W3C Editor's Draft

rdf:PlainLiteral: A Datatype for RDF Plain Literals

W3C Editor's Draft 31 May 2009

This version:
http://www.w3.org/2007/OWL/draft/ED-rdf-text-20090531/
Latest editor's draft:
http://www.w3.org/2007/OWL/draft/rdf-text/
Previous version:
http://www.w3.org/2007/OWL/draft/ED-rdf-text-20090528/ (color-coded diff)
Authors:
Jie Bao, Rensselaer Polytechnic Institute, Troy, New York, USA
Axel Polleres, DERI Galway at the National University of Ireland, Galway, Ireland
Boris Motik, Oxford University, Oxford, UK

This document is also available in these non-normative formats: PDF version.

Abstract

This document presents the specification of a primitive datatype for the plain literals of RDF.

Status of this Document

May Be Superseded

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

W3C Editor's Draft

Summary of Changes

The name of the datatype was changed from "rdf:text" to "rdf:PlainLiteral" to emphasize the relationship to RDF plain literals. Additionally, the introduction and section 4 were re-written to further clarify this, and the names of the builtins were changed to match.

Please Comment By 19 July 2009

The OWL Working Group and the Rule Interchange Format (RIF) Working Group seek public feedback on this Working Draft. Please send your comments to public-owl-comments@w3.org (public archive). If possible, please offer specific changes to the text that would address your concern. You may also wish to check the Wiki Version of this document and see if the relevant text has already been updated.

No Endorsement

Publication as a Working Draft does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

Patents

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

1 Introduction

The Resource Description Framework [*RDF*] is defined to have an extensible system of typed literals, based on XML Schema datatypes [*XSD*], and also to have plain literals. In the RDF specification, plain literals differ from typed literals in that plain literals have no datatype and can optionally have a language tag, indicating the natural language of the content. (See Tags for Identifying Languages [*BCP 47*]). This branching approach to the design for RDF literals complicates specifications which connect to RDF, such as RIF and OWL. Furthermore, RDF does not provide a name for the set of all plain literals, which, for example, prevents one from stating in OWL that the range of some property must be a plain literal with a language tag.

To address these deficiencies, this specification introduces a datatype called rdf:PlainLiteral. The datatype is in the "rdf:" namespace because it refers to parts of the conceptual model of RDF. This extension, however, does not change that conceptual model, and thus does not affect specifications that depend on it such as SPARQL [*SPARQL*]. The value space of rdf:PlainLiteral consists of all data values assigned to RDF plain literals, which allows RDF applications to explicitly refer to this set (e.g., in rdfs:range assertions).

Because RDF plain literals are already a part of RDF and SPARQL syntaxes, rdf:PlainLiteral literals are written as RDF plain literals in RDF and SPARQL syntaxes.

Editor's Note: In previous drafts of this specification and the RIF and OWL 2 specifications, this datatype was called "rdf:text". The name was changed to emphasize the correspondence between this datatype and RDF plain literals. The syntactic form of the name parallels rdf:XMLLiteral.

2 Preliminaries

A *character* is an atomic unit of text. Each character has a Universal Character Set (UCS) code point [*ISO/IEC 10646*] (or, equivalently, a Unicode code point [*UNICODE*]) that *must* match the Char production from XML [*XML*] thus ensuring compatibility with XML Schema Datatypes, version 1.1 [*XML Schema Datatypes*]. Code points are sometimes represented in this document as U+ followed by a four-digit hexadecimal value of the code point.

A *string* is a finite sequence of zero or more characters. The *length* of a string is the number of characters in it. Strings are written in this specification by enclosing them in double quotes. Two strings are identical if and only if they contain exactly the same characters in exactly the same sequence.

Example:

UCS [*ISO/IEC 10646*] and Unicode [*UNICODE*] provide for 1,114,112 different code points. The Char production from XML [*XML*], however, excludes the surrogate code points and the code points U+FFFE and U+FFFF. Thus, `rdf:PlainLiteral` provides a total of 1,112,033 different characters. This number is important, as it can affect the satisfiability of an OWL 2 ontology. Consider the following example:

```
ClassAssertion( a:i MinCardinality( n a:property
DatatypeRestriction( xs:string xs:length 1 ) ) )
```

This OWL 2 axiom states that the individual `a:i` is connected by the property `a:property` to at least *n* different strings of length one. The number of such strings is limited to 1,112,033 by the above definitions, so this ontology is satisfiable if and only if *n* is smaller than or equal to 1,112,033.

A *language tag* is a string matching the `langtag` production from BCP 47 [*BCP 47*]. Furthermore, note that this definition corresponds to the *well-formed* rather than the *valid* class of conformance in BCP 47. A language tag *may* contain subtags that are not registered in the IANA Language Subtag Registry, although an `rdf:PlainLiteral` implementation *may* also choose to reject such invalid language tags.

Example:

The language tag `"en-fubar"` is not registered with the IANA Language Subtag Registry, so an `rdf:PlainLiteral` implementation is allowed to reject it. This string, however, matches the `langtag` production from BCP 47, so it is a perfectly valid language tag for the purpose of this specification. Consequently, the value space of `rdf:PlainLiteral` (see Section 3 for its definition) contains, say, the pair `( "some string" , "en-fubar" )`.

This specification uses Uniform Resource Identifiers (URIs) for naming datatypes and their components, which are defined in RFC 3986 [*RFC 3986*]. For readability, URIs prefixes are often abbreviated by a short prefix name according to the convention of RDF [*RDF*]. The following prefix names are used throughout this document:

- the prefix name `xs:` stands for `http://www.w3.org/2001/XMLSchema#`
- the prefix name `rdf:` stands for `http://www.w3.org/1999/02/22-rdf-syntax-ns#`

The names of the built-in functions defined in Section 5 are QNames, as defined in the XML namespaces specification [*XML Namespaces*]. The following namespace abbreviations are used in Section 5:

- `fn` stands for the `http://www.w3.org/2005/xpath-functions` namespace
- `rtfn` stands for the `http://www.w3.org/2009/rdf-text-functions` namespace

Whether an expression of the form `pr:ln` denotes an abbreviated URI or a QName should be clear from the context: only the names of the built-in functions in Section 5 are QNames; all other such expressions denote abbreviated URIs.

Datatypes are defined in this document along the lines of XML Schema Datatypes [*XML Schema Datatypes*]. Each datatype is identified by a URI and is described by the following components:

- The *value space* is a set determining the set of values of the datatype. Elements of the value space are called *data values*.
- The *lexical space* is a set of strings that can be used to refer to data values. Each member of the lexical space is called a *lexical form*, and it is mapped to a particular data value.
- The *facet space* is a set of pairs of the form `( F v )`, where `F` is a URI called a *constraining facet*, and `v` is an arbitrary data value called a *constraining value*. Each such pair is mapped to a subset of the value space of the datatype.

A *plain literal* is a string with an optional language tag [*RDF*]. A plain literal without a language tag is interpreted in an RDF interpretation by itself. A plain literal with a language tag can be written as `"abc"@langTag`, and is interpreted in an RDF interpretation as a pair `( "abc" , "langTag" )`.

A *typed literal* consists of a string and a datatype URI [*RDF*] and can be written as `"abc"^^datatypeURI`. Given an RDF datatype identified by `datatypeURI`, an RDF datatyped-interpretation that includes the datatype interprets the typed literal as the data value that the datatype assigns to the lexical form `"abc"`.

The italicized keywords *must*, *must not*, *should*, *should not*, and *may* specify certain aspects of the normative behavior of tools implementing this specification, and are interpreted as specified in RFC 2119 [*RFC 2119*].

3 Definition of the `rdf:PlainLiteral` Datatype

The datatype identified by the URI `http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral` (abbreviated `rdf:PlainLiteral`) is defined as follows.

Value Space. The value space of `rdf:PlainLiteral` consists of

- all strings, and
- all pairs of the form `( "abc" , "lc-langtag" )` where `"abc"` is a string and `"lc-langtag"` is a lowercase language tag.

Lexical Space. An `rdf:PlainLiteral` lexical form is a string of the form `"abc@langTag"` where `"abc"` is an arbitrary (possibly empty) string, and `"langTag"` is either the empty string or a (not necessarily lowercase) language tag. Each such lexical form is mapped to a data value `dv` as follows:

- If `"langTag"` is empty, then `dv` is equal to the string `"abc"` and
- If `"langTag"` is not empty, then `dv` is equal to the pair `( "abc", "lc-langtag" )` where `"lc-langtag"` is `"langTag"` normalized to lowercase.

Example:

The following table shows several `rdf:PlainLiteral` lexical forms and their corresponding data values.

Lexical form	Corresponding data value
`"Family Guy@en"`	`( "Family Guy" , "en" )`
`"Family Guy@EN"`	`( "Family Guy" , "en" )`
`"Family Guy@FOX@en"`	`( "Family Guy@FOX" , "en" )`
`"Family Guy@"`	`"Family Guy"`
`"Family Guy@FOX@"`	`"Family Guy@FOX"`

The following table shows several of strings that are not `rdf:PlainLiteral` lexical forms.

String	The reason for not being an `rdf:PlainLiteral` lexical form
`"Family Guy"`	does not contain at least one `@` (U+0040) character
`"Family Guy@12"`	`"12"` is not a language tag according to BCP 47

Facet Space. The facet space of `rdf:PlainLiteral` is defined as shown in Table 1.

Table 1. The Facet Space of `rdf:PlainLiteral`

A pair `( F v )` is in the facet space of `rdf:PlainLiteral` if...	Each such pair is mapped to the subset of the value space of `rdf:PlainLiteral` containing...
...`F` is `xs:length`, `xs:minLength`, `xs:maxLength`, `xs:pattern`, `xs:enumeration`, or `xs:assertions` and `( F v )` is in the facet space of `xs:string`.	...all strings of the form `"abc"` and all pairs of the form `( "abc" , "lc-langtag" )` such that `"abc"` is contained in the subset of `xs:string` determined by `( F v )` as specified by XML Schema Datatypes [*XML Schema Datatypes*].
...`F` is `rdf:langRange` and `v` is an *extended language range* as specified in Section 2.2 of [*RFC4647*].	...all pairs of the form `( "abc" , "lc-langtag" )` such that `"lc-langtag"` matches `v` under *extended filtering* as specified in Section 3.3.2 of [*RFC4647*].

Example:

The facet `xs:length` can be used to refer to a subset of strings of a particular length regardless of whether they have a language tag or not. Thus, the subset of the value space of `rdf:PlainLiteral` corresponding to the pair `( xs:length 3 )` contains the string `"abc"`, as well as the pairs `( "abc" , "en" )` and `( "abc" , "de" )`.

Example:

The facet `rdf:langRange` can be used to refer to a subset of strings containing the language tag. Note that the language range need not be in lowercase, and that the matching algorithm is case-insensitive. Thus, the subset of the value space of `rdf:PlainLiteral` corresponding to the pair `( rdf:langRange "de-DE" )` contains the pairs `( "abc" , "de-de" )` and `( "abc" , "de-de-1996" )` (because these match the language range `"de-DE"` according to RFC 4647), but not the string `"abc"` (because it is not a pair with a language tag) or the pairs `( "abc" , "de-deva" )` and `( "abc" , "de-latn-de" )` (because these do not match the language range `"de-DE"` according to RFC 4647).

Example:

The pair `( rdf:langRange "*" )` is mapped to the subset of the value space of `rdf:PlainLiteral` containing all pairs of the form `( "abc" , "lc-langtag" )`. In languages such as OWL 2, this can be used to specify that a data value must contain the language tag.

4 Syntax for rdf:PlainLiteral Literals

It is obvious from the above that in datatyped interpretations that include the rdf:PlainLiteral datatype the value space of rdf:PlainLiteral contains exactly all data values assigned to plain literals (with or without a language tag). The rdf:PlainLiteral datatype thus provides an explicit way of referring to this set.

To eliminate another source of syntactic redundancy and to retain a large degree of interoperability with application that do not understand the rdf:PlainLiteral datatype, the form of rdf:PlainLiteral literals in syntaxes for RDF graphs and for SPARQL is the already existing syntax for the corresponding plain literal, not the syntax for a typed literal. Therefore, typed literals with rdf:PlainLiteral as the datatype do not occur in syntaxes for RDF graphs, nor in syntaxes for SPARQL.

To implement this design and provide this interoperability, applications that employ this datatype *must* use plain literals (instead of rdf:PlainLiteral typed literals) whenever a syntax for plain literals is provided, such as in existing syntaxes for RDF graphs and SPARQL results.

5 Functions on `rdf:PlainLiteral` Data Values

Editor's Note: These functions have not yet been renamed. The word "text" in the function names may change to "plain-literal", and the namespace may change from .../rdf-text-functions to .../plain-literal-functions

This section defines functions that construct and operate on `rdf:PlainLiteral` data values. The terminology used and the way in which these functions are described are in accordance with the XQuery 1.0 and XPath 2.0 Functions and Operators [*XPathFunc*]. Each function is identified by a QName [*XML Namespaces*]. The error codes used in this section are given in Appendix G of the XPath 2.0 specification [*XPath20*] and Appendix C of XQuery and XPath function specification [*XPathFunc*].

5.1 Functions for Assembling and Disassembling `rdf:PlainLiteral` Data Values

5.1.1 `rtfn:text-from-string`

```
rtfn:text-from-string( $arg1 as xs:string ) as rdf:PlainLiteral
rtfn:text-from-string( $arg1 as xs:string, $arg2 as xs:string) as rdf:Pla
```

Summary: returns the data value `( $arg1, lowercase($arg2) )` if `$arg2` is present, and returns the data value `$arg1` otherwise. Both arguments must be of type `xs:string` or one of its subtypes, and `$arg2` — if present — must be a (nonempty) language tag; otherwise, this function raises type error err:FORG0006. Note that, since the lexical forms of `rdf:PlainLiteral` require language tags to be in lowercase, this function converts `$arg2` to lowercase.

5.1.2 `rtfn:string-from-text`

```
rtfn:string-from-text( $arg as rdf:PlainLiteral) as xs:string
```

Summary: returns the string part `s` from the argument `$arg`, which must be an `rdf:PlainLiteral` data value of the form `( s, l )` or of the form `s`. If `$arg` is not of type `rdf:PlainLiteral`, this function raises type error err:FORG0006.

5.1.3 `rtfn:lang-from-text`

```
rtfn:lang-from-text( $arg as rdf:PlainLiteral ) as xs:lang
```

Summary: returns the language tag `l` if `$arg` is an `rdf:PlainLiteral` data value of the form `( s, l )`, and returns the empty string if `$arg` is an `rdf:PlainLiteral` data value of the form `s`. If `$arg` is not of type `rdf:PlainLiteral`, this function raises type error err:FORG0006.

5.2 The Comparison of `rdf:PlainLiteral` Data Values

The notion of collations used in this section is taken from Section 7.3.1 of XPath and XQuery function specification [*XPathFunc*].

5.2.1 `rtfn:compare`

```
rtfn:compare( $comparand1  as rdf:PlainLiteral?, $comparand2 as rdf:Plai
```

```
rtfn:compare( $comparand1  as rdf:PlainLiteral?, $comparand2 as rdf:Plai
```

Summary: if either `$comparand1` or `$comparand2` is not of type `rdf:PlainLiteral`, of if `$collation` is specified but is not of type `xs:string`, this function raises type error err:FORG0006. Otherwise, the function returns the empty sequence if one of the arguments is empty, if one of `$comparand1` and `$comparand2` has a language tag and the other one does not, or if the language parts of `$comparand1` and `$comparand2` are unequal; otherwise, this function returns -1, 0, or 1 depending on whether the value of the string-part of `$comparand1` (or `$comparand1` itself, respectively, if it has no language tag) is respectively less than, equal to, or greater than the value of the string-part of `$comparand2` (or `$comparand2` itself, respectively, if it has no language tag). The collation used by the invocation of this function is determined according to the rules in Section 7.3.1 of the XPath and XQuery functions specification [*XPathFunc*].

The first version of this function backs up the XQuery operators `"eq"`, `"ne"`, `"gt"`, `"lt"`, `"le"`, and `"ge"` on `rdf:PlainLiteral` values.

Feature At Risk #1: `rtfn:compare`

The final version of this specification might not include `rtfn:compare`, or it might contain an alternative solution: since `xs:string` values are `rdf:PlainLiteral` data values, the `fn:compare` function from XPath/ XQuery might be extended to cover `rdf:PlainLiteral` values.

Please send feedback to public-owl-comments@w3.org.

The two functions may be viewed as declared XQuery functions with the following definitions:

```
declare function  rtfn:compare( $comparand1 as rdf:PlainLiteral?, $compar
 {
  return
    if ( fn:compare (  rtfn:lang-from-text( $comparand1 ),   rtfn:lang-fr
       fn:compare (  rtfn:string-from-text( $comparand1 ) ,  rtfn:string-
 }
```

```
declare function  rtfn:compare( $comparand1  as rdf:PlainLiteral?, $compa
 {
  return
    if ( fn:compare ( fn:lang-from-text( $comparand1 ),   rtfn:lang-from-
       fn:compare (  rtfn:string-from-text( $comparand1 ) ,  rtfn:string-
 }
```

5.3 Other Functions on `rdf:PlainLiteral` Data Values

5.3.1 `rtfn:length`

```
rtfn:length($arg as rdf:PlainLiteral) as xs:integer
```

Summary: returns the number of characters in the string part `s` if `$arg` is an `rdf:PlainLiteral` data value of the form `( s, l )` or a string value `s`, respectively. If `$arg` is not of type `rdf:PlainLiteral`, this function raises type error err:FORG0006.

Feature At Risk #2: `rtfn:length`

The final version of this specification might not include `rtfn:length`, or it might contain an alternative solution: since `xs:string` values are `rdf:PlainLiteral` data values, the `fn:string-length` function from XPath/XQuery might be extended towards coverage of `rdf:PlainLiteral` values.

Please send feedback to public-owl-comments@w3.org.

This function may be viewed as a declared XQuery function with the following definition:

```
declare function  rtfn:text-length($arg as rdf:PlainLiteral?) as xs:integ
 {
  return
     fn:string-length (  rtfn:string-from-text( $arg ) )
 }
```

5.3.2 `rtfn:matches-language-range`

```
 rtfn:matches-language-range($arg as rdf:PlainLiteral?, $range as xs:stri
```

Summary: This function is only defined if `$arg` is a sequence of length 0 or 1 of literals of type `rdf:PlainLiteral` and `$range` is of type `xs:string`; if the parameters do not satisfy these typing conditions, the function raises a type error err:FORG0006. If the typing conditions are fulfilled, the function returns `true` in case `$arg` is an `rdf:PlainLiteral` data value of the form `( s, l )` with `l` a language tag that matches the extended language range `$range` as specified by the extended filtering algorithm for "Matching of Language Tags" [*BCP-47*]; otherwise, it returns `false`. This means that the function returns `false` if the argument is a string `rdf:PlainLiteral` data value. An empty input sequence is treated as a `rdf:PlainLiteral` data value consisting of the empty string, and accordingly on such input this function also returns `false`.

6 Acknowledgments

The RIF WG and the OWL WG made parallel efforts to support strings written in different languages. This specification is the outcome of a collaboration between the two groups, and it is based on the work on the `rif:text` datatype on the RIF side and the `owl:internationalizedString` datatype on the OWL side. A short description of the design process is available here.

7 References

[RFC 2119]
RFC 2119: Key words for use in RFCs to Indicate Requirement Levels. Network Working Group, S. Bradner. Internet Best Current Practice, March 1997.

[RFC 3986]
RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax. T. Berners-Lee, R. Fielding, and L. Masinter, IETF, January 2005.

[RFC 4647]
RFC 4647 - Matching of Language Tags. A. Phillips and M. Davis, IETF, September 2006.

[UNICODE]
The Unicode Standard. Unicode The Unicode Consortium, Version 5.1.0, ISBN 0-321-48091-0, as updated from time to time by the publication of new versions. (See http://www.unicode.org/unicode/standard/versions for the latest version and additional information on versions of the standard and of the Unicode Character Database)."

[ISO/IEC 10646]
ISO/IEC 10646-1:2000. Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane and ISO/IEC 10646-2:2001. Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 2: Supplementary Planes, as, from time to time, amended, replaced by a new edition or expanded by the addition of new parts. [Geneva]: International Organization for Standardization. ISO (International Organization for Standardization).

[BCP 47]
BCP-47 - Tags for Identifying Languages. A. Phillips, M. Davis, eds., IETF, September 2006, http://www.rfc-editor.org/rfc/bcp/bcp47.txt.

[RDF]
Resource Description Framework (RDF): Concepts and Abstract Syntax. Graham Klyne, Jeremy J. Carroll, and Brian McBride, eds., W3C Recommendation 10 February 2004.

[RDF Semantics]
RDF Semantics. Patrick Hayes, ed., W3C Recommendation 2004

[SPARQL]
SPARQL Query Language for RDF. Eric Prud'hommeaux, Andy Seaborne, eds., W3C Recommendation 15 January 2008.

[XML]
Extensible Markup Language (XML) 1.0 (Fifth Edition). Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau, eds., W3C Recommendation 26 November 2008.

[XML Namespaces]
Namespaces in XML 1.0 (Second Edition). Tim Bray, Dave Hollander, Andrew Layman, Richard Tobin, eds., W3C Recommendation 16 August 2006.

[XML Schema Datatypes]
W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. D. Peterson, S. Gao, A. Malhotra, C. M. Sperberg-McQueen, H. S. Thompson, eds., W3C Working Draft 30 January 2009.

[XPath20]
XML Path Language (XPath) 2.0. Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernández, Michael Kay, Jonathan Robie, and Jérôme Siméon, eds. W3C Recommendation 23 January 2007.

[XPathFunc]
XQuery 1.0 and XPath 2.0 Functions and Operators. Ashok Malhotra, Jim Melton, and Norman Walsh, eds. W3C Recommendation 23 January 2007.