RDF 1.1 TriG

RDF Dataset Language

W3C Recommendation 25 February 2014

This version:
http://www.w3.org/TR/2014/REC-trig-20140225/
Latest published version:
http://www.w3.org/TR/trig/
Test suite:
http://www.w3.org/TR/2014/NOTE-rdf11-testcases-20140225/
Implementation report:
http://www.w3.org/2013/TriGReports/index.html
Previous version:
http://www.w3.org/TR/2014/PR-trig-20140109/
Editors:
Gavin Carothers, Lex Machina
Andy Seaborne, Apache Software Foundation
Authors:
Chris Bizer, Freie Universität Berlin
Richard Cyganiak, Freie Universität Berlin

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

This document defines a textual syntax for RDF called TriG that allows an RDF dataset to be completely written in a compact and natural text form, with abbreviations for common usage patterns and datatypes. TriG is an extension of the Turtle [TURTLE] format.

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 part of the RDF 1.1 document suite. TriG is intended the meet the charter requirement of the RDF Working Group to define an RDF syntax for multiple graphs. TriG is an extension of the Turtle syntax for RDF [TURTLE]. The current

document is based on the original proposal by Chris Bizer and Richard Cyganiak.

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 document defines TriG, a concrete syntax for RDF as defined in the RDF Concepts and Abstract Syntax document [RDF11-CONCEPTS]. TriG is an extension of Turtle [TURTLE], extended to support representing a complete RDF Dataset.

2. TriG Language

This section is non-normative.

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

Graph statements are a pair of an IRI or blank node label and a group of triple statements surrounded by {}. The IRI or blank node label of the graph statement may be used in another graph statement which implies taking the union of the tripes generated by each graph statement. An IRI or blank node label used as a graph label may also reoccur as part of any triple statement. Optionally a graph statement may not not be labeled with an IRI. Such a graph statement corresponds to the Default Graph of an RDF Dataset.

The construction of an RDF Dataset from a TriG document is defined in section 4. TriG Grammar and section 5. Parsing.

2.1 Triple Statements

As TriG is an extention of the Turtle language it allows for any constructs from the Turtle language. Simple Triples, Predicate Lists, and Object Lists can all be used either inside a graph statement, or on their own as in a Turtle document. When outside a graph statement, the triples are considered to be part of the default graph of the RDF Dataset.

2.2 Graph Statements

A graph statement pairs an IRI or blank node with a RDF graph. The triple statements that make up the graph are enclosed in {}.

In a TriG document a graph IRI or blank node may be used as label for more than one graph statements. The graph label of a graph statement may be omitted. In this case the graph is considered the default graph of the RDF Dataset.

A RDF Dataset might contain only a single graph.

EXAMPLE 1

```
# This document encodes one graph.
@prefix ex: <http://www.example.org/vocabulary#> .
@prefix : <http://www.example.org/exampleDocument#> .

:G1 { :Monica a ex:Person ;
              ex:name "Monica Murphy" ;
              ex:homepage <http://www.monicamurphy.org> ;
              ex:email <mailto:monica@monicamurphy.org> ;
              ex:hasSkill ex:Management ,
                          ex:Programming . }
```

A RDF Dataset may contain a default graph, and named graphs.

EXAMPLE 2

```
# This document contains a default graph and two named graphs.

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dc: <http://purl.org/dc/terms/> .
```

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

# default graph
    {
      <http://example.org/bob> dc:publisher "Bob" .
      <http://example.org/alice> dc:publisher "Alice" .
    }

<http://example.org/bob>
    {
       _:a foaf:name "Bob" .
       _:a foaf:mbox <mailto:bob@oldcorp.example.org> .
       _:a foaf:knows _:b .
    }

<http://example.org/alice>
    {
       _:b foaf:name "Alice" .
       _:b foaf:mbox <mailto:alice@work.example.org> .
    }
```

TriG provides various alternative ways to write graphs and triples, giving the data writer choices for clarity:

EXAMPLE 3

```
# This document contains a same data as the
previous example.

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dc: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

# default graph - no {} used.
<http://example.org/bob> dc:publisher "Bob" .
<http://example.org/alice> dc:publisher "Alice" .

# GRAPH keyword to highlight a named graph
# Abbreviation of triples using ;
GRAPH <http://example.org/bob>
{
   [] foaf:name "Bob" ;
      foaf:mbox <mailto:bob@oldcorp.example.org> ;
      foaf:knows _:b .
}

GRAPH <http://example.org/alice>
{
    _:b foaf:name "Alice" ;
        foaf:mbox <mailto:alice@work.example.org>
}
```

2.3 Other Terms

All other terms and directives come from Turtle.

2.3.1 Special Considerations for Blank Nodes

BlankNodes sharing the same label in differently labeled graph statements are considered to be the same BlankNode.

3. 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:

- TriG documents
- TriG parsers

A conforming **TriG document** is a Unicode string that conforms to the grammar and additional constraints defined in section 4. TriG Grammar, starting with the `trigDoc` production. A TriG document serializes an RDF dataset.

A conforming **TriG parser** is a system capable of reading TriG documents on behalf of an application. It makes the serialized RDF dataset, as defined in section 5. Parsing, available to the application, usually through some form of API.

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

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

3.1 Media Type and Content Encoding

The media type of TriG is `application/trig`. The content encoding of TriG content is always UTF-8.

4. TriG Grammar

A TriG 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.

4.1 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 TriG parser.

White space is significant in the production String.

4.2 Comments

Comments in TriG take the form of '#', outside an IRI or a 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.

4.3 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` 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 TriG 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` directive sets a new In-Scope Base URI, relative to the previous one.

4.4 Escape Sequences

There are three forms of escapes used in TriG 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 or @base declarations	yes	no	no
local names	no	no	yes

	numeric escapes	string escapes	reserved character escapes
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 TriG 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`.

4.5 Grammar

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

Notes:

1. A blank node label represents the same blank node throughout the TriG document.
2. Keywords in single quotes ('`@base`', '`@prefix`', '`a`', '`true`', '`false`') are case-sensitive. Keywords in double quotes ("`BASE`", "`PREFIX`" "`GRAPH`") are case-insensitive.
3. Escape sequences markers `\u`, `\U` and those in `ECHAR` are case sensitive.
4. When tokenizing the input and choosing grammar rules, the longest match is chosen.
5. The TriG grammar is LL(1) and LALR(1) when the rules with uppercased names are used as terminals.
6. The entry point into the grammar is `trigDoc`.
7. In signed numbers, no white space is allowed between the sign and the number.
8. 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.
9. 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. `"z"@base`) is in the TriG language.

```
[1g]   trigDoc                ::= (directive | block)*
[2g]   block                  ::= triplesOrGraph | wrappedGraph |
                                  triples2 | "GRAPH" labelOrSubject
                                  wrappedGraph
[3g]   triplesOrGraph         ::= labelOrSubject (wrappedGraph |
                                  predicateObjectList '.')
[4g]   triples2               ::= blankNodePropertyList
                                  predicateObjectList? '.' |
                                  collection predicateObjectList '.'
[5g]   wrappedGraph           ::= '{' triplesBlock? '}'
[6g]   triplesBlock           ::= triples ('.' triplesBlock?)?
[7g]   labelOrSubject         ::= iri | BlankNode
```

```
[3]    directive                          ::= prefixID | base | sparqlPrefix |
                                              sparqlBase
[4]    prefixID                           ::= '@prefix' PNAME_NS IRIREF '.'
[5]    base                               ::= '@base' IRIREF '.'
[5s]   sparqlPrefix                       ::= "PREFIX" PNAME_NS IRIREF
[6s]   sparqlBase                         ::= "BASE" IRIREF
[6]    triples                            ::= subject predicateObjectList |
                                              blankNodePropertyList
                                              predicateObjectList?
[7]    predicateObjectList                ::= verb objectList (';' (verb
                                              objectList)?)*
[8]    objectList                         ::= object (',' object)*
[9]    verb                               ::= predicate | 'a'
[10]   subject                            ::= iri | blank
[11]   predicate                          ::= iri
[12]   object                             ::= iri | blank | blankNodePropertyList
                                              | literal
[13]   literal                            ::= RDFLiteral | NumericLiteral |
                                              BooleanLiteral
[14]   blank                              ::= BlankNode | collection
[15]   blankNodePropertyList              ::= '[' predicateObjectList ']'
[16]   collection                         ::= '(' object* ')'
[17]   NumericLiteral                     ::= INTEGER | DECIMAL | DOUBLE
[128s] RDFLiteral                         ::= String (LANGTAG | '^^' iri)?
[133s] BooleanLiteral                     ::= 'true' | 'false'
[18]   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

```
[19]   IRIREF                             ::= '<' ([^#x00-#x20<>"{}|^`\] |
                                              UCHAR)* '>'
[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]+)*
[20]   INTEGER                            ::= [+-]? [0-9]+
[21]   DECIMAL                            ::= [+-]? ([0-9]* '.' [0-9]+)
[22]   DOUBLE                             ::= [+-]? ([0-9]+ '.' [0-9]* EXPONENT |
                                              '.' [0-9]+ EXPONENT | [0-9]+
                                              EXPONENT)
[154s] EXPONENT                           ::= [eE] [+-]? [0-9]+
[23]   STRING_LITERAL_QUOTE               ::= '"' ([^#x22#x5C#xA#xD] | ECHAR |
                                              UCHAR)* '"'
[24]   STRING_LITERAL_SINGLE_QUOTE        ::= "'" ([^#x27#x5C#xA#xD] | ECHAR |
                                              UCHAR)* "'"
[25]   STRING_LITERAL_LONG_SINGLE_QUOTE   ::= "'''" (("'" | "''")? ([^'\] | ECHAR
                                              | UCHAR))* "'''"
[26]   STRING_LITERAL_LONG_QUOTE          ::= '"""' (('"' | '""')? ([^"\] | ECHAR
                                              | UCHAR))* '"""'
[27]   UCHAR                              ::= '\u' HEX HEX HEX HEX | '\U' HEX HEX
                                              HEX HEX HEX HEX HEX HEX
[159s] ECHAR                              ::= '\' [tbnrf"'\]
[160s] NIL                                ::= '(' WS* ')'
[161s] WS                                 ::= #x20 | #x9 | #xD | #xA
[162s] ANON                               ::= '[' WS* ']'
```

```
[163s] 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]
[164s] PN_CHARS_U    ::= PN_CHARS_BASE | '_'
[166s] PN_CHARS      ::= PN_CHARS_U | '-' | [0-9] | #00B7 |
                         [#0300-#036F] | [#203F-#2040]
[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  ::= '\' ('_' | '~' | '.' | '-' | '!' |
                         '$' | '&' | "'" | '(' | ')' | '*' |
                         '+' | ',' | ';' | '=' | '/' | '?' |
                         '#' | '@' | '%')
```

5. Parsing

The RDF Concepts and Abstract Syntax [RDF11-CONCEPTS] specification 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 4.5 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.

5.1 Parser State

Parsing TriG requires a state of six 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`.
- RDF_Term `curGraph` — The `curGraph` is bound to the label of the graph that is the destination of triples produced in parsing. When undefined, triples are destined for the default graph.

5.2 RDF Term Constructors

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

production	type	procedure
IRIREF	IRI	The characters between "<"

production	type	procedure
		and ">" are taken, with the numeric escape sequences unescaped, to form the unicode string of the IRI. Relative IRI resolution is performed per section 4.3 IRI References.
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.

production	type	procedure
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 `iri`, depending on which rule matched the input. If the `LANGTAG` rule matched, the datatype is `rdf:langString` and the language tag is `LANGTAG`. If neither a language tag nor a datatype IRI is provided, the literal has a datatype of `xsd:string`.
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.

5.3 RDF Triples Construction

A TriG document defines an RDF Dataset composed of one default graph and zero or more named graphs. Each graph is composed of a set of RDF triples.

5.3.1 Output Graph

The state `curGraph` is initially unset. It records the label of the graph for triples produced during parsing. If undefined, the default graph is used.

The rule `labelOrSubject` sets both `curGraph` and `curSubject` (only one of these will be used).

The following grammar production clauses set `curGraph` to be undefined, indicating the default graph:

- The grammar production clause `wrappedGraph` in rule `block`.
- The grammar production in rule `triples2`.

The grammar production `labelOrSubject predicateObjectList '.'` unsets `curGraph` before handling `predicateObjectLists` in rule `triplesOrGraph`.

5.3.2 Triple Output

Each RDF triple produced is added to `curGraph`, or the default graph if `curGraph` is not set at that point in the parsing process.

The `subject` production sets the `curSubject`. The `verb` production sets the `curPredicate`.

Triples are produced at the following points in the parsing process and each RDF triple produced is added to the graph identified by `curGraph`.

5.3.2.1 Triple Production

Each object `N` in the document produces an RDF triple: `curSubject curPredicate N`.

5.3.2.2 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`.

5.3.2.3 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.

6. Acknowledgements

This section is non-normative.

The editors gratefully acknowledge the work of Chris Bizer and Richard Cyganiak in creating the original TriG specification. Valuable contributions to this version were made by Gregg Kellogg, Eric Prud'hommeaux and Sandro Hawke.

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

A. Differences from Previous TriG

This section is non-normative.

This section describes the main differences between TriG, as defined in this document, and earlier forms.

- Syntax is aligned to the Turtle [TURTLE] recommendation for RDF terms.
- Graph labels can be blank nodes.
- The default graph, or sections of the default graph, do not need to be enclosed in `{ ... }`.
- No support for optional `=` graph naming operator or optional "." after each graph.
- Graph labels do not have to be unique within a TriG document. Reusing a graph label causes all the triples for that graph to be included in the resulting graph. Sections with the same label are combined by set union.
- Keywords `BASE`, `PREFIX` as in [TURTLE].
- The optional `GRAPH` keyword is allowed to aid SPARQL alignment.

B. Media Type Registration

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 TriG is "application/trig".

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

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

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

Type name:
application
Subtype name:
trig
Required parameters:
None
Optional parameters:
None
Encoding considerations:
The syntax of TriG 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:
TriG 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.
TriG 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 TriG documents. Security/privacy protocols must be imposed which reflect the sensitivity of the embedded information.
TriG can express data which is presented to the user, for example, RDF Schema labels. Application rendering strings retrieved from untrusted TriG 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.
TriG uses IRIs as term identifiers. Applications interpreting data expressed in TriG 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 TriG 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):
TriG documents may have the strings 'prefix' or 'base' (case independent) near the beginning of the document.

File extension(s):
".trig"

Base URI:
The TriG base directive can change the current base URI for relative IRIrefs in the 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 TriG specification is the product of the RDF WG. The W3C reserves change control over this specifications.

C. Changes since the last publication of this document

Error in grammar productions [24] and [25] fixed.

D. References

D.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

[TURTLE]
Eric Prud'hommeaux, Gavin Carothers. *RDF 1.1 Turtle: Terse RDF Triple Language.* W3C Recommendation, 25 February 2014. URL: http://www.w3.org/TR/2014/REC-turtle-20140225/. The latest edition is available at http://www.w3.org/TR/turtle/

[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

D.2 Informative references

[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/