W3C Recommendation

RDF 1.1 Turtle

Terse RDF Triple Language

W3C Recommendation 25 February 2014

This version:
http://www.w3.org/TR/2014/REC-turtle-20140225/
Latest published version:
http://www.w3.org/TR/turtle/
Test suite:
http://www.w3.org/TR/2014/NOTE-rdf11-testcases-20140225/
Implementation report:
http://www.w3.org/2013/TurtleReports/index.html
Previous version:
http://www.w3.org/TR/2014/PR-turtle-20140225/
Editors:
Eric Prud'hommeaux, W3C
Gavin Carothers, Lex Machina, Inc
Authors:
David Beckett
Tim Berners-Lee, W3C
Eric Prud'hommeaux, W3C
Gavin Carothers, Lex Machina, Inc

Please check the **errata** for any errors or issues reported since publication.

The English version of this specification is the only normative version. Non-normative translations may also be available.

Abstract

The Resource Description Framework (RDF) is a general-purpose language for representing information in the Web.

This document defines a textual syntax for RDF called Turtle that allows an RDF graph to be completely written in a compact and natural text form, with abbreviations for common usage patterns and datatypes. Turtle provides levels of compatibility with the N-Triples [N-TRIPLES] format as well as the triple pattern syntax of the SPARQL W3C Recommendation.

Status of This Document

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/.

This document is a part of the RDF 1.1 document suite. The document defines Turtle, the Terse RDF Triple Language, a concrete syntax for RDF [RDF11-CONCEPTS].

This document was published by the RDF Working Group as a Recommendation. If you wish to make comments regarding this document, please send them to public-rdf-comments@w3.org (subscribe, archives). All comments are welcome.

Please see the Working Group's implementation report.

This document has been reviewed by W3C Members, by software developers, and by other W3C groups and interested parties, and is endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

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

This section is non-normative.

This document defines Turtle, the Terse RDF Triple Language, a concrete syntax for RDF [RDF11-CONCEPTS].

A Turtle document is a textual representations of an RDF graph. The following Turtle document describes the relationship between Green Goblin and Spiderman.

EXAMPLE 1

```
@base <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rel: <http://www.perceive.net/schemas/relationship/> .

<#green-goblin>
    rel:enemyOf <#spiderman> ;
    a foaf:Person ;    # in the context of the Marvel universe
    foaf:name "Green Goblin" .

<#spiderman>
    rel:enemyOf <#green-goblin> ;
    a foaf:Person ;
    foaf:name "Spiderman", "Человек-паук"@ru .
```

This example introduces many of features of the Turtle language: @base and Relative IRIs, @prefix and prefixed names, predicate lists separated by ';', object lists separated by ',', the token `a`, and literals.

The Turtle grammar for `triples` is a subset of the SPARQL 1.1 Query Language [SPARQL11-QUERY] grammar for `TriplesBlock`. The two grammars share production and terminal names where possible.

The construction of an RDF graph from a Turtle document is defined in Turtle Grammar and Parsing.

2. Turtle Language

This section is non-normative.

A Turtle document allows writing down an RDF graph in a compact textual form. An RDF graph is made up of triples consisting of a subject, predicate and object.

Comments may be given after a '#' that is not part of another lexical token and continue to the end of the line.

2.1 Simple Triples

The simplest triple statement is a sequence of (subject, predicate, object) terms, separated by whitespace and terminated by '.' after each triple.

EXAMPLE 2

```
<http://example.org/#spiderman> <http://www.perceive.net/schemas/relationship/enemyOf> <http://example.org/#green-goblin>
```

2.2 Predicate Lists

Often the same subject will be referenced by a number of predicates. The predicateObjectList production matches a series of predicates and objects, separated by ';', following a subject. This expresses a series of RDF Triples with that subject and each predicate and object allocated to one triple. Thus, the ';' symbol is used to repeat the subject of triples that vary only in predicate and object RDF terms.

These two examples are equivalent ways of writing the triples about Spiderman.

EXAMPLE 3

```
<http://example.org/#spiderman> <http://www.perceive.net/schemas/relationship/enemyOf> <http://example.org/#green-goblin>
                                <http://xmlns.com/foaf/0.1/name> "Spiderman" .
```

EXAMPLE 4

```
<http://example.org/#spiderman> <http://www.perceive.net/schemas/relationship/enemyOf> <http://example.org/#green-goblin>
<http://example.org/#spiderman> <http://xmlns.com/foaf/0.1/name> "Spiderman" .
```

2.3 Object Lists

As with predicates often objects are repeated with the same subject and predicate. The objectList production matches a series of objects separated by ',' following a predicate. This expresses a series of RDF Triples with the corresponding subject and predicate and each object allocated to one triple. Thus, the ',' symbol is used to repeat the subject and predicate of triples that only differ in the object RDF term.

These two examples are equivalent ways of writing Spiderman's name in two languages.

EXAMPLE 5

```
<http://example.org/#spiderman> <http://xmlns.com/foaf/0.1/name> "Spiderman", "Человек-паук"@ru .
```

EXAMPLE 6

```
<http://example.org/#spiderman> <http://xmlns.com/foaf/0.1/name> "Spiderman" .
<http://example.org/#spiderman> <http://xmlns.com/foaf/0.1/name> "Человек-паук"@ru .
```

There are three types of *RDF Term* defined in RDF Concepts: IRIs (Internationalized Resource Identifiers), literals and blank nodes. Turtle provides a number of ways of writing each.

2.4 IRIs

IRIs may be written as relative or absolute IRIs or prefixed names. Relative and absolute IRIs are enclosed in '<' and '>' and may contain numeric escape sequences (described below). For example `<http://example.org/#green-goblin>`.

Relative IRIs like `<#green-goblin>` are resolved relative to the current base IRI. A new base IRI can be defined using the '`@base`' or '`BASE`' directive. Specifics of this operation are defined in section 6.3 IRI References

The token '`a`' in the predicate position of a Turtle triple represents the IRI `http://www.w3.org/1999/02/22-rdf-syntax-ns#type` .

A *prefixed name* is a prefix label and a local part, separated by a colon ":". A prefixed name is turned into an IRI by concatenating the IRI associated with the prefix and the local part. The '`@prefix`' or '`PREFIX`' directive associates a prefix label with an IRI. Subsequent '`@prefix`' or '`PREFIX`' directives may re-map the same prefix label.

NOTE

The Turtle language originally permitted only the syntax including the '`@`' character for writing prefix and base directives. The case-insensitive '`PREFIX`' and '`BASE`' forms were added to align Turtle's syntax with that of SPARQL. It is advisable to serialize RDF using the '`@prefix`' and '`@base`' forms until RDF 1.1 Turtle parsers are widely deployed.

To write `http://www.perceive.net/schemas/relationship/enemyOf` using a prefixed name:

1. Define a prefix label for the vocabulary IRI `http://www.perceive.net/schemas/relationship/` as `somePrefix`
2. Then write `somePrefix:enemyOf` which is equivalent to writing `<http://www.perceive.net/schemas/relationship/enemyOf>`

This can be written using either the original Turtle syntax for prefix declarations:

EXAMPLE 7

```
@prefix somePrefix: <http://www.perceive.net/schemas/relationship/> .
```

```
<http://example.org/#green-goblin> somePrefix:enemyOf <http://example.org/#spiderman> .
```

or SPARQL's syntax for prefix declarations:

EXAMPLE 8

```
PREFIX somePrefix: <http://www.perceive.net/schemas/relationship/>

<http://example.org/#green-goblin> somePrefix:enemyOf <http://example.org/#spiderman> .
```

NOTE

Prefixed names are a superset of XML QNames. They differ in that the local part of prefixed names may include:

- leading digits, e.g. `leg:3032571` or `isbn13:9780136019701`
- non leading colons, e.g. `og:video:height`
- reserved character escape sequences, e.g. `wgs:lat\-long`

The following Turtle document contains examples of all the different ways of writing IRIs in Turtle.

EXAMPLE 9

```
# A triple with all absolute IRIs
<http://one.example/subject1> <http://one.example/predicate1> <http://one.example/object1> .

@base <http://one.example/> .
<subject2> <predicate2> <object2> .     # relative IRIs, e.g. http://one.example/subject2

BASE <http://one.example/>
<subject2> <predicate2> <object2> .     # relative IRIs, e.g. http://one.example/subject2

@prefix p: <http://two.example/> .
p:subject3 p:predicate3 p:object3 .     # prefixed name, e.g. http://two.example/subject3

PREFIX p: <http://two.example/>
p:subject3 p:predicate3 p:object3 .     # prefixed name, e.g. http://two.example/subject3

@prefix p: <path/> .                    # prefix p: now stands for http://one.example/path/
p:subject4 p:predicate4 p:object4 .     # prefixed name, e.g. http://one.example/path/subject4

@prefix : <http://another.example/> .    # empty prefix
:subject5 :predicate5 :object5 .        # prefixed name, e.g. http://another.example/subject5

:subject6 a :subject7 .                 # same as :subject6 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> :subject7 .

<http://伝言.example/?user=أكرم&amp;channel=R%26D> a :subject8 . # a multi-script subject IRI .
```

NOTE

The '`@prefix`' and '`@base`' directives require a trailing '.' after the IRI, the equalivent '`PREFIX`' and '`BASE`' must not have a trailing '.' after the IRI part of the directive.

2.5 RDF Literals

Literals are used to identify values such as strings, numbers, dates.

EXAMPLE 10

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<http://example.org/#green-goblin> foaf:name "Green Goblin" .

<http://example.org/#spiderman> foaf:name "Spiderman" .
```

2.5.1 Quoted Literals

Quoted Literals (Grammar production RDFLiteral) have a lexical form followed by a language tag, a datatype IRI, or neither. The representation of the lexical form consists of an initial delimiter, e.g. `"` (U+0022), a sequence of permitted characters or numeric escape sequence or string escape sequence, and a final delimiter. The corresponding RDF lexical form is the characters between the delimiters, after processing any escape sequences. If present, the language tag is preceded by a '`@`' (U+0040). If there is no language tag, there may be a datatype IRI, preceeded by '`^^`' (U+005E U+005E). The datatype IRI in Turtle may be written using either an absolute IRI, a relative IRI, or prefixed name. If there is no datatype IRI and no language tag, the datatype is `xsd:string`.

'`\`' (U+005C) may not appear in any quoted literal except as part of an escape sequence. Other restrictions depend on the delimiter:

- Literals delimited by `'` (U+0027), may not contain the characters `'`, `LF` (U+000A), or `CR` (U+000D).
- Literals delimited by `"`, may not contain the characters `"`, `LF`, or `CR`.
- Literals delimited by `'''` may not contain the sequence of characters `'''`.

- Literals delimited by `"""` may not contain the sequence of characters `"""`.

EXAMPLE 11

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix show: <http://example.org/vocab/show/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

show:218 rdfs:label "That Seventies Show"^^xsd:string .            # literal with XML Schema string datatype
show:218 rdfs:label "That Seventies Show"^^<http://www.w3.org/2001/XMLSchema#string> . # same as above
show:218 rdfs:label "That Seventies Show" .                                 # same again
show:218 show:localName "That Seventies Show"@en .                 # literal with a language tag
show:218 show:localName 'Cette Série des Années Soixante-dix'@fr . # literal delimited by single quote
show:218 show:localName "Cette Série des Années Septante"@fr-be .  # literal with a region subtag
show:218 show:blurb '''This is a multi-line                        # literal with embedded new lines and quotes
literal with many quotes (""""")
and up to two sequential apostrophes ('').''' .
```

2.5.2 Numbers

Numbers can be written like other literals with lexical form and datatype (e.g. `"-5.0"^^xsd:decimal`). Turtle has a shorthand syntax for writing integer values, arbitrary precision decimal values, and double precision floating point values.

Data Type	Abbreviated	Lexical	Description
xsd:integer	`-5`	`"-5"^^xsd:integer`	Integer values may be written as an optional sign and a series of digits. Integers match the regular expression "`[+-]?[0-9]+`".
xsd:decimal	`-5.0`	`"-5.0"^^xsd:decimal`	Arbitrary-precision decimals may be written as an optional sign, zero or more digits, a decimal point and one or more digits. Decimals match the regular expression "`[+-]?[0-9]*\.[0-9]+`".
xsd:double	`4.2E9`	`"4.2E9"^^xsd:double`	Double-precision floating point values may be written as an optionally signed mantissa with an optional decimal point, the letter "e" or "E", and an optionally signed integer exponent. The exponent matches the regular expression "`[+-]?[0-9]+`" and the mantissa one of these regular expressions: "`[+-]?[0-9]+\.[0-9]+`", "`[+-]?\.[0-9]+`" or "`[+-]?[0-9]`".

EXAMPLE 12

```
@prefix : <http://example.org/elements> .
<http://en.wikipedia.org/wiki/Helium>
    :atomicNumber 2 ;               # xsd:integer
    :atomicMass 4.002602 ;          # xsd:decimal
    :specificGravity 1.663E-4 .     # xsd:double
```

2.5.3 Booleans

Boolean values may be written as either '`true`' or '`false`' (case-sensitive) and represent RDF literals with the datatype xsd:boolean.

EXAMPLE 13

```
@prefix : <http://example.org/stats> .
<http://somecountry.example/census2007>
    :isLandlocked false .           # xsd:boolean
```

2.6 RDF Blank Nodes

RDF blank nodes in Turtle are expressed as `_:` followed by a blank node label which is a series of name characters. The characters in the label are built upon PN_CHARS_BASE, liberalized as follows:

- The characters `_` and digits may appear anywhere in a blank node label.
- The character `.` may appear anywhere except the first or last character.
- The characters `-`, `U+00B7`, `U+0300` to `U+036F` and `U+203F` to `U+2040` are permitted anywhere except the first character.

A fresh RDF blank node is allocated for each unique blank node label in a document. Repeated use of the same blank node label identifies the same RDF blank node.

EXAMPLE 14

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:alice foaf:knows _:bob .
_:bob foaf:knows _:alice .
```

2.7 Nesting Unlabeled Blank Nodes in Turtle

In Turtle, fresh RDF blank nodes are also allocated when matching the production blankNodePropertyList and the terminal ANON. Both of these may appear in the subject or object position of a triple (see the Turtle Grammar). That subject or object is a fresh RDF blank node. This blank node also serves as the subject of the triples produced by matching the predicateObjectList production embedded in a blankNodePropertyList. The generation of these triples is described in Predicate Lists. Blank nodes are also allocated for collections described below.

EXAMPLE 15

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

# Someone knows someone else, who has the name "Bob".
[] foaf:knows [ foaf:name "Bob" ] .
```

The Turtle grammar allows blankNodePropertyLists to be nested. In this case, each inner `[` establishes a new subject blank node which reverts to the outer node at the `]`, and serves as the current subject for predicate object lists.

The use of predicateObjectList within a blankNodePropertyList is a common idiom for representing a series of properties of a node.

Abbreviated:

EXAMPLE 16

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

[ foaf:name "Alice" ] foaf:knows [
    foaf:name "Bob" ;
    foaf:knows [
        foaf:name "Eve" ] ;
    foaf:mbox <bob@example.com> ] .
```

Corresponding simple triples:

EXAMPLE 17

```
_:a <http://xmlns.com/foaf/0.1/name> "Alice" .
_:a <http://xmlns.com/foaf/0.1/knows> _:b .
_:b <http://xmlns.com/foaf/0.1/name> "Bob" .
_:b <http://xmlns.com/foaf/0.1/knows> _:c .
_:c <http://xmlns.com/foaf/0.1/name> "Eve" .
_:b <http://xmlns.com/foaf/0.1/mbox> <bob@example.com> .
```

2.8 Collections

RDF provides a Collection [RDF11-MT] structure for lists of RDF nodes. The Turtle syntax for Collections is a possibly empty list of RDF terms enclosed by `()`. This collection represents an `rdf:first`/`rdf:rest` list structure with the sequence of objects of the `rdf:first` statements being the order of the terms enclosed by `()`.

The `(...)` syntax MUST appear in the subject or object position of a triple (see the Turtle Grammar). The blank node at the head of the list is the subject or object of the containing triple.

EXAMPLE 18

```
@prefix : <http://example.org/foo> .
# the object of this triple is the RDF collection blank node
:subject :predicate ( :a :b :c ) .

# an empty collection value - rdf:nil
:subject :predicate2 () .
```

3. Examples

This section is non-normative.

This example is a Turtle translation of example 7 in the RDF/XML Syntax specification (example1.ttl):

EXAMPLE 19

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ex: <http://example.org/stuff/1.0/> .

<http://www.w3.org/TR/rdf-syntax-grammar>
  dc:title "RDF/XML Syntax Specification (Revised)" ;
  ex:editor [
    ex:fullname "Dave Beckett";
    ex:homePage <http://purl.org/net/dajobe/>
  ] .
```

An example of an RDF collection of two literals.

EXAMPLE 20

```
PREFIX : <http://example.org/stuff/1.0/>
:a :b ( "apple" "banana" ) .
```

which is short for (example2.ttl):

EXAMPLE 21

```
@prefix : <http://example.org/stuff/1.0/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
:a :b
  [ rdf:first "apple";
    rdf:rest [ rdf:first "banana";
               rdf:rest rdf:nil ]
  ] .
```

An example of two identical triples containing literal objects containing newlines, written in plain and long literal forms. The line breaks in this example are LINE FEED characters (U+000A). (example3.ttl):

EXAMPLE 22

```
@prefix : <http://example.org/stuff/1.0/> .

:a :b "The first line\nThe second line\n  more" .

:a :b """The first line
The second line
  more""" .
```

As indicated by the grammar, a collection can be either a subject or an object. This subject or object will be the novel blank node for the first object, if the collection has one or more objects, or `rdf:nil` if the collection is empty.

For example,

EXAMPLE 23

```
@prefix : <http://example.org/stuff/1.0/> .
(1 2.0 3E1) :p "w" .
```

is syntactic sugar for (noting that the blank nodes `b0`, `b1` and `b2` do not occur anywhere else in the RDF graph):

EXAMPLE 24

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
    _:b0  rdf:first  1 ;
          rdf:rest   _:b1 .
    _:b1  rdf:first  2.0 ;
          rdf:rest   _:b2 .
    _:b2  rdf:first  3E1 ;
          rdf:rest   rdf:nil .
    _:b0  :p         "w" .
```

RDF collections can be nested and can involve other syntactic forms:

EXAMPLE 25

```
PREFIX : <http://example.org/stuff/1.0/>
(1 [:p :q] ( 2 ) ) :p2 :q2 .
```

is syntactic sugar for:

EXAMPLE 26

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
    _:b0  rdf:first  1 ;
          rdf:rest   _:b1 .
    _:b1  rdf:first  _:b2 .
    _:b2  :p         :q .
    _:b1  rdf:rest   _:b3 .
    _:b3  rdf:first  _:b4 .
    _:b4  rdf:first  2 ;
          rdf:rest   rdf:nil .
    _:b3  rdf:rest   rdf:nil .
```

4. Turtle compared to SPARQL

This section is non-normative.

The SPARQL 1.1 Query LanguageF (SPARQL) [SPARQL11-QUERY] uses a Turtle style syntax for its TriplesBlock production. This production differs from the Turtle language in that:

1. SPARQL permits RDF Literals as the subject of RDF triples.
2. SPARQL permits variables (`?`*name* or `$`*name*) in any part of the triple of the form.
3. Turtle allows prefix and base declarations anywhere outside of a triple. In SPARQL, they are only allowed in the Prologue (at the start of the SPARQL query).
4. SPARQL uses case insensitive keywords, except for '`a`'. Turtle's `@prefix` and `@base` declarations are case sensitive, the SPARQL dervied `PREFIX` and `BASE` are case insensitive.
5. '`true`' and '`false`' are case insensitive in SPARQL and case sensitive in Turtle. `TrUe` is not a valid boolean value in Turtle.

For further information see the Syntax for IRIs and SPARQL Grammar sections of the SPARQL query document [SPARQL11-QUERY].

5. Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words MUST, MUST NOT, REQUIRED, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this specification are to be interpreted as described in [RFC2119].

This specification defines conformance criteria for:

- Turtle documents
- Turtle parsers

A conforming **Turtle document** is a Unicode string that conforms to the grammar and additional constraints defined in section 6. Turtle Grammar, starting with the `turtleDoc` production. A Turtle document serializes an RDF Graph.

A conforming **Turtle parser** is a system capable of reading Turtle documents on behalf of an application. It makes the serialized RDF dataset, as defined in section 7. Parsing, available to the application, usually through some form of API.

The IRI that identifies the Turtle language is: `http://www.w3.org/ns/formats/Turtle`

> NOTE
>
> This specification does not define how Turtle parsers handle non-conforming input documents.

5.1 Media Type and Content Encoding

The media type of Turtle is `text/turtle`. The content encoding of Turtle content is always UTF-8. Charset parameters on the mime type are required until such time as the `text/` media type tree permits UTF-8 to be sent without a charset parameter. See section B. Internet Media Type, File Extension and Macintosh File Type for the media type registration form.

6. Turtle Grammar

A Turtle document is a Unicode[UNICODE] character string encoded in UTF-8. Unicode characters only in the range U+0000 to U+10FFFF inclusive are allowed.

6.1 White Space

White space (production WS) is used to separate two terminals which would otherwise be (mis-)recognized as one terminal. Rule names below in capitals indicate where white space is significant; these form a possible choice of terminals for constructing a Turtle parser.

White space is significant in the production String.

6.2 Comments

Comments in Turtle take the form of '#', outside an IRIREF or String, and continue to the end of line (marked by characters U+000D or U+000A) or end of file if there is no end of line after the comment marker. Comments are treated as white space.

6.3 IRI References

Relative IRIs are resolved with base IRIs as per Uniform Resource Identifier (URI): Generic Syntax [RFC3986] using only the basic algorithm in section 5.2. Neither Syntax-Based Normalization nor Scheme-Based Normalization (described in sections 6.2.2 and 6.2.3 of RFC3986) are performed. Characters additionally allowed in IRI references are treated in the same way that unreserved characters are treated in URI references, per section 6.5 of Internationalized Resource Identifiers (IRIs) [RFC3987].

The `@base` or `BASE` directive defines the Base IRI used to resolve relative IRIs per RFC3986 section 5.1.1, "Base URI Embedded in Content". Section 5.1.2, "Base URI from the Encapsulating Entity" defines how the In-Scope Base IRI may come from an encapsulating document, such as a SOAP envelope with an xml:base directive or a mime multipart document with a Content-Location header. The "Retrieval URI" identified in 5.1.3, Base "URI from the Retrieval URI", is the URL from which a particular Turtle document was retrieved. If none of the above specifies the Base URI, the default Base URI (section 5.1.4, "Default Base URI") is used. Each `@base` or `BASE` directive sets a new In-Scope Base URI, relative to the previous one.

6.4 Escape Sequences

There are three forms of escapes used in turtle documents:

- *numeric escape sequences* represent Unicode code points:

Escape sequence	Unicode code point
'\u' hex hex hex hex	A Unicode character in the range U+0000 to U+FFFF inclusive corresponding to the value encoded by the four hexadecimal digits interpreted from most significant to least significant digit.
'\U' hex hex hex hex hex hex hex hex	A Unicode character in the range U+0000 to U+10FFFF inclusive corresponding to the value encoded by the eight hexadecimal digits interpreted from most significant to least significant digit.

  where HEX is a hexadecimal character

```
HEX ::= [0-9] | [A-F] | [a-f]
```

- *string escape sequences* represent the characters traditionally escaped in string literals:

Escape sequence	Unicode code point
'\t'	U+0009
'\b'	U+0008
'\n'	U+000A

'\r'	U+000D
'\f'	U+000C
'\"'	U+0022
'\''	U+0027
'\\'	U+005C

- *reserved character escape sequences* consist of a '\' followed by one of `~.-!$&'()*+,;=/?#@%_` and represent the character to the right of the '\'.

Context where each kind of escape sequence can be used

	numeric escapes	string escapes	reserved character escapes
IRIs, used as RDF terms or as in @prefix, PREFIX, @base, or BASE declarations	yes	no	no
local names	no	no	yes
Strings	yes	yes	no

NOTE

%-encoded sequences are in the character range for IRIs and are explicitly allowed in local names. These appear as a '%' followed by two hex characters and represent that same sequence of three characters. These sequences are *not* decoded during processing. A term written as `<http://a.example/%66oo-bar>` in Turtle designates the IRI `http://a.example/%66oo-bar` and not IRI `http://a.example/foo-bar`. A term written as `ex:%66oo-bar` with a prefix `@prefix ex: <http://a.example/>` also designates the IRI `http://a.example/%66oo-bar`.

6.5 Grammar

The EBNF used here is defined in XML 1.0 [EBNF-NOTATION]. Production labels consisting of a number and a final 's', e.g. [60s], reference the production with that number in the SPARQL 1.1 Query Language grammar [SPARQL11-QUERY].

Notes:

1. Keywords in single quotes ('`@base`', '`@prefix`', '`a`', '`true`', '`false`') are case-sensitive. Keywords in double quotes ("`BASE`", "`PREFIX`") are case-insensitive.
2. Escape sequences `UCHAR` and `ECHAR` are case sensitive.
3. When tokenizing the input and choosing grammar rules, the longest match is chosen.
4. The Turtle grammar is LL(1) and LALR(1) when the rules with uppercased names are used as terminals.
5. The entry point into the grammar is `turtleDoc`.
6. In signed numbers, no white space is allowed between the sign and the number.
7. The [162s] `ANON` ::= '`[`' WS* '`]`' token allows any amount of white space and comments between `[]`s. The single space version is used in the grammar for clarity.
8. The strings '`@prefix`' and '`@base`' match the pattern for LANGTAG, though neither "`prefix`" nor "`base`" are registered language subtags. This specification does not define whether a quoted literal followed by either of these tokens (e.g. `"A"@base`) is in the Turtle language.

```
[1]    turtleDoc                ::= statement*
[2]    statement                ::= directive | triples '.'
[3]    directive                ::= prefixID | base | sparqlPrefix | sparqlBase
[4]    prefixID                 ::= '@prefix' PNAME_NS IRIREF '.'
[5]    base                     ::= '@base' IRIREF '.'
[5s]   sparqlBase               ::= "BASE" IRIREF
[6s]   sparqlPrefix             ::= "PREFIX" PNAME_NS IRIREF
[6]    triples                  ::= subject predicateObjectList | blankNodePropertyList predicateObjectList?
[7]    predicateObjectList      ::= verb objectList (';' (verb objectList)?)*
[8]    objectList               ::= object (',' object)*
[9]    verb                     ::= predicate | 'a'
[10]   subject                  ::= iri | BlankNode | collection
[11]   predicate                ::= iri
[12]   object                   ::= iri | BlankNode | collection | blankNodePropertyList | literal
[13]   literal                  ::= RDFLiteral | NumericLiteral | BooleanLiteral
[14]   blankNodePropertyList    ::= '[' predicateObjectList ']'
[15]   collection               ::= '(' object* ')'
[16]   NumericLiteral           ::= INTEGER | DECIMAL | DOUBLE
[128s] RDFLiteral               ::= String (LANGTAG | '^^' iri)?
[133s] BooleanLiteral           ::= 'true' | 'false'
[17]   String                   ::= STRING_LITERAL_QUOTE | STRING_LITERAL_SINGLE_QUOTE |
                                    STRING_LITERAL_LONG_SINGLE_QUOTE | STRING_LITERAL_LONG_QUOTE
[135s] iri                      ::= IRIREF | PrefixedName
[136s] PrefixedName             ::= PNAME_LN | PNAME_NS
[137s] BlankNode                ::= BLANK_NODE_LABEL | ANON
```

Productions for terminals

```
[18]   IRIREF                   ::= '<' ([^#x00-#x20<>"{}|^`\] | UCHAR)* '>' /* #x00=NULL #01-#x1F=control codes
                                    #x20=space */
[139s] PNAME_NS                 ::= PN_PREFIX? ':'
```

```
[140s] PNAME_LN                          ::= PNAME_NS PN_LOCAL
[141s] BLANK_NODE_LABEL                  ::= '_:' (PN_CHARS_U | [0-9]) ((PN_CHARS | '.')* PN_CHARS)?
[144s] LANGTAG                           ::= '@' [a-zA-Z]+ ('-' [a-zA-Z0-9]+)*
[19]   INTEGER                           ::= [+-]? [0-9]+
[20]   DECIMAL                           ::= [+-]? [0-9]* '.' [0-9]+
[21]   DOUBLE                            ::= [+-]? ([0-9]+ '.' [0-9]* EXPONENT | '.' [0-9]+ EXPONENT | [0-9]+ EXPONENT)
[154s] EXPONENT                          ::= [eE] [+-]? [0-9]+
[22]   STRING_LITERAL_QUOTE              ::= '"' ([^#x22#x5C#xA#xD] | ECHAR | UCHAR)* '"' /* #x22=" #x5C=\ #xA=new line
                                             #xD=carriage return */
[23]   STRING_LITERAL_SINGLE_QUOTE       ::= "'" ([^#x27#x5C#xA#xD] | ECHAR | UCHAR)* "'" /* #x27=' #x5C=\ #xA=new line
                                             #xD=carriage return */
[24]   STRING_LITERAL_LONG_SINGLE_QUOTE  ::= "'''" (("'" | "''")? ([^'\] | ECHAR | UCHAR))* "'''"
[25]   STRING_LITERAL_LONG_QUOTE         ::= '"""' (('"' | '""')? ([^"\] | ECHAR | UCHAR))* '"""'
[26]   UCHAR                             ::= '\u' HEX HEX HEX HEX | '\U' HEX HEX HEX HEX HEX HEX HEX HEX
[159s] ECHAR                             ::= '\' [tbnrf"'\]
[161s] WS                                ::= #x20 | #x9 | #xD | #xA /* #x20=space #x9=character tabulation #xD=carriage
                                             return #xA=new line */
[162s] ANON                              ::= '[' WS* ']'
[163s] PN_CHARS_BASE                     ::= [A-Z] | [a-z] | [#x00C0-#x00D6] | [#x00D8-#x00F6] | [#x00F8-#x02FF] | [#x0370-
                                             #x037D] | [#x037F-#x1FFF] | [#x200C-#x200D] | [#x2070-#x218F] | [#x2C00-#x2FEF]
                                             | [#x3001-#xD7FF] | [#xF900-#xFDCF] | [#xFDF0-#xFFFD] | [#x10000-#xEFFFF]
[164s] PN_CHARS_U                        ::= PN_CHARS_BASE | '_'
[166s] PN_CHARS                          ::= PN_CHARS_U | '-' | [0-9] | #x00B7 | [#x0300-#x036F] | [#x203F-#x2040]
[167s] PN_PREFIX                         ::= PN_CHARS_BASE ((PN_CHARS | '.')* PN_CHARS)?
[168s] PN_LOCAL                          ::= (PN_CHARS_U | ':' | [0-9] | PLX) ((PN_CHARS | '.' | ':' | PLX)* (PN_CHARS | ':'
                                             | PLX))?
[169s] PLX                               ::= PERCENT | PN_LOCAL_ESC
[170s] PERCENT                           ::= '%' HEX HEX
[171s] HEX                               ::= [0-9] | [A-F] | [a-f]
[172s] PN_LOCAL_ESC                      ::= '\' ('_' | '~' | '.' | '-' | '!' | '$' | '&' | "'" | '(' | ')' | '*' | '+' | ','
                                             | ';' | '=' | '/' | '?' | '#' | '@' | '%')
```

7. Parsing

The RDF 1.1 Concepts and Abstract Syntax specification [RDF11-CONCEPTS] defines three types of *RDF Term*: IRIs, literals and blank nodes. Literals are composed of a lexical form and an optional language tag [BCP47] or datatype IRI. An extra type, `prefix`, is used during parsing to map string identifiers to namespace IRIs. This section maps a string conforming to the grammar in section 6.5 Grammar to a set of triples by mapping strings matching productions and lexical tokens to RDF terms or their components (e.g. language tags, lexical forms of literals). Grammar productions change the parser state and emit triples.

7.1 Parser State

Parsing Turtle requires a state of five items:

- IRI `baseURI` — When the base production is reached, the second rule argument, `IRIREF`, is the base URI used for relative IRI resolution.
- Map[prefix -> IRI] `namespaces` — The second and third rule arguments (`PNAME_NS` and `IRIREF`) in the prefixID production assign a namespace name (`IRIREF`) for the prefix (`PNAME_NS`). Outside of a `prefixID` production, any `PNAME_NS` is substituted with the namespace. Note that the prefix may be an empty string, per the `PNAME_NS` production: `(PN_PREFIX)? ":"`.
- Map[string -> blank node] `bnodeLabels` — A mapping from string to blank node.
- RDF_Term `curSubject` — The `curSubject` is bound to the `subject` production.
- RDF_Term `curPredicate` — The `curPredicate` is bound to the `verb` production. If token matched was "`a`", `curPredicate` is bound to the IRI `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`.

7.2 RDF Term Constructors

This table maps productions and lexical tokens to `RDF terms` or components of `RDF terms` listed in section 7. Parsing:

production	type	procedure
IRIREF	IRI	The characters between "<" and ">" are taken, with the numeric escape sequences unescaped, to form the unicode string of the IRI. Relative IRI resolution is performed per Section 6.3.
PNAME_NS	prefix	When used in a prefixID or sparqlPrefix production, the `prefix` is the potentially empty unicode string matching the first argument of the rule is a key into the namespaces map.
	IRI	When used in a PrefixedName production, the `iri` is the value in the namespaces map corresponding to the first argument of the rule.
PNAME_LN	IRI	A potentially empty prefix is identified by the first sequence, `PNAME_NS`. The namespaces map MUST have a corresponding `namespace`. The unicode string of the IRI is formed by unescaping the reserved characters in the second argument, `PN_LOCAL`, and concatenating this onto the `namespace`.
STRING_LITERAL_SINGLE_QUOTE	lexical form	The characters between the outermost "'"s are taken, with numeric and string escape sequences unescaped, to form the unicode string of a lexical form.
STRING_LITERAL_QUOTE	lexical form	The characters between the outermost '"'s are taken, with numeric and string escape sequences unescaped, to form the unicode string of a lexical form.
STRING_LITERAL_LONG_SINGLE_QUOTE	lexical form	The characters between the outermost "'''"s are taken, with numeric and string escape sequences unescaped, to form the unicode string of a lexical form.
STRING_LITERAL_LONG_QUOTE	lexical form	The characters between the outermost '"""'s are taken, with numeric and string escape sequences unescaped, to form the unicode string of a lexical form.
LANGTAG	language tag	The characters following the `@` form the unicode string of the language tag.

RDFLiteral	literal	The literal has a lexical form of the first rule argument, `String`. If the `'^^' iri` rule matched, the datatype is `iri` and the literal has no language tag. If the `LANGTAG` rule matched, the datatype is `rdf:langString` and the language tag is `LANGTAG`. If neither matched, the datatype is `xsd:string` and the literal has no language tag.
INTEGER	literal	The literal has a lexical form of the input string, and a datatype of `xsd:integer`.
DECIMAL	literal	The literal has a lexical form of the input string, and a datatype of `xsd:decimal`.
DOUBLE	literal	The literal has a lexical form of the input string, and a datatype of `xsd:double`.
BooleanLiteral	literal	The literal has a lexical form of the `true` or `false`, depending on which matched the input, and a datatype of `xsd:boolean`.
BLANK_NODE_LABEL	blank node	The string matching the second argument, `PN_LOCAL`, is a key in bnodeLabels. If there is no corresponding blank node in the map, one is allocated.
ANON	blank node	A blank node is generated.
blankNodePropertyList	blank node	A blank node is generated. Note the rules for `blankNodePropertyList` in the next section.
collection	blank node	For non-empty lists, a blank node is generated. Note the rules for `collection` in the next section.
	IRI	For empty lists, the resulting IRI is `rdf:nil`. Note the rules for `collection` in the next section.

7.3 RDF Triples Constructors

A Turtle document defines an RDF graph composed of set of RDF triples. The `subject` production sets the `curSubject`. The `verb` production sets the `curPredicate`. Each object `N` in the document produces an RDF triple: `curSubject curPredicate N` .

Property Lists:

Beginning the `blankNodePropertyList` production records the `curSubject` and `curPredicate`, and sets `curSubject` to a novel `blank node B`. Finishing the `blankNodePropertyList` production restores `curSubject` and `curPredicate`. The node produced by matching `blankNodePropertyList` is the blank node `B`.

Collections:

Beginning the `collection` production records the `curSubject` and `curPredicate`. Each `object` in the `collection` production has a `curSubject` set to a novel `blank node B` and a `curPredicate` set to `rdf:first`. For each object $\texttt{object}_n$ after the first produces a triple:$\texttt{object}_{n-1}$ `rdf:rest` $\texttt{object}_n$. Finishing the `collection` production creates an additional triple `curSubject rdf:rest rdf:nil` . and restores `curSubject` and `curPredicate` The node produced by matching `collection` is the first blank node `B` for non-empty lists and `rdf:nil` for empty lists.

7.4 Parsing Example

This section is non-normative.

The following informative example shows the semantic actions performed when parsing this Turtle document with an LALR(1) parser:

EXAMPLE 27

```
@prefix ericFoaf: <http://www.w3.org/People/Eric/ericP-foaf.rdf#> .
@prefix : <http://xmlns.com/foaf/0.1/> .
ericFoaf:ericP :givenName "Eric" ;
               :knows <http://norman.walsh.name/knows/who/dan-brickley> ,
                       [ :mbox <mailto:timbl@w3.org> ] ,
                       <http://getopenid.com/amyvdh> .
```

- Map the prefix `ericFoaf` to the IRI `http://www.w3.org/People/Eric/ericP-foaf.rdf#`.
- Map the empty prefix to the IRI `http://xmlns.com/foaf/0.1/`.
- Assign `curSubject` the IRI `http://www.w3.org/People/Eric/ericP-foaf.rdf#ericP`.
- Assign `curPredicate` the IRI `http://xmlns.com/foaf/0.1/givenName`.
- Emit an RDF triple: `<...rdf#ericP> <.../givenName> "Eric"` .
- Assign `curPredicate` the IRI `http://xmlns.com/foaf/0.1/knows`.
- Emit an RDF triple: `<...rdf#ericP> <.../knows> <...who/dan-brickley>` .
- Emit an RDF triple: `<...rdf#ericP> <.../knows> _:1` .
- Save `curSubject` and reassign to the blank node `_:1`.
- Save `curPredicate`.
- Assign `curPredicate` the IRI `http://xmlns.com/foaf/0.1/mbox`.
- Emit an RDF triple: `_:1 <.../mbox> <mailto:timbl@w3.org>` .
- Restore `curSubject` and `curPredicate` to their saved values (`<...rdf#ericP>`, `<.../knows>`).
- Emit an RDF triple: `<...rdf#ericP> <.../knows> <http://getopenid.com/amyvdh>` .

A. Embedding Turtle in HTML documents

This section is non-normative.

HTML [HTML5] `script` tags can be used to embed data blocks in documents. Turtle can be easily embedded in HTML this way.

EXAMPLE 28

```
<script type="text/turtle">
@prefix dc: <http://purl.org/dc/terms/> .
@prefix frbr: <http://purl.org/vocab/frbr/core#> .
```

```
<http://books.example.com/works/45U8QJGZSQKDH8N> a frbr:Work ;
     dc:creator "Wil Wheaton"@en ;
     dc:title "Just a Geek"@en ;
     frbr:realization <http://books.example.com/products/9780596007683.BOOK>,
         <http://books.example.com/products/9780596802189.EBOOK> .

<http://books.example.com/products/9780596007683.BOOK> a frbr:Expression ;
     dc:type <http://books.example.com/product-types/BOOK> .

<http://books.example.com/products/9780596802189.EBOOK> a frbr:Expression ;
     dc:type <http://books.example.com/product-types/EBOOK> .
</script>
```

Turtle content should be placed in a `script` tag with the `type` attribute set to `text/turtle`. `<` and `>` symbols do not need to be escaped inside of script tags. The character encoding of the embedded Turtle will match the HTML documents encoding.

A.1 XHTML

This section is non-normative.

Like JavaScript, Turtle authored for HTML (`text/html`) can break when used in XHTML (`application/xhtml+xml`). The solution is the same one used for JavaScript.

EXAMPLE 29

```
<script type="text/turtle">
# <![CDATA[
@prefix frbr: <http://purl.org/vocab/frbr/core#> .

<http://books.example.com/works/45U8QJGZSQKDH8N> a frbr:Work .
# ]]>
</script>
```

When embedded in XHTML Turtle data blocks must be enclosed in CDATA sections. Those CDATA markers must be in Turtle comments. If the character sequence "`]]>`" occurs in the document it must be escaped using strings escapes (`\u005d\u0054\u003e`). This will also make Turtle safe in polyglot documents served as both `text/html` and `application/xhtml+xml`. Failing to use CDATA sections or escape "`]]>`" may result in a non well-formed XML document.

A.2 Parsing Turtle in HTML

This section is non-normative.

There are no syntactic or grammar differences between parsing Turtle that has been embedded and normal Turtle documents. A Turtle document parsed from an HTML DOM will be a stream of character data rather than a stream of UTF-8 encoded bytes. No decoding is necessary if the HTML document has already been parsed into DOM. Each `script` data block is considered to be it's own Turtle document. `@prefix` and `@base` declarations in a Turtle data bloc are scoped to that data block and do not effect other data blocks. The HTML `lang` attribute or XHTML `xml:lang` attribute have no effect on the parsing of the data blocks. The base URI of the encapsulating HTML document provides a "Base URI Embedded in Content" per RFC3986 section 5.1.1.

B. Internet Media Type, File Extension and Macintosh File Type

Contact:
Eric Prud'hommeaux
See also:
How to Register a Media Type for a W3C Specification
Internet Media Type registration, consistency of use
TAG Finding 3 June 2002 (Revised 4 September 2002)

The Internet Media Type / MIME Type for Turtle is "text/turtle".

It is recommended that Turtle files have the extension ".ttl" (all lowercase) on all platforms.

It is recommended that Turtle files stored on Macintosh HFS file systems be given a file type of "TEXT".

This information that follows has been submitted to the IESG for review, approval, and registration with IANA.

Type name:
text
Subtype name:
turtle
Required parameters:
None
Optional parameters:
`charset` — this parameter is required when transferring non-ASCII data. If present, the value of `charset` is always `UTF-8`.
Encoding considerations:
The syntax of Turtle is expressed over code points in Unicode [UNICODE]. The encoding is always UTF-8 [UTF-8].
Unicode code points may also be expressed using an \uXXXX (U+0000 to U+FFFF) or \UXXXXXXXX syntax (for U+10000 onwards) where X is a hexadecimal digit [0-9A-Fa-f]
Security considerations:
Turtle is a general-purpose assertion language; applications may evaluate given data to infer more assertions or to dereference IRIs, invoking the security considerations of the scheme for that IRI. Note in particular, the privacy issues in [RFC3023] section 10 for HTTP IRIs. Data obtained from an inaccurate or malicious data source may lead to inaccurate or misleading conclusions, as well as the dereferencing of unintended IRIs. Care must be taken to align the trust in consulted resources with the sensitivity of the intended use of the data;

inferences of potential medical treatments would likely require different trust than inferences for trip planning.
Turtle is used to express arbitrary application data; security considerations will vary by domain of use. Security tools and protocols applicable to text (e.g. PGP encryption, MD5 sum validation, password-protected compression) may also be used on Turtle documents. Security/privacy protocols must be imposed which reflect the sensitivity of the embedded information.
Turtle can express data which is presented to the user, for example, RDF Schema labels. Application rendering strings retrieved from untrusted Turtle documents must ensure that malignant strings may not be used to mislead the reader. The security considerations in the media type registration for XML ([RFC3023] section 10) provide additional guidance around the expression of arbitrary data and markup.
Turtle uses IRIs as term identifiers. Applications interpreting data expressed in Turtle should address the security issues of Internationalized Resource Identifiers (IRIs) [RFC3987] Section 8, as well as Uniform Resource Identifier (URI): Generic Syntax [RFC3986] Section 7.
Multiple IRIs may have the same appearance. Characters in different scripts may look similar (a Cyrillic "o" may appear similar to a Latin "o"). A character followed by combining characters may have the same visual representation as another character (LATIN SMALL LETTER E followed by COMBINING ACUTE ACCENT has the same visual representation as LATIN SMALL LETTER E WITH ACUTE). Any person or application that is writing or interpreting data in Turtle must take care to use the IRI that matches the intended semantics, and avoid IRIs that make look similar. Further information about matching of similar characters can be found in Unicode Security Considerations [UNICODE-SECURITY] and Internationalized Resource Identifiers (IRIs) [RFC3987] Section 8.

Interoperability considerations:
There are no known interoperability issues.

Published specification:
This specification.

Applications which use this media type:
No widely deployed applications are known to use this media type. It may be used by some web services and clients consuming their data.

Additional information:

Magic number(s):
Turtle documents may have the strings '@prefix' or '@base' (case sensitive) or the strings 'PREFIX' or 'BASE' (case insensitive) near the beginning of the document.

File extension(s):
".ttl"

Base URI:
The Turtle '@base <IRIref>' or 'BASE <IRIref>' term can change the current base URI for relative IRIrefs in the query language that are used sequentially later in the document.

Macintosh file type code(s):
"TEXT"

Person & email address to contact for further information:
Eric Prud'hommeaux <eric@w3.org>

Intended usage:
COMMON

Restrictions on usage:
None

Author/Change controller:
The Turtle specification is the product of the RDF WG. The W3C reserves change control over this specifications.

C. Acknowledgements

This work was described in the paper New Syntaxes for RDF which discusses other RDF syntaxes and the background to the Turtle (Submitted to WWW2004, referred to as *N-Triples Plus* there).

This work was started during the Semantic Web Advanced Development Europe (SWAD-Europe) project funded by the EU IST-7 programme IST-2001-34732 (2002-2004) and further development supported by the Institute for Learning and Research Technology at the University of Bristol, UK (2002-Sep 2005).

Valuable contributions to this version were made by Gregg Kellogg, Andy Seaborn, Sandro Hawke and the members of the RDF Working Group.

The document was improved through the review process by the wider community.

D. Change Log

D.1 Changes since January 2014 Proposed Recommendation

- Missing prefix added in example 11 in response to comment from Lars Svensson.
- Error in grammar productions [21] and [23] fixed.
- Error in grammar productions [24] and [25] fixed.

D.2 Changes from February 2013 Candidate Recommendation to January 2014 Proposed Recommendation

- The addition of sparqlPrefix and sparqlBase which allow for using SPARQL style `BASE` and `PREFIX` directives in a Turtle document was marked "at risk" in the Candidate Recommendation publication. This feature is no longer at risk.
- The title of this document was changed from "*Turtle*" to "*RDF 1.1 Turtle*".
- Removed the obsolete links to tests in Sec. 7.1.

D.3 Changes from August 2011 First Public Working Draft to Candidate Recommendation

- Renaming for STRING_* productions to STRING_LITERAL_QUOTE sytle names rather than numbers
- Local part of prefix names can now include ":"
- Turtle in HTML
- Renaming of grammar tokens and rules around IRIs
- Reserved character escape sequences
- String escape sequences limited to strings
- Numeric escape sequences limited to IRIs and Strings

- Support top-level blank-predicate-object lists
- Whitespace required between @prefix and prefix label

D.4 Changes from January 2008 Team Submission to First Public Working Draft

- Adopted three additional string syntaxes from SPARQL: STRING_LITERAL2, STRING_LITERAL_LONG1, STRING_LITERAL_LONG2
- Adopted SPARQL's syntax for prefixed names (see editor's draft):
  - '.'s in names in all positions of a local name apart from the first or last, e.g. `ex:first.name`.
  - digits in the first character of the PN_LOCAL lexical token, e.g. `ex:7tm`.
- adopted SPARQL's IRI resolution and prefix substitution text.
- explicitly allowed re-use of the same prefix.
- Added parsing rules.

See also the pre-W3C Submission changelog.

E. References

E.1 Normative references

[BCP47]
A. Phillips; M. Davis. *Tags for Identifying Languages*. September 2009. IETF Best Current Practice. URL: http://tools.ietf.org/html/bcp47

[EBNF-NOTATION]
Tim Bray; Jean Paoli; C. M. Sperberg-McQueen; Eve Maler; François Yergeau. *EBNF Notation* 26 November 2008. W3C Recommendation. URL: http://www.w3.org/TR/REC-xml/#sec-notation

[RDF11-CONCEPTS]
Richard Cyganiak, David Wood, Markus Lanthaler. *RDF 1.1 Concepts and Abstract Syntax.* W3C Recommendation, 25 February 2014. URL: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/. The latest edition is available at http://www.w3.org/TR/rdf11-concepts/

[RFC2119]
S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels.* March 1997. Internet RFC 2119. URL: http://www.ietf.org/rfc/rfc2119.txt

[RFC3023]
M. Murata; S. St.Laurent; D. Kohn. *XML Media Types (RFC 3023)*. January 2001. RFC. URL: http://www.ietf.org/rfc/rfc3023.txt

[RFC3986]
T. Berners-Lee; R. Fielding; L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax (RFC 3986)*. January 2005. RFC. URL: http://www.ietf.org/rfc/rfc3986.txt

[RFC3987]
M. Dürst; M. Suignard. *Internationalized Resource Identifiers (IRIs)*. January 2005. RFC. URL: http://www.ietf.org/rfc/rfc3987.txt

[UNICODE]
The Unicode Standard. URL: http://www.unicode.org/versions/latest/

[UTF-8]
F. Yergeau. *UTF-8, a transformation format of ISO 10646*. IETF RFC 3629. November 2003. URL: http://www.ietf.org/rfc/rfc3629.txt

E.2 Informative references

[HTML5]
Robin Berjon; Steve Faulkner; Travis Leithead; Erika Doyle Navara; Edward O'Connor; Silvia Pfeiffer. *HTML5*. 4 February 2014. W3C Candidate Recommendation. URL: http://www.w3.org/TR/html5/

[N-TRIPLES]
Gavin Carothers, Andy Seabourne. *RDF 1.1 N-Triples*. W3C Recommendation, 25 February 2014. URL: http://www.w3.org/TR/2014/REC-n-triples-20140225/. The latest edition is available at http://www.w3.org/TR/n-triples/

[RDF11-MT]
Patrick J. Hayes, Peter F. Patel-Schneider. *RDF 1.1 Semantics.* W3C Recommendation, 25 February 2014. URL: http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/. The latest edition is available at http://www.w3.org/TR/rdf11-mt/

[SPARQL11-QUERY]
Steven Harris; Andy Seaborne. *SPARQL 1.1 Query Language*. 21 March 2013. W3C Recommendation. URL: http://www.w3.org/TR/sparql11-query/

[UNICODE-SECURITY]
Mark Davis; Michel Suignard. *Unicode Security Considerations*. URL: http://www.unicode.org/reports/tr36/