Turtle

Terse RDF Triple Language

W3C Working Draft 09 August 2011

This version:
http://www.w3.org/TR/2011/WD-turtle-20110809/
Latest published version:
http://www.w3.org/TR/turtle/
Latest editor's draft:
http://dvcs.w3.org/hg/rdf/raw-file/default/rdf-turtle/index.html
Editors:
Eric Prud'hommeaux, W3C
Gavin Carothers, TopQuadrant, Inc
Authors:
David Beckett
Tim Berners-Lee, W3C
Eric Prud'hommeaux, W3C

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 existing 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/.

Turtle is already a reasonably settled serialization of RDF. Many implementations of Turtle already exist, we are hoping for feedback from those existing implementers and other people deciding that now would be a good time to support Turtle. There are still a few rough edges that need polishing, and better alignment with the SPARQL triple patterns. The working group does not expect to make any large changes to the existing syntax.

This document was published by the RDF Working Group as a First Public Working Draft. This document is intended to become a W3C Recommendation. If you wish to make comments regarding this document, please send them to public-rdf-comments@w3.org (subscribe, archives). All feedback is welcome.

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.

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 document defines Turtle, the Terse RDF Triple Language, a concrete syntax for RDF as defined in the RDF Concepts and Abstract Syntax ([*RDF-CONCEPTS*]) W3C Recommendation. Turtle is an extension of N-Triples ([*N-TRIPLES*]) carefully taking the most useful and appropriate things added from Notation 3 ([*N3*]) while staying within the RDF model.

Issue

ISSUE-4: A future version of this document is expected to define N-Triples.

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

2. An Introduction to Turtle (Informative)

This section is informative. The Turtle Syntax and Turtle Grammar sections formally define the language.

A Turtle document allows writing down an RDF graph in a compact textual form. It consists of a sequence of directives, triple-generating statements or blank lines. Comments may be given after a `#` that is not part of another lexical token and continue to the end of the line.

Simple triples are a sequence of (subject, predicate, object) terms, separated by whitespace and terminated by '.' after each triple. This corresponds to N-Triples [*N-TRIPLES*].

Issue

ISSUE-4: A future version of this document is expected to define N-Triples.

There are three types of *RDF Term*: Internationalized Resource Identifiers (IRIs for short), literals and blank nodes.

2.1 RDF Terms

IRIs are written enclosed in '<' and '>' and may be absolute RDF URI References or relative to the current base IRI (described below).

Example

```
# this is not a complete turtle document
<http://example.org/path/>

<http://example.org/path/#fragment>
</path>
<#fragment>
<>
```

IRIs may also be abbreviated by using Turtle's `@prefix` directive that allows declaring a short prefix name for a long prefix of repeated IRIs. This is useful for many RDF vocabularies that are all defined with a common namespace like IRI.

Note

While `@prefix` works somewhat like XML namespaces the restrictions from XML QNames do NOT apply. `leg:3032571` is a perfectly fine prefixed name.

Once a prefix such as `@prefix foo: <http://example.org/ns#>` is defined, any mention of a URI later in the document may use a *prefixed name* that

starts `foo:` to stand for the longer IRI. So for example, the prefixed name `foo:bar` is a shorthand for the IRI `http://example.org/ns#bar`.

Example

```
# this is a complete turtle document
@prefix foo: <http://example.org/ns#> .
@prefix : <http://other.example.org/ns#> .
foo:bar foo: : .
:bar : foo:bar .
```

Literals are written either using double-quotes when they do not contain linebreaks like `"simple literal"` or `"""long literal"""` when they may contain linebreaks.

Example

```
# this is not a complete turtle document
"a string"
"""a string"""
"""a string
with newlines
"""
```

Literals have either a language suffix or a datatype IRI but not both. Languages are indicated by appending the simple literal with `@` and the language tag. Datatype IRIs similarly append `^^` followed by any legal IRI form (full or prefixed) as described above to give the datatype IRI. Literals may be written without either a language tag or a datatype IRI as a shortcut for a literal with the type `xsd:string`.

Example

```
# this is not a complete turtle document
"That Seventies Show"
"That Seventies Show"@en
"Cette Série des Années Soixante-dix"@fr
"Cette Série des Années Septante"@fr-be
"mylexicaldata"^^<http://example.org/my/datatype>
"""10"""^^xsd:decimal
```

The `"That Seventies Show"` above is equivalent to `"That Seventies Show"^^xsd:string`.

Issue

ISSUE-12 The RDF Working Group is currently examining a simplification of RDF which considers plain literals with no language tag to be literals with a datatype `xsd:string`.

Blank nodes are written as `_:`*BLANK_NODE_LABEL* to provide a blank node either from the given BLANK_NODE_LABEL. A generated blank node may also be made with `[]` which is useful to provide the subject of RDF triples for each pair from the predicateObjectList or the root of the collection.

Example

```
# this is not a complete turtle document
_:me
_:a1234
```

Literals , prefixed names and IRIs may also contain escapes to encode surrounding syntax, non-printable characters and to encode Unicode characters by codepoint number (although they may also be given directly, encoded as UTF-8). The character escapes are:

- `\t` (U+0009, tab)
- `\n` (U+000A, linefeed)
- `\r` (U+000D, carriage return)
- `\"` (U+0022, double quote - only allowed inside strings)
- `\>` (U+003E, greater than - only allowed inside IRI_REFs)
- `\\` (U+005C, backslash)
- `\u`*HHHH* or `\U`*HHHHHHHH* for writing Unicode characters by hexadecimal codepoint where *H* is a single hexadecimal digit.

See the String escapes section for full details.

Issue

ISSUE 67 The inclusion of escape sequences in prefixed names is undecided.

2.2 Abbreviating IRIs

The current base IRI may be altered in a Turtle document using the `@base` directive. It allows further abbreviation of IRIs but is usually for simplifying the IRIs in the data, where the prefix directives are for vocabularies that describe the data.

Whenever this directive appears, it defines the base IRI for which all relative IRIs are resolved against. That includes IRIs, qualified names, prefix directives as well as later base directives.

Example

```
# this is a complete turtle document
# In-scope base URI is the document URI at this point
<a1> <b1> <c1> .
@base <http://example.org/ns/> .
# In-scope base URI is http://example.org/ns/ at this point

<a2> <http://example.org/ns/b2> <c2> .
@base <foo/> .
# In-scope base URI is http://example.org/ns/foo/ at this point
<a3> <b3> <c3> .
@prefix : <bar#> .
:a4 :b4 :c4 .
@prefix : <http://example.org/ns2#> .
:a5 :b5 :c5 .
```

The token `a` is equivalent to the IRI `<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>`

Example

```
# this is a complete turtle document
@prefix doc: <http://example.org/#ns> .

<http://example.org/path> a doc:Document .
```

2.3 Abbreviating common datatypes

Decimal integers may be written directly and correspond to the XML Schema Datatype xsd:integer in both syntax and datatype IRI.

Example

```
# this is not a complete turtle document
-5
0
1
10
+1
# some long form examples
"-5"^^xsd:integer
"10"^^<http://www.w3.org/2001/XMLSchema#integer>
```

Decimal floating point double/fixed precision numbers may be written directly and correspond to the XML Schema Datatype xsd:double in both syntax and datatype IRI.

Example

```
# this is not a complete turtle document
1.3e2
10e0
-12.5e10
# some long form examples
"1.3e2"^^xsd:double
"-12.5e10"^^<http://www.w3.org/2001/XMLSchema#double>
```

Decimal floating point arbitrary precision numbers may be written directly and correspond to the XML Schema Datatype xsd:decimal. in both syntax and datatype IRI.

Example

```
# this is not a complete turtle document
0.0
1.0
1.234567890123456789
-5.0
# some long form examples
"0.0"^^xsd:decimal
"-5.0"^^<http://www.w3.org/2001/XMLSchema#decimal>
```

Boolean may be written directly as `true` or `false` and correspond to the the XML Schema Datatype xsd:boolean in both syntax and datatype IRI.

Example

```
-

# this is not a complete turtle document
true
false
# same in long form
"true"^^xsd:boolean
"false"^^<http://www.w3.org/2001/XMLSchema#boolean>
```

2.4 Abbreviating groups of triples

The `,` symbol may be used to repeat the subject and predicate of triples that only differ in the object RDF term.

Example

```
# this is not a complete turtle document
:subject :predicate :object1 ,
    :object2 .
# creates two triples, the last triple is :subject :predicate :object2 .
```

The `;` symbol may be used to repeat the subject of triples that vary only in predicate and object RDF terms.

Example

```
# this is not a complete turtle document
:subject :predicate1 :obj1 ;
:predicate2 :obj2 .
# creates two triples, the last triple is :subject :predicate2 :obj2 .
```

2.5 Abbreviating RDF Collections

An RDF Collection may be abbreviated using a sequence of RDF Terms enclosed in `( )` brackets. Whitespace may be used to separate them, as usual. This format provides a blank node at the start of RDF Collection which may be used in further abbreviations.

Example

```
# this is a complete turtle document
@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 () .
```

See section Collections for the details on the long form of the generated triples.

3. Syntax for IRIs, Literals and Blank Nodes

Turtle is a language for an RDF graph, a set of RDF triples. An RDF graph is composed of URI references (now interpreted as IRIs), literals and blank nodes.

The Turtle syntax for IRIs is identical to that of SPARQL Query, including the use of `prefix` and `base` directives, though these are spelled `@prefix` and `@base` respectively in Turtle. Per RFC3986 section 5.1.1 [*RFC3986*], the parsing begins with a context-defined In-Scope Base URI. Each `@base` directive sets a new In-Scope Base URI, relative to the previous one. `@prefix` directives map a local name to an IRI, also resolved against the current In-Scope Base URI. Subsequent `@prefix` may re-map the same local name.

Turtle IRI syntax, including relative IRI resolution, is defined by SPARQL Query section 4.1.1 (noting the different spellings of the `PREFIX` and `BASE` keywords).

Example (test-30.ttl) with document base IRI http://www.w3.org/2001/sw/DataAccess/df1/tests/

Example

```
# In-scope base URI is http://www.w3.org/2001/sw/DataAccess/df1/tests/ at this point
<a1> <b1> <c1> .
@base <http://example.org/ns/> .
# In-scope base URI is http://example.org/ns/ at this point
<a2> <http://example.org/ns/b2> <c2> .
@base <foo/> .
# In-scope base URI is http://example.org/ns/foo/ at this point
<a3> <b3> <c3> .
```

```
@prefix : <bar#> .
:a4 :b4 :c4 .
@prefix : <http://example.org/ns2#> .
:a5 :b5 :c5 .
```

encodes the following N-Triples (test-30.out):

```
Example

<http://www.w3.org/2001/sw/DataAccess/df1/tests/a1> <http://www.w3.org/2001/sw/DataAccess/df1/tests/b1> <http://www.w3.org/2001/sw/Dat
<http://example.org/ns/a2> <http://example.org/ns/b2> <http://example.org/ns/c2> .
<http://example.org/ns/foo/a3> <http://example.org/ns/foo/b3> <http://example.org/ns/foo/c3> .
<http://example.org/ns/foo/bar#a4> <http://example.org/ns/foo/bar#b4> <http://example.org/ns/foo/bar#c4> .
<http://example.org/ns2#a5> <http://example.org/ns2#b5> <http://example.org/ns2#c5> .
```

The Turtle syntax for literals and blank nodes are defined by SPARQL Query section 4.1.2 and SPARQL Query section 4.1.4 respectively.

4. Turtle Grammar

A Turtle document is a Unicode[*UNICODE*] character string encoded in UTF-8. Unicode codepoints only in the range U+0 to U+10FFFF inclusive are allowed.

4.1 White Space

White space (production ws) is used to separate two tokens which would otherwise be (mis-)recognized as one token.

White space is significant in tokens IRI_REF and string.

4.2 Comments

Comments in Turtle take the form of '#', outside an IRI_REF or strings, 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.

4.3 String Escapes

Turtle strings and IRIs can use \-escape sequences to represent Unicode code points.

The following table describes all the escapes allowed inside a string or IRI_REF:

Escape	Unicode code point
'\u' hex hex hex hex	A Unicode codepoint in the range U+0 to U+FFFF inclusive corresponding to the encoded hexadecimal value.
'\U' hex hex hex hex hex hex hex hex	A Unicode codepoint in the range U+10000 to U+10FFFF inclusive corresponding to the encoded hexadecimal value.
'\t'	U+0009
'\n'	U+000A
'\r'	U+000D
'\"' (inside string)	U+0022
'\>' (inside IRI_REF only)	U+003E
'\\'	U+005C

where HEX is a hexadecimal character

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

4.4 Grammar

The EBNF used here is defined in XML 1.0 (Third Edition) [*EBNF-NOTATION*]. Production labels consisting of a number and a final 's', e.g. [60s], reference to the production with that number in the SPARQL Query Language for RDF grammar [*RDF-SPARQL-QUERY*].

Note

There are known formating issues with the table form of the grammar. Please see turtle.bnf for exact grammar.

Turtle - Terse RDF Triple Language EBNF

[1]	turtleDoc	::=	(statement)*
[2]	statement	::=	directive "." | triples "."
[3]	directive	::=	prefixID | base
[4]	prefixID	::=	PREFIX PNAME_NS IRI_REF

```
[5]     base                    ::= BASE IRI_REF
[6]     triples                 ::= subject predicateObjectList
[7]     predicateObjectList     ::= verb objectList ( ";" verb objectList )* (";")?
[8]     objectList              ::= object ( "," object )*
[9]     verb                    ::= predicate
                                  | "a"
[10]    subject                 ::= IRIref
                                  | blank
[11]    predicate               ::= IRIref
[12]    object                  ::= IRIref
                                  | blank
                                  | literal
[13]    literal                 ::= RDFLiteral
                                  | NumericLiteral
                                  | BooleanLiteral
[14]    blank                   ::= BlankNode
                                  | blankNodePropertyList
                                  | collection
[15]    blankNodePropertyList   ::= "[" predicateObjectList "]"
[16]    collection              ::= "(" object* ")"
[60s]   RDFLiteral              ::= String ( LANGTAG | ( "^^" IRIref ) )?
[61s]   NumericLiteral          ::= NumericLiteralUnsigned
                                  | NumericLiteralPositive
                                  | NumericLiteralNegative
[62s]   NumericLiteralUnsigned  ::= INTEGER
                                  | DECIMAL
                                  | DOUBLE
[63s]   NumericLiteralPositive  ::= INTEGER_POSITIVE
                                  | DECIMAL_POSITIVE
                                  | DOUBLE_POSITIVE
[64s]   NumericLiteralNegative  ::= INTEGER_NEGATIVE
                                  | DECIMAL_NEGATIVE
                                  | DOUBLE_NEGATIVE
[65s]   BooleanLiteral          ::= "true"
                                  | "false"
[66s]   String                  ::= STRING_LITERAL1
                                  | STRING_LITERAL2
                                  | STRING_LITERAL_LONG1
                                  | STRING_LITERAL_LONG2
[67s]   IRIref                  ::= IRI_REF
                                  | PrefixedName
[68s]   PrefixedName            ::= PNAME_LN
                                  | PNAME_NS
[69s]   BlankNode               ::= BLANK_NODE_LABEL
                                  | ANON
[17]    <BASE>                  ::= "@base"
[18]    <PREFIX>                ::= "@prefix"
[70s]   <IRI_REF>               ::= "<" ( [^<>\"{}|^`\\] - [#0000- ] | UCHAR )* ">"
[71s]   <PNAME_NS>              ::= (PN_PREFIX)? ":"
[72s]   <PNAME_LN>              ::= PNAME_NS PN_LOCAL
[73s]   <BLANK_NODE_LABEL>      ::= "_:" PN_LOCAL
[74s]   <VAR1>                  ::= "?" VARNAME
[75s]   <VAR2>                  ::= "$" VARNAME
[76s]   <LANGTAG>               ::= BASE
                                  | PREFIX
                                  | "@" [a-zA-Z]+ ( "-" [a-zA-Z0-9]+ )*
[77s]   <INTEGER>               ::= [0-9]+
[78s]   <DECIMAL>               ::= [0-9]+ "." [0-9]*
                                  | "." [0-9]+
[79s]   <DOUBLE>                ::= [0-9]+ "." [0-9]* EXPONENT
                                  | "." ( [0-9] )+ EXPONENT
                                  | ( [0-9] )+ EXPONENT
[80s]   <INTEGER_POSITIVE>      ::= "+" INTEGER
[81s]   <DECIMAL_POSITIVE>      ::= "+" DECIMAL
```

```
[82s]   <DOUBLE_POSITIVE>        ::=  "+" DOUBLE
[83s]   <INTEGER_NEGATIVE>       ::=  "-" INTEGER
[84s]   <DECIMAL_NEGATIVE>       ::=  "-" DECIMAL
[85s]   <DOUBLE_NEGATIVE>        ::=  "-" DOUBLE
[86s]   <EXPONENT>               ::=  [eE] [+-]? [0-9]+
[87s]   <STRING_LITERAL1>        ::=  "'" ( ( [^'\\\n\r] ) | ECHAR | UCHAR )* "'"
[88s]   <STRING_LITERAL2>        ::=  '"' ( ( [^\"\\\n\r] ) | ECHAR | UCHAR )* '"'
[89s]   <STRING_LITERAL_LONG1>   ::=  "'''" ( ( "'" | "''" )? ( [^'\\] | ECHAR | UCHAR ) )* "'''"
[90s]   <STRING_LITERAL_LONG2>   ::=  '"""' ( ( '"' | '""' )? ( [^\"\\] | ECHAR | UCHAR ) )* '"""'
[19]    <UCHAR>                  ::=  ( "\\u" [0-9a-fA-F] [0-9a-fA-F] [0-9a-fA-F] [0-9a-fA-F] )
                                    | ( "\\U" [0-9a-fA-F] [0-9a-fA-F] [0-9a-fA-F] [0-9a-fA-F] [0-9a-fA-F] [0-9a-fA-F] [0-9a-fA-F] [0-
                                    9a-fA-F] )
[91s]   <ECHAR>                  ::=  "\\" [tbnrf\\\"']
[92s]   <NIL>                    ::=  "(" (WS)* ")"
[93s]   <WS>                     ::=  " "
                                    | "\t"
                                    | "\r"
                                    | "\n"
[94s]   <ANON>                   ::=  "[" (WS)* "]"
[95s]   <PN_CHARS_BASE>          ::=  [A-Z]
                                    | [a-z]
                                    | [#00C0-#00D6]
                                    | [#00D8-#00F6]
                                    | [#00F8-#02FF]
                                    | [#0370-#037D]
                                    | [#037F-#1FFF]
                                    | [#200C-#200D]
                                    | [#2070-#218F]
                                    | [#2C00-#2FEF]
                                    | [#3001-#D7FF]
                                    | [#F900-#FDCF]
                                    | [#FDF0-#FFFD]
                                    | [#10000-#EFFFF]
                                    | UCHAR
[96s]   <PN_CHARS_U>             ::=  PN_CHARS_BASE
                                    | "_"
[97s]   <VARNAME>                ::=  ( PN_CHARS_U | [0-9] ) ( PN_CHARS_U | [0-9] | #00B7 | [#0300-#036F] | [#203F-#2040] )*
[98s]   <PN_CHARS>               ::=  PN_CHARS_U
                                    | "-"
                                    | [0-9]
                                    | #00B7
                                    | [#0300-#036F]
                                    | [#203F-#2040]
[99s]   <PN_PREFIX>              ::=  PN_CHARS_BASE ( ( PN_CHARS | "." )* PN_CHARS )?
[100s]  <PN_LOCAL>               ::=  ( PN_CHARS_U | [0-9] ) ( ( PN_CHARS | "." )* PN_CHARS )?
[-]     PASSED TOKENS            ::=  [ \t\r\n]+
                                    | "#" [^\r\n]*
```

5. Parsing

The RDF Concepts and Abstract Syntax ([*RDF-CONCEPTS*]) specification defines three types of *RDF Term*: RDF URI References (here called IRIs), literals and blank nodes. Literals are composed of a lexical form and an optional language tag 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 4.4 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). Some productions change the parser state (base or prefix declarations).

5.1 Parser State

Parsing Turtle requires a state of four items:

- IRI `baseURI` — When the base production is reached, the second rule argument, `IRI_REF`, is the base URI used for relative IRI resolution (test: base1 base2).
- Map[prefix -> IRI] `namespaces` — The second and third rule arguments (`PNAME_NS` and `IRI_REF`) in the prefixID production assign a namespace name (`IRI_REF`) for the prefix (`PNAME_NS`). Outside of a `prefixID` production, any `PNAME_NS` is substituted with the namespace (test: prefix1 escapedNamespace1). Note that the prefix may be an empty string, per the `PNAME_NS,` production: `(PN_PREFIX)? ":"` (test: default1).
- Map[string -> blank node] `bnodeLabels` — A mapping from string to blank node label.
- 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` (test: type).

5.2 RDF Term Constructors

This table maps productions and lexical tokens to `RDF terms` or components of `RDF terms` listed in section 5:

production	type	procedure
IRI_REF	IRI	The characters between "<" and ">" are unescaped[1] to form the unicode string of the IRI. Relative IRI resolution is performed per SPARQL Query section 4.1.1.
PNAME_NS	prefix	The potentially empty unicode string matching the first argument of the rule is a key into the namespaces map.
PNAME_LN	IRI	A prefix is identified by the first argument, `PNAME_NS`. The namespaces map has a corresponding `namespace`. The unicode string of the IRI is formed by concatenating this `namespace` and the second argument, `PN_LOCAL`. Relative IRI resolution is performed per SPARQL Query section 4.1.1.
STRING_LITERAL1	lexical form	The characters between the outermost ""'s are unescaped[1] to form the unicode string of a lexical form.
STRING_LITERAL2	lexical form	The characters between the outermost "'s are unescaped[1] to form the unicode string of a lexical form.
STRING_LITERAL_LONG1	lexical form	The characters between the outermost """"s are unescaped[1] to form the unicode string of a lexical form.
STRING_LITERAL_LONG2	lexical form	The characters between the outermost """"s are unescaped[1] 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`) and either a language tag of `LANGTAG` or a datatype IRI of `IRIref`, depending on which rule matched the input.
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	A blank node is generated. Note the rules for `collection` in the next section.

[1] Section 3.3 defines an mapping from `escaped unicode strings` to `unicode strings`. The following lexical tokens are unescaped to produce `unicode strings`: IRI_REF, STRING_LITERAL1, STRING_LITERAL2, STRING_LITERAL_LONG1 and STRING_LITERAL_LONG2 .

5.3 RDF Triples Constructors

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

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`.

Beginning the `collection` production records the `curSubject` and `curPredicate`, sets `curSubject` to a novel `blank node` B_{head} and sets `curSubject` and `curPredicate` to B_{head} and `rdf:first` respectively. Each object `O` in `collection` allocates a novel `blank node` B_n, creates an additional triple `curSubject rdf:rest` B_n . and sets `curSubject` to B_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 blank node B_{head}.

5.4 Parsing Example (Informative)

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

```
@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> .`

6. Examples (Informative)

This section is non-normative.

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

Example

```
@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

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

which is short for (example2.ttl):

Example

```
@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. Assumes that line feeds in this document are #xA. (example3.ttl):

Example

```
@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

```
(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

```
_: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

```
(1 [:p :q] ( 2 ) ) .
```

is syntactic sugar for:

Example

```
_: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 .
```

7. Identifiers for the Turtle Language

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

The XML (Namespace name, Local name) pair that identifies the Turtle language is:
Namespace: `http://www.w3.org/ns/formats/Turtle`
Local name: `turtle`
The suggested namespace prefix is `ttl` (informative) which would make this `ttl:turtle` as an XML QName.

Issue

Previous versions of the Turtle specified `http://www.w3.org/2008/turtle#turtle` as the IRI for the Turtle language. This change aligns Turtle with identifiers for RDF/XML, N3, POWDER, etc

8. Conformance

Systems conforming to Turtle MUST pass all the following test cases:

1. The N-Triples tests in the RDF Test Cases W3C Recommendation.
2. The Turtle Test Suite

   Passing these tests means:

   1. All the `test-n.ttl` tests MUST generate equivalent RDF triples to those given in the corresponding `test-n.out` N-Triples file.
   2. All the `bad-n.ttl` tests MUST NOT generate RDF triples.

9. 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 B. Internet Media Type, File Extension and Macintosh File Type for the media type registration form.

10. Turtle compared

Turtle is related to a number of other languages.

10.1 Turtle compared to N-Triples (Informative)

This section is non-normative.

Issue

ISSUE-4: A future version of this document is expected to define N-Triples. By default, the RDF WG will specify N-Triples to allow UTF-8 characters in IRIs, literals and blank node identifiers. Readers with an opinion about whether or not N-Triples should be ASCII-only may wish to comment.

All N-Triples files are vaild Turtle documents. Turtle adds the following syntax to N-Triples:

1. Whitespace restrictions removed
2. Text content-encoding changed from ASCII to UTF-8
3. Three additional string syntaxes: STRING_LITERAL2, STRING_LITERAL_LONG1, STRING_LITERAL_LONG2
4. `@base` directive for setting a base IRI
5. `@prefix` directive for assigning namespace prefixes
6. Prefixed names
7. Object lists separated by `,`
8. Predicate object lists separated by `;`
9. Unlabled blank nodes indicated by `[]`
10. `rdf:type` shorthand `a`
11. RDF collection constructor bound by `()`s
12. Decimal integer literals of type `xsd:integer`
13. Decimal double literals of type `xsd:double`
14. Decimal arbitrary length literals of type `xsd:decimal`
15. Boolean literals of type `xsd:boolean`

10.2 Turtle compared to Notation 3 (Informative)

This section is non-normative.

Turtle is similar to and inspired by the more powerful Notation 3 (N3). Please see the most recent Notation3 specification for comparison with Turtle.

10.3 Turtle compared to RDF/XML (Informative)

This section is non-normative.

RDF/XML ([*RDF-SYNTAX-GRAMMAR*]) has certain restrictions imposed by XML and the use of XML Namespaces that prevent it encoding all RDF graphs (some predicate URIs are forbidden and XML 1.0 forbids encoding some Unicode codepoints). These restrictions do not apply to Turtle.

10.4 Turtle compared to SPARQL (Informative)

This section is non-normative.

The SPARQL Query Language for RDF (SPARQL) [*RDF-SPARQL-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 (per Last Call draft)
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).

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

A. Internet Media Type, File Extension and Macintosh File Type (Normative)

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+0 to U+FFFF) or \UXXXXXXXX syntax (for U+10000 onwards) where X is a hexadecimal digit [0-9A-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 [*UNISEC*] 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 dependent) near the beginning of the document.

File extension(s):
".ttl"

Base URI:
The Turtle '@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 David Beckett and Tim Berners-Lee. A W3C Working Group may assume maintenance of this document; W3C reserves change control over this specifications.

B. Acknowledgements (Informative)

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).

C. Changes (Informative)

Changes since the last publication of this document W3C Turtle Submission 2008-01-14 . See the Previous changelog for further information

- Adopted three additional string syntaxes from SPARQL: STRING_LITERAL2, STRING_LITERAL_LONG1, STRING_LITERAL_LONG2
- Adopted case-independent constants for XSD booleans `true` and `false`.
- 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`.
- Made syntax section normative.
  - adopted SPARQL's IRI resolution and prefix substitution text.
  - explicitly allowed re-use of the same prefix.
- Added parsing rules.

D. References

D.1 Normative references

[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

[N-TRIPLES]
Jan Grant; Dave Beckett. *N-Triples* 10 February 2004. W3C Recommendation. URL: http://www.w3.org/TR/rdf-testcases/#ntriples

[RDF-CONCEPTS]
Graham Klyne; Jeremy J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax.* 10 February 2004. W3C Recommendation. URL: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/
(The referenced RDF Concepts document has yet not been re-published by the RDF Working Group. It is included here instead of the last published version as it includes new material relevant to the interpretation of the Turtle specification.)

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

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

[RFC3987]
M. Dürst; M. Suignard. *Internationalized Resource Identifiers (IRIs).* January 2005. Internet RFC 3987. URL: http://www.ietf.org/rfc/rfc3987.txt

[UNICODE]
The Unicode Consortium. *The Unicode Standard.* 2003. Defined by: The Unicode Standard, Version 4.0 (Boston, MA, Addison-Wesley, ISBN 0-321-18578-1), as updated from time to time by the publication of new versions URL: http://www.unicode.org/unicode/standard/versions/enumeratedversions.html

[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

D.2 Informative references

[N3]
Tim Berners-Lee; Dan Connolly. *Notation3 (N3): A readable RDF syntax.* 14 January 2008. W3C Team Submission. URL: http://www.w3.org/TeamSubmission/2008/SUBM-n3-20080114/

[RDF-SPARQL-QUERY]
Andy Seaborne; Eric Prud'hommeaux. *SPARQL Query Language for RDF.* 15 January 2008. W3C Recommendation. URL: http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/

[RDF-SYNTAX-GRAMMAR]
Dave Beckett. *RDF/XML Syntax Specification (Revised).* 10 February 2004. W3C Recommendation. URL: http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/

[UNISEC]
Mark Davis; Michel Suignard. *Unicode Security Considerations* 4 August 2010. URL: http://www.unicode.org/reports/tr36/