normalized representation of (a subset of) RDF modeled knowledge which should be suitable for efficient use with most token keyword based web search engines (as well as more capable engines supporting a broader range of basic datatypes such as numbers and dates).

### A Little Bit About RDF

It's not necessary to go into all the gory details of RDF to understand how to encode and use RDF Facets. The essential bit to understand is that RDF is all about statements describing things.

An RDF statement has three parts:

1. The thing being described (the "subject").
2. The characteristic/property being attributed to that thing (the "predicate").
3. The realization/specification/value of that property (the "object").

So an RDF statement is simply

subject + predicate + object

RDF statements are also called "triples", due to their three-part structure.

Subjects and predicates are (for our purposes) denoted by URIs. Objects can be URIs, simple strings, language qualified strings, XML content, or datatyped values. There is also a facility for abbreviating URIs

---

as qualified names, using XML Namespace prefixes, providing for a much more highly compressed encoding.

E.g. if we have a ringtone identified by the URI `http://example.org/coolRingtone` and the ringtone had a title "A very cool ringtone" defined using the Dublin Core (DC) vocabulary title property, then the RDF statement (triple) would look like

```
<http://example.org/coolRingtone> <http://purl.org/dc/elements/1.1/title> "A very cool ringtone" .
```

Because datatypes are essential for reliable and meaningful comparison of values, we need to provide for a select few core, essential value types; namely strings, numbers, dates and times, XML fragments, and of course URIs. It's not practical to try to support any arbitrary datatypes, so any datatyped values which cannot be coerced into one of these core types are simply treated as "literal" raw values, where only equality can be tested.

Ideally, the mappings from various datatypes to the RDF Facet core types would be defined in RDF, such that an agent which converts RDF graphs to RDF Facets would utilize that knowledge in determining which mappings were appropriate for each particular property value. This mapping information could also, of course, be hard coded in the agent, but would of course require modifying the code to support additional datatypes.

(For those familiar with RDF, note that blank nodes are not (yet) supported in RDF Facets, and hence there is no representation of system identifiers)

In addition to the basic statement (triple) structure of RDF, it is also possible to qualify statements via additional statements about statements. This is called reification. In Forum Nokia, we use reification to assign a relevance percentage to topic classifications (e.g. a given resource may be 30% relevant to Java and 80% relevant to MMS).

A reification is a set of statements, which specify the subject, predicate, and object of another statement, along with additional statements qualifying the reified statement (see example below).

RDF Facets are a means to encode the predicate+object portion of an RDF statement as an alphanumeric string without loss of the boundary between the property and its value and with a fundamental degree of datatype support.

A description of a URI denoted resource, including statement reifications (but excluding statements with blank node objects) can then be distilled down into a set of normalized facets based on the core set of value types, which are all that any given search engine need support.

## RDF Index Facet Syntax

The syntax for RDF Index Facets is:

```
DESCRIPTION      = (SUBJECT WHITESPACE+)? INDEX_FACET+
INDEX_FACET      = PREDICATE [ OBJECT REIFICATION* ] WHITESPACE+
SUBJECT          = "<" URI ">"
PREDICATE        = ( FACET_NAME | FACET_URI )
OBJECT           = ( U_OBJECT | N_OBJECT | S_OBJECT | B_OBJECT | X_OBJECT | D_OBJECT | L_OBJECT )
U_OBJECT         = "U" ( FACET_NAME | FACET_URI )
N_OBJECT         = "N" LEXICAL_FORM
S_OBJECT         = "S" LEXICAL_FORM ( "L" LANG )?
B_OBJECT         = "B" LEXICAL_FORM
X_OBJECT         = "X" LEXICAL_FORM
D_OBJECT         = "D" LEXICAL_FORM
L_OBJECT         = "L" DATATYPE_URI LEXICAL_FORM
REIFICATION      = "R" PREDICATE OBJECT
FACET_NAME       = "q" <Facet encoded XML Namespace prefix> "z" <Facet encoded URI localname> "Q"
FACET_URI        = "x" <Facet encoded URI> "X"
LEXICAL_FORM     = "x" <Facet encoded lexical representation of the property value> "X"
DATATYPE_URI    = ( FACET_NAME | FACET_URI )
URI              = <non-escaped, literal URI>
LANG             = <Facet encoded ISO conformant language code, with optional dialect, all lowercase>
WHITESPACE      = <whitespace character>
```

Where the following datatype mappings apply:

```
"U" <-> URI          (any valid URI)
```

```

"S" <-> String      (any valid Unicode string)
"N" <-> Number      (any valid lexical form of xsd:decimal)
"D" <-> Date(Time)  (any valid lexical form of xsd:date or xsd:dateTime)
"B" <-> Boolean     (any valid lexical form of xsd:boolean)
"X" <-> XML        (any valid lexical form of rdfs:XMLLiteral)
"L" <-> Literal Value (any valid Unicode string)

```

Sequences of more than one reification are ordered alphabetically first by the URI of the predicate and then, for the same predicate, by the lexical form of the object.

## Facet Encoding

Facet encoding employs the following reserved characters:

```
D E L q Q R x X Z
```

Any character in the input which is either a reserved character, or not an alphanumerical character (a-zA-Z0-9) is escaped as

```
"E" <two digit hexadecimal value>
```

The resulting encoding string consists entirely of alphanumeric characters, and hence provides an optimal token for indexing and search.

For example, the URI "http://sw.nokia.com/MARS-3/title" is facet escaped to become

```
httpE3aE2fE2fswE2enokiaE2ecomE2fMAE52SE2d3E2ftitle
```

Similarly, the string "A very cool ringtone" becomes

```
AE20veryE20coolE20ringtone
```

Thus, the above RDF statement about the ringtone would be represented as an RDF Facet as

```
xhttpE3aE2fE2fswE2enokiaE2ecomE2fMAE52SE2d3E2ftitleXSxAE20veryE20coolE20ringtoneX
```

or in a more compressed format, utilizing XML namespaces to produce an abbreviated, or "qualified", name for the property, as

```
qmarsZtitleQSxAE20veryE20coolE20ringtoneX
```

In the index, the property+value (predicate+object) portion can be treated as a single keyword or "facet" associated with the target (subject). Since the encoding used is fully bidirectional, one can later obtain the actual URIs, lexical forms, etc. as required for comparisons or for e.g. mapping the facets back to RDF/XML (though converting from qualified names back to URIs requires knowledge of the XML Namespace prefix mappings used to abbreviate those URIs and can only be done reliably in a closed system or between systems with consistently shared prefix mappings).

Any web search engine which can accept strings comprised of 7-bit alphanumeric characters as index keywords will be able to accept RDF Facets as index keywords (though some search engines may not support arbitrarily long keywords, which could preclude the use of overly long facets with some search engines). Thus the keyword

```
qmarsZtitleQSxAE20veryE20coolE20ringtoneX
```

representing the property+value pair for a title "A very cool ringtone" can be associated with the resource in the same way as a simple token keyword occurring in some indexed textual content, and comparison is performed using a simple string equality test the same as for any keyword.

Furthermore, the length of the RDF derived keyword should not impose a substantial efficiency penalty for most search engines, as most employ hashing and similar methods to optimize keyword storage and comparison, thus even if thousands of resources would be qualified with a given RDF property+value facet, the complete facet string will typically be stored only once.

**Note:** the use of qualified names in facets introduces a risk, albeit very slight, for naming collisions where the same XML Namespace prefix is mapped by different systems from different URI bases, which can introduce ambiguity in query execution. In normal practice, however, this should not be a problem.

The following is an example of a facet encoding of a statement with reification for a resource classified for the topic 'java' with 30% relevance, i.e.

```
<http://example.org/someResource> <http://sw.nokia.com/FN-1/topic> <http://sw.nokia.com/FN-1/topic/java> .
_:s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement> .
_:s <http://www.w3.org/1999/02/22-rdf-syntax-ns#subject> <http://example.org/someResource> .
_:s <http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate> <http://sw.nokia.com/FN-1/topic> .
_:s <http://www.w3.org/1999/02/22-rdf-syntax-ns#object> <http://sw.nokia.com/FN-1/topic/java> .
_:s <http://sw.nokia.com/MARS-3/relevance> "30"^^<http://sw.nokia.com/MARS-3/Percentage> .
```

where the datatype `<http://sw.nokia.com/MARS-3/Percentage>` maps to a number ("N") value type, would be represented as an RDF Facet Description as

```
<http://example.org/someResource>
qfnZtopicQUqfnTopicZjavaQRqmarsZrelevanceQNx30X
```

## Expansion of Facets

Prior to indexing, all reified and language qualified facets should be expanded into more generalized facets which omit the reification(s) or language qualifications, to facilitate more efficient identification of resources with the specified facet where reification or language qualification is not relevant. In addition, all facets with predicate and object defined should be expanded to predicate only facets, indicating that some value for that predicate was defined (used with the ANY and NO query operators detailed below). Expanded facets should not be duplicated. I.e. if multiple reified, language qualified, and predicate/object facets generalize to the same more basic facet, that facet should only be specified and indexed once. E.g.:

```
<http://example.org/someResource>
qfnZtopicQUqfnTopicZjavaQRqmarsZrelevanceQNx30X
qmarsZtitleQSxAE20veryE20coolE20ringtoneXLen
```

expands to

```
<http://example.org/someResource>
qfnZtopicQUqfnTopicZjavaQRqmarsZrelevanceQNx30X
qfnZtopicQUqfnTopicZjavaQ
qfnZtopicQ
qmarsZtitleQSxAE20veryE20coolE20ringtoneXLen
qmarsZtitleQSxAE20veryE20coolE20ringtoneX
qmarsZtitleQ
```

## RDF Query Facet Syntax

When it comes time to execute an RDF-based query, we can construct a variant of the index facets which can be efficiently compared. These are intended to be compared against the database of index facets (or some internal representation specific to the particular search engine), with the goal being the identification of the subjects having the matched index facets.

The syntax for RDF Query Facets is:

```
QUERY          = QUERY_FACET ( WHITESPACE+ QUERY_FACET ) *
QUERY_FACET    = QUALIFIED_VALUE REIFICATION *
QUALIFIED_VALUE = ( "ANY" | "NO" ) PREDICATE
QUALIFIED_VALUE = U_OPERATOR? PREDICATE U_OBJECT
QUALIFIED_VALUE = N_OPERATOR? PREDICATE N_OBJECT
QUALIFIED_VALUE = S_OPERATOR? PREDICATE S_OBJECT
QUALIFIED_VALUE = B_OPERATOR? PREDICATE B_OBJECT
QUALIFIED_VALUE = X_OPERATOR? PREDICATE X_OBJECT
QUALIFIED_VALUE = D_OPERATOR? PREDICATE D_OBJECT
QUALIFIED_VALUE = L_OPERATOR? PREDICATE L_OBJECT
PREDICATE      = ( FACET_NAME | FACET_URI )
U_OBJECT       = "U" ( FACET_NAME | FACET_URI )
N_OBJECT       = "N" LEXICAL_FORM
S_OBJECT       = "S" LEXICAL_FORM ( "L" LANG ) ?
```

```

B_OBJECT      = "B" LEXICAL_FORM
X_OBJECT      = "X" LEXICAL_FORM
D_OBJECT      = "D" LEXICAL_FORM
D_OBJECT      = MILESTONE
L_OBJECT      = "L" DATATYPE_URI LEXICAL_FORM
U_OPERATOR    = ( "NE" )
N_OPERATOR    = ( "NE" | "LT" | "GT" | "LE" | "GE" | "NLT" | "NGT" | "NLE" | "NGE" )
S_OPERATOR    = ( "NE" | "LT" | "GT" | "LE" | "GE" | "NLT" | "NGT" | "NLE" | "NGE" | "CN" | "NCN" )
E_OPERATOR    = ( "NE" )
X_OPERATOR    = ( "NE" )
D_OPERATOR    = ( "NE" | "LT" | "GT" | "LE" | "GE" | "NLT" | "NGT" | "NLE" | "NGE" )
L_OPERATOR    = ( "NE" )
MILESTONE    = ( "HOUR" | "DAY" | "WEEK" | "MONTH" | "YEAR" )
REIFICATION   = "R" QUALIFIED_VALUE
FACET_NAME    = "q" <Facet encoded XML Namespace prefix> "Z" <Facet encoded URI localname> "Q"
FACET_URI     = "x" <Facet encoded URI> "X"
LEXICAL_FORM  = "x" <a Facet encoded lexical representation of value> "X"
DATATYPE_URI  = ( FACET_NAME | FACET_URI )
URI           = <non-escaped, literal URI>
LANG          = <Facet encoded ISO conformant language code, with optional dialect, all lowercase>
WHITESPACE    = <a whitespace character>

```

Where the following operator interpretations apply:

"ANY"	resource has any (some) value defined for property
"NO"	resource has no value defined for property
"NE"	not equal to
"LT"	less than
"LE"	less than or equal to
"GT"	greater than
"GE"	greater than or equal to
"NLT"	not less than
"NLE"	not less than or equal to
"NGT"	not greater than
"NGE"	not greater than or equal to
"CN"	contains
"NCN"	does not contain

And where the following milestone interpretations apply:

"HOUR"	date-time equal to one hour ago from present
"DAY"	date-time equal to one day ago from present
"WEEK"	date-time equal to one week ago from present
"MONTH"	date-time equal to one month ago from present
"YEAR"	date-time equal to one year ago from present

And where the following datatype mappings apply:

"U"	any valid URI
"S"	any valid Unicode string
"N"	any valid lexical form of xsd:decimal (a basic number)
"D"	any valid lexical form of xsd:date or xsd:dateTime
"B"	any valid lexical form of xsd:boolean
"X"	any valid lexical form of rdfs:XMLLiteral
"L"	any valid Unicode string (an RDF literal value)

Sequences of more than one reification **MUST** be ordered alphabetically first by the URI of the predicate (prior to any mapping to a qualified name) then, for the same predicate, by the lexical form of the object (again, prior to any mapping to a qualified name, if the value is a URI).

Note that several of the operators have logical equivalents which could be expressed in another fashion using another operator; e.g. GT = NLE. These "aliases" are provided so that a user's perspective can be captured and users are not forced to rethink their query in terms of some other logical equivalent. It's easy enough for the system executing queries to map the aliases to a subset of minimal local operators.

The absence of any operator at the beginning of the query facet indicates a test for equality. Note that this equality query facet form is equivalent to the predicate+object portion of an index facet, allowing for comparison using a simple string equality test.

Furthermore, one can easily differentiate equality query facets from other query facets by checking the first character of the query facet. If it is "x" or "q" then the query facet is a test for equality, otherwise, it involves some other value comparison operation.

A simple search engine could support only equality tests, and still benefit greatly from employing this methodology to obtain string encoded keyword facets derived from RDF property+value pairs.

Note that this facet based query model, is "flat" and resource centric, reflecting the general nature of typical web search engines. As such, it does not provide for structured values (where the value is a blank node with

additional properties) nor does it provide for complex queries where more than one resource is described by the query (e.g. find all documents who's author's first name is "Bob").

It also defines a query as a simple set of query facets, and it is not specified how query facets might be otherwise combined in structured boolean expressions or whether all or some of a set of query facets must match (i.e. complete vs. best match). Different search engines will provide different levels of functionality in this regard. This document aims to nail down the common and most fundamental bits which should be relevant to most web search engines.

The following is a query facet corresponding to the query "match all targets with a Forum Nokia publication date later than (greater than) 2004-01-01":

```
GTqfnZpublishedQDx2004E2d01E2d01X
```

which would match the index facet description

```
<http://example.org/myRingtone>  
qfnZpublishedQDx2004E2d03E2d31X
```

by ultimately being evaluated within the search engine as a comparison of the dates, e.g.

```
? datetime("2004-03-31") > datetime("2004-01-01")
```

and thereby identifying the target <http://example.org/myRingtone>, etc.

An example query facet which identifies all resources relating to the Forum Nokia topic 'java' with a relevance greater than or equal to 20 would be:

```
qfnZtopicQUqfnTopicZjavaQRGEqmarsZrelevanceQNx20X
```

A query facet which identifies all resources having any (some) Forum Nokia type definition (i.e. all Forum Nokia resources) would be:

```
ANYqfnZtypeQ
```

A conjunctive query (where all query facets must match) which identifies all resources having any Forum Nokia type definition but having no Forum Nokia topic definition would be:

```
ANYqfnZtypeQ  
NoqfnZtopicQ
```

## Query Analysis of User Input Keywords: "Keyword to Facet Mapping"

Free-form, user input, natural language based keywords can be associated with particular facets, as a means to map from user input to precise metadata classifications, as well as a means to map from precise queries to federated search engine interfaces which have no RDF or RDF Facet support.

E.g., we could associate the input keywords "games" and "gaming" with the facet

```
qfnZtopicQUqfnTopicZgamesQ
```

corresponding to the property+value pair for the Forum Nokia 'topic' property and the 'games' topic value. When a user inputs into a basic search field either keyword "games" or "gaming", rather than searching only for those words in the textual content of resources, we also interpret the query to mean they are looking for resources relating to the precise topic 'games', adding the more precise facet, thereby providing more precise and useful search results.

This mapping between keywords and metadata facets is fully bidirectional.

I.e., a query broker could use this mapping to generate keyword-based searches to external search engines from query facets. Thus, the above query facet could be mapped to a Google search sub-pattern 'games OR gaming', joining all keywords associated with the facet disjunctively.

One of the greatest benefits to this approach is that it allows most of the knowledge to be maintained in a declarative fashion, and also simplifies the query broker's mapping task to one strictly concerned with syntax (how to express conjunctive keyword searches) -- which reduces the overall implementation complexity and also should significantly enhance runtime efficiency.

## Inference and Run-Time Search Index Generation

When generating/updating the run-time search index based on the RDF index facets, inference/reasoning should be employed to produce a maximal facet set for any given resource. Thus, if we know, based on our ontologies and their defined relationships, that e.g. any resource classified for topic "X" also pertains to topic "Y", we will generate the explicit facet for both topics "X" and "Y" as well.

When someone then searches for resources pertaining to topic "Y", they will find the above mentioned resource, even though in the knowledge management side, it is only explicitly classified for topic "X", because the run-time search engine will maintain a maximal view of that resources characteristics, both explicitly and implicit.

This approach allows the indexer/search engine to be as simple as possible (not having to provide support for keyword aliases, taxonomies, etc.) and thus also be as efficient as possible (the two tend to go hand in hand) since the bulk of the complexity is confined to the indexing process and not to run-time execution of queries.

And since this inference/reasoning is done during the index generation process, the ontological complexity does not directly impact the run-time efficiency of the run-time search service (the cost being increased storage and run-time memory requirements to accommodate the expanded index).

---