RDF 1.1 XML Syntax

W3C Recommendation 25 February 2014

This version:
http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/
Latest published version:
http://www.w3.org/TR/rdf-syntax-grammar/
Test suite:
http://www.w3.org/TR/2014/NOTE-rdf11-testcases-20140225/
Previous version:
http://www.w3.org/TR/2014/PER-rdf-syntax-grammar-20140109/
Editors:
Fabien Gandon, INRIA
Guus Schreiber, VU University Amsterdam
Previous Editors:
Dave Beckett

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

This document is also available in this non-normative format: diff w.r.t. 2004 Recommendation

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

Abstract

This document defines an XML syntax for RDF called RDF/XML in terms of Namespaces in XML, the XML Information Set and XML Base.

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 an edited version of the 2004 RDF XML Syntax Specification Recommendation. The purpose of this revision is to make this document available as part of the RDF 1.1 document set. Changes are limited to revised references, terminology updates, and adaptations to the introduction. The technical content of the document is unchanged, except for the fact that the datatype XMLLiiteral is marked as non-normative in RDF 1.1. The (non-normative) algorithm for parsing XMLLiteral (Sec. 7.2.17) has been updated to be in line with the current state of XML technology. Details of the changes are listed in the Changes section. Since the edits to this document do not invalidate previous implementations the Director decided no new implementation report was required.

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.

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 the XML [XML10] syntax for RDF graphs.

This document revises the original RDF/XML grammar [RDFMS] in terms of XML Information Set [XML-INFOSET] information items which moves away from the rather low-level details of XML, such as particular forms of empty elements. This allows the grammar to be more precisely recorded and the mapping from the XML syntax to the RDF Graph more clearly shown. The mapping to the RDF graph is done by emitting statements in the N-Triples [N-TRIPLES] format.

This document is part of the suite of RDF 1.1 documents. Other documents in this suite are:

- A document describing the basic concepts underlying RDF, as well as abstract syntax ("RDF Concepts and Abstract Syntax") [RDF11-CONCEPTS]
- A document describing the formal model-theoretic semantics of RDF ("RDF Semantics") [RDF11-MT]
- Specifications of concrete syntaxes for RDF:
  - Turtle [TURTLE] and TriG [TRIG]
  - JSON-LD [JSON-LD] (JSON based)
  - RDFa [RDFA-PRIMER] (for HTML embedding)
  - N-Triples and N-Quads (line-based exchange formats)
- A document describing RDF Schema [RDF11-SCHEMA], which provides a data-modeling vocabulary for RDF data.

For a longer introduction to the RDF/XML syntax with a historical perspective, see "RDF: Understanding the Striped RDF/XML Syntax" [STRIPEDRDF].

2. An XML Syntax for RDF

This section introduces the RDF/XML syntax, describes how it encodes RDF graphs and explains this with examples. If there is any conflict between this informal description and the formal description of the syntax and grammar in sections 6 Syntax Data Model and 7 RDF/XML Grammar, the latter two sections take precedence.

2.1 Introduction

The RDF Concepts and Abstract Syntax document [RDF11-CONCEPTS] defines the RDF Graph data model and the RDF Graph abstract syntax. Along with the RDF Semantics [RDF11-MT] this provides an abstract syntax with a formal semantics for it. The RDF graph has *nodes* and labeled directed *arcs* that link pairs of nodes and this is represented as a set of RDF triples where each triple contains a *subject node*, *predicate* and *object node*. Nodes are IRIs, literals, or blank nodes. Blank nodes may be given a document-local identifier called a blank node identifier. Predicates are IRIs and can be interpreted as either a relationship between the two nodes or as defining an attribute value (object node) for some subject node.

In order to encode the graph in XML, the nodes and predicates have to be represented in XML terms — element names, attribute names, element contents and attribute values. RDF/XML uses XML QNames as defined in Namespaces in XML [XML-NAMES] to represent IRIs. All QNames have a namespace name which is an IRI and a short local name. In addition, QNames can either have a short prefix or be declared with the default namespace declaration and have none (but still have a namespace name)

The IRI represented by a QName is determined by appending the local name part of the QName after the namespace name (IRI) part of the QName. This is used to shorten the IRI of all predicates and some nodes. IRIs identifying subject and object nodes can also be stored as XML attribute values. RDF literals which can only be object nodes, become either XML element text content or XML attribute values.

A graph can be considered a collection of paths of the form node, predicate arc, node, predicate arc, node, predicate arc, ... node which cover the entire graph. In RDF/XML these turn into sequences of elements inside elements which alternate between elements for nodes and predicate arcs. This has been called a series of node/arc stripes. The node at the start of the sequence turns into the outermost element, the next predicate arc turns into a child element, and so on. The stripes generally start at the top of an RDF/XML document and always begin with nodes.

Several RDF/XML examples are given in the following sections building up to complete RDF/XML documents. Example 7 is the first complete RDF/XML document.

2.2 Node Elements and Property Elements

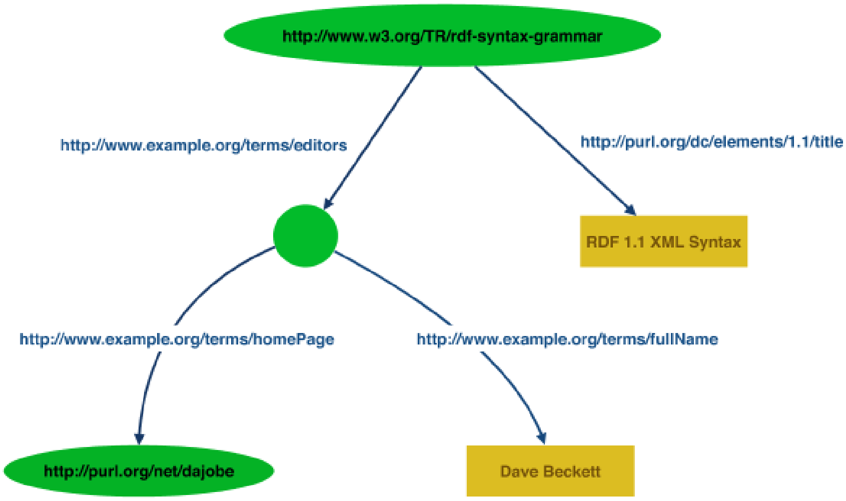

Fig. 1 Graph for RDF/XML Example (SVG version)

An RDF graph is given in Figure 1 where the nodes are represented as ovals and contain their IRIs where they have them, all the predicate arcs are labeled with IRIs and string literals nodes have been written in rectangles.

If we follow one node, predicate arc ... , node path through the graph shown in Figure 2:

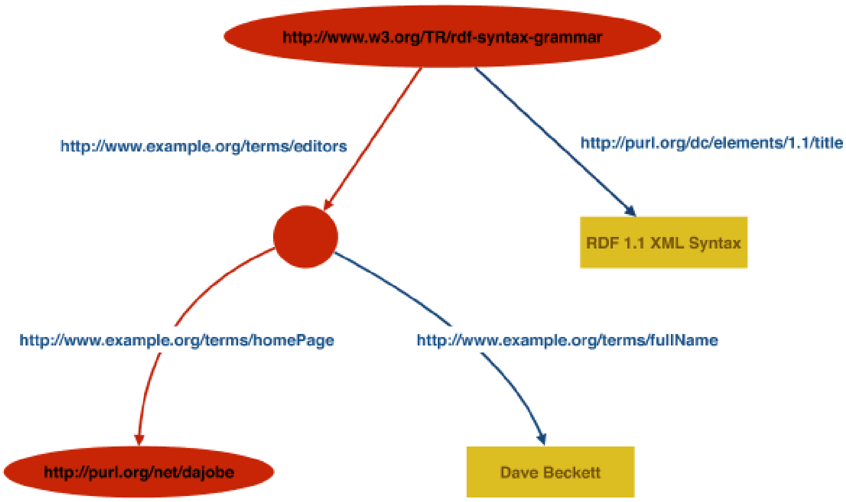

Fig. 2 One Path Through the Graph (SVG version)

The left hand side of the Figure 2 graph corresponds to the node/predicate arc stripes:

1. Node with IRI `http://www.w3.org/TR/rdf-syntax-grammar`
2. Predicate Arc labeled with IRI `http://example.org/terms/editor`
3. Node with no IRI
4. Predicate Arc labeled with IRI `http://example.org/terms/homePage`
5. Node with IRI `http://purl.org/net/dajobe/`

In RDF/XML, the sequence of 5 nodes and predicate arcs on the left hand side of Figure 2 corresponds to the usage of five XML elements of two types, for the graph nodes and predicate arcs. These are conventionally called *node elements* and *property elements* respectively. In the striping shown in Example 1, `rdf:Description` is the node element (used three times for the three nodes) and `ex:editor` and `ex:homePage` are the two property elements.

EXAMPLE 1

Striped RDF/XML (nodes and predicate arcs)

```
<rdf:Description>
  <ex:editor>
    <rdf:Description>
      <ex:homePage>
        <rdf:Description>
        </rdf:Description>
      </ex:homePage>
    </rdf:Description>
  </ex:editor>
</rdf:Description>
```

The Figure 2 graph consists of some nodes that are IRIs (and others that are not) and this can be added to the RDF/XML using the `rdf:about` attribute on node elements to give the result in Example 2:

EXAMPLE 2

Node Elements with IRIs added

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:homePage>
        <rdf:Description rdf:about="http://purl.org/net/dajobe/">
        </rdf:Description>
      </ex:homePage>
    </rdf:Description>
  </ex:editor>
</rdf:Description>
```

Adding the other two paths through the Figure 1 graph to the RDF/XML in Example 2 gives the result in Example 3 (this example fails to show that the blank node is shared between the two paths, see 2.10):

EXAMPLE 3

Complete description of all graph paths

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:homePage>
        <rdf:Description rdf:about="http://purl.org/net/dajobe/">
        </rdf:Description>
      </ex:homePage>
    </rdf:Description>
  </ex:editor>
</rdf:Description>

<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:fullName>Dave Beckett</ex:fullName>
    </rdf:Description>
  </ex:editor>
</rdf:Description>

<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <dc:title>RDF 1.1 XML Syntax</dc:title>
</rdf:Description>
```

2.3 Multiple Property Elements

There are several abbreviations that can be used to make common uses easier to write down. In particular, it is common that a subject node in the RDF graph has multiple outgoing predicate arcs. RDF/XML provides an abbreviation for the corresponding syntax when a node element about a resource has multiple property elements. This can be abbreviated by using multiple child property elements inside the node element describing the subject node.

Taking Example 3, there are two node elements that can take multiple property elements. The subject node with IRI `http://www.w3.org/TR/rdf-syntax-grammar` has property elements `ex:editor` and `ex:title` and the node element for the blank node can take `ex:homePage` and `ex:fullName`. This abbreviation gives the result shown in Example 4 (this example does show that there is a single blank node):

EXAMPLE 4

Using multiple property elements on a node element

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:homePage>
        <rdf:Description rdf:about="http://purl.org/net/dajobe/">
        </rdf:Description>
      </ex:homePage>
      <ex:fullName>Dave Beckett</ex:fullName>
    </rdf:Description>
  </ex:editor>
  <dc:title>RDF 1.1 XML Syntax</dc:title>
</rdf:Description>
```

2.4 Empty Property Elements

When a predicate arc in an RDF graph points to an object node which has no further predicate arcs, which appears in RDF/XML as an empty node element `<rdf:Description rdf:about="..."> </rdf:Description>` (or `<rdf:Description rdf:about="..." />`) this form can be shortened. This is done by using the IRI of the object node as the value of an XML attribute `rdf:resource` on the containing property element and making the property element empty.

In this example, the property element `ex:homePage` contains an empty node element with the IRI `http://purl.org/net/dajobe/`. This can be replaced with the empty property element form giving the result shown in Example 5:

EXAMPLE 5

Empty property elements

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:homePage rdf:resource="http://purl.org/net/dajobe/"/>
      <ex:fullName>Dave Beckett</ex:fullName>
    </rdf:Description>
  </ex:editor>
  <dc:title>RDF 1.1 XML Syntax</dc:title>
</rdf:Description>
```

2.5 Property Attributes

When a property element's content is string literal, it may be possible to use it as an XML attribute on the containing node element. This can be done for multiple properties on the same node element only if the property element name is not repeated (required by XML — attribute names are unique on an XML element) and any in-scope `xml:lang` on the property element's string literal (if any) are the same (see Section 2.7) This abbreviation is known as a *Property Attribute* and can be applied to any node element.

This abbreviation can also be used when the property element is `rdf:type` and it has an `rdf:resource` attribute the value of which is interpreted as a IRI object node.

In Example 5:, there are two property elements with string literal content, the `dc:title` and `ex:fullName` property elements. These can be replaced with property attributes giving the result shown in Example 6:

EXAMPLE 6

Replacing property elements with string literal content into property attributes

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar"
           dc:title="RDF 1.1 XML Syntax">
  <ex:editor>
    <rdf:Description ex:fullName="Dave Beckett">
      <ex:homePage rdf:resource="http://purl.org/net/dajobe/"/>
    </rdf:Description>
  </ex:editor>
</rdf:Description>
```

2.6 Completing the Document: Document Element and XML Declaration

To create a complete RDF/XML document, the serialization of the graph into XML is usually contained inside an `rdf:RDF` XML element which becomes the top-level XML document element. Conventionally the `rdf:RDF` element is also used to declare the XML namespaces that are used, although that is not required. When there is only one top-level node element inside `rdf:RDF`, the `rdf:RDF` can be omitted although any XML namespaces must still be declared.

The XML specification also permits an XML declaration at the top of the document with the XML version and possibly the XML content encoding. This is optional but recommended.

Completing the RDF/XML could be done for any of the correct complete graph examples from Example 4 onwards but taking the smallest Example 6 and adding the final components, gives a complete RDF/XML representation of the original Figure 1 graph in Example 7:

EXAMPLE 7

Complete RDF/XML description of Figure 1 graph (example07.rdf, output example07.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:ex="http://example.org/stuff/1.0/">

  <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar"
             dc:title="RDF1.1 XML Syntax">
    <ex:editor>
      <rdf:Description ex:fullName="Dave Beckett">
        <ex:homePage rdf:resource="http://purl.org/net/dajobe/" />
      </rdf:Description>
    </ex:editor>
  </rdf:Description>

</rdf:RDF>
```

It is possible to omit `rdf:RDF` in Example 7 above since there is only one `rdf:Description` inside `rdf:RDF` but this is not shown here.

2.7 Languages: `xml:lang`

RDF/XML permits the use of the `xml:lang` attribute as defined by 2.12 Language Identification of XML 1.0 [XML10] to allow the identification of content language. The `xml:lang` attribute can be used on any node element or property element to indicate that the included content is in the given language. Typed literals which includes XML literals are not affected by this attribute. The most specific in-scope language present (if any) is applied to property element string literal content or property attribute values. The `xml:lang=""` form indicates the absence of a language identifier.

Some examples of marking content languages for RDF properties are shown in Example 8:

EXAMPLE 8

Complete example of `xml:lang`
(example08.rdf, output example08.nt)

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:dc="http://purl.org/dc/elements/1.1/">

  <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
    <dc:title>RDF 1.1 XML Syntax</dc:title>
    <dc:title xml:lang="en">RDF 1.1 XML Syntax</dc:title>
    <dc:title xml:lang="en-US">RDF 1.1 XML Syntax</dc:title>
  </rdf:Description>

  <rdf:Description rdf:about="http://example.org/buecher/baum" xml:lang="de">
    <dc:title>Der Baum</dc:title>
    <dc:description>Das Buch ist außergewöhnlich</dc:description>
    <dc:title xml:lang="en">The Tree</dc:title>
  </rdf:Description>

</rdf:RDF>
```

2.8 XML Literals: `rdf:parseType="Literal"`

This section is non-normative.

RDF allows XML literals [RDF11-CONCEPTS] to be given as the object node of a predicate. These are written in RDF/XML as content of a property element (not a property attribute) and indicated using the `rdf:parseType="Literal"` attribute on the containing property element.

An example of writing an XML literal is given in Example 9 where there is a single RDF triple with the subject node IRI `http://example.org/item01`, the predicate IRI `http://example.org/stuff/1.0/prop` (from `ex:prop`) and the object node with XML literal content beginning `a:Box`.

EXAMPLE 9

Complete example of `rdf:parseType="Literal"`
(example09.rdf, output example09.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:ex="http://example.org/stuff/1.0/">

  <rdf:Description rdf:about="http://example.org/item01">
    <ex:prop rdf:parseType="Literal" xmlns:a="http://example.org/a#">
      <a:Box required="true">
        <a:widget size="10" />
        <a:grommit id="23" />
      </a:Box>
    </ex:prop>
  </rdf:Description>

</rdf:RDF>
```

2.9 Typed Literals: `rdf:datatype`

RDF allows typed literals to be given as the object node of a predicate. Typed literals consist of a literal string and a datatype IRI. These are written in RDF/XML using the same syntax for literal string nodes in the property element form (not property attribute) but with an additional `rdf:datatype="`*datatypeURI*`"` attribute on the property element. Any IRI can be used in the attribute.

An example of an RDF typed literal is given in Example 10 where there is a single RDF triple with the subject node IRI `http://example.org/item01`, the predicate IRI `http://example.org/stuff/1.0/size` (from `ex:size`) and the object node with the typed literal ("123", `http://www.w3.org/2001/XMLSchema#int`) to be interpreted as an XML Schema [XMLSCHEMA-2] datatype `int`.

EXAMPLE 10

Complete example of `rdf:datatype`
(example10.rdf, output example10.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
            xmlns:ex="http://example.org/stuff/1.0/">

  <rdf:Description rdf:about="http://example.org/item01">
    <ex:size rdf:datatype="http://www.w3.org/2001/XMLSchema#int">123</ex:size>
  </rdf:Description>

</rdf:RDF>
```

2.10 Identifying Blank Nodes: `rdf:nodeID`

Blank nodes in the RDF graph are distinct but have no IRI identifier. It is sometimes required that the same graph blank node is referred to in the RDF/XML in multiple places, such as at the subject and object of several RDF triples. In this case, a blank node identifier can be given to the blank node for identifying it in the document. Blank node identifiers in RDF/XML are scoped to the containing XML Information Set document information item. A blank node identifier is used on a node element to replace `rdf:about="`*IRI*`"` or on a property element to replace `rdf:resource="`*IRI*`"` with `rdf:nodeID="`*blank node identifier*`"` in both cases.

Taking Example 7 and explicitly giving a blank node identifier of `abc` to the blank node in it gives the result shown in Example 11. The second `rdf:Description` property element is about the blank node.

EXAMPLE 11

Complete RDF/XML description of graph using `rdf:nodeID` identifying the blank node (example11.rdf, output example11.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:ex="http://example.org/stuff/1.0/">

  <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar"
             dc:title="RDF 1.1 XML Syntax">
    <ex:editor rdf:nodeID="abc"/>
  </rdf:Description>

  <rdf:Description rdf:nodeID="abc" ex:fullName="Dave Beckett">
    <ex:homePage rdf:resource="http://purl.org/net/dajobe/"/>
  </rdf:Description>

</rdf:RDF>
```

2.11 Omitting Blank Nodes: `rdf:parseType="Resource"`

Blank nodes (not IRI nodes) in RDF graphs can be written in a form that allows the `<rdf:Description> </rdf:Description>` pair to be omitted. The omission is done by putting an `rdf:parseType="Resource"` attribute on the containing property element that turns the property element into a property-and-node element, which can itself have both property elements and property attributes. Property attributes and the `rdf:nodeID` attribute are not permitted on property-and-node elements.

Taking the earlier Example 7, the contents of the `ex:editor` property element could be alternatively done in this fashion to give the form shown in Example 12:

EXAMPLE 12

Complete example using `rdf:parseType="Resource"` (example12.rdf, output: example12.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:ex="http://example.org/stuff/1.0/">
  <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar"
                   dc:title="RDF 1.1 XML Syntax">
    <ex:editor rdf:parseType="Resource">
      <ex:fullName>Dave Beckett</ex:fullName>
      <ex:homePage rdf:resource="http://purl.org/net/dajobe/"/>
    </ex:editor>
  </rdf:Description>
</rdf:RDF>
```

2.12 Omitting Nodes: Property Attributes on an empty Property Element

If all of the property elements on a blank node element have string literal values with the same in-scope `xml:lang` value (if present) and each of these property elements appears at most once and there is at most one `rdf:type` property element with a IRI object node, these can be abbreviated by moving them to be property attributes on the containing property element which is made an empty element.

Taking the earlier Example 5, the `ex:editor` property element contains a blank node element with two property elements `ex:fullname` and `ex:homePage`. `ex:homePage` is not suitable here since it does not have a string literal value, so it is being *ignored* for the purposes of this example. The abbreviated form removes the `ex:fullName` property element and adds a new property attribute `ex:fullName` with the string literal value of the deleted property element to the `ex:editor` property element. The blank node element becomes implicit in the now empty `ex:editor` property element. The result is shown in Example 13.

EXAMPLE 13

Complete example of property attributes on an empty property element (example13.rdf, output example13.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:ex="http://example.org/stuff/1.0/">

  <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar"
            dc:title="RDF 1.1 XML Syntax">
    <ex:editor ex:fullName="Dave Beckett" />
            <!-- Note the ex:homePage property has been ignored for this example -->
  </rdf:Description>

</rdf:RDF>
```

2.13 Typed Node Elements

It is common for RDF graphs to have `rdf:type` predicates from subject nodes. These are conventionally called *typed nodes* in the graph, or *typed node elements* in the RDF/XML. RDF/XML allows this triple to be expressed more concisely. by replacing the `rdf:Description` node element name with the namespaced-element corresponding to the IRI of the value of the type relationship. There may, of course, be multiple `rdf:type` predicates but only one can be used in this way, the others must remain as property elements or property attributes.

The typed node elements are commonly used in RDF/XML with the built-in classes in the RDF vocabulary: `rdf:Seq`, `rdf:Bag`, `rdf:Alt`, `rdf:Statement`, `rdf:Property` and `rdf:List`.

For example, the RDF/XML in Example 14 could be written as shown in Example 15.

EXAMPLE 14

Complete example with `rdf:type` (example14.rdf, output example14.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:ex="http://example.org/stuff/1.0/">

  <rdf:Description rdf:about="http://example.org/thing">
    <rdf:type rdf:resource="http://example.org/stuff/1.0/Document"/>
    <dc:title>A marvelous thing</dc:title>
  </rdf:Description>
</rdf:RDF>
```

EXAMPLE 15

Complete example using a typed node element to replace an `rdf:type` (example15.rdf, output example15.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:ex="http://example.org/stuff/1.0/">

  <ex:Document rdf:about="http://example.org/thing">
    <dc:title>A marvelous thing</dc:title>
  </ex:Document>

</rdf:RDF>
```

2.14 Abbreviating URIs: `rdf:ID` and `xml:base`

RDF/XML allows further abbreviating IRIs in XML attributes in two ways. The XML Infoset provides a base URI attribute `xml:base` that sets the base URI for resolving relative IRIs, otherwise the base URI is that of the document. The base URI applies to all RDF/XML attributes that deal with IRIs which are `rdf:about`, `rdf:resource`, `rdf:ID` and `rdf:datatype`.

The `rdf:ID` attribute on a node element (not property element, that has another meaning) can be used instead of `rdf:about` and gives a relative IRI equivalent to `#` concatenated with the `rdf:ID` attribute value. So for example if `rdf:ID="name"`, that would be equivalent to `rdf:about="#name"`. `rdf:ID` provides an additional check since the same *name* can only appear once in the scope of an `xml:base` value (or document, if none is given), so is useful for defining a set of distinct, related terms relative to the same IRI.

Both forms require a base URI to be known, either from an in-scope `xml:base` or from the URI of the RDF/XML document.

Example 16 shows abbreviating the node IRI of `http://example.org/here/#snack` using an `xml:base` of `http://example.org/here/` and an `rdf:ID` on the `rdf:Description` node element. The object node of the `ex:prop` predicate is an absolute IRI resolved from the `rdf:resource` XML attribute value using the in-scope base URI to give the IRI `http://example.org/here/fruit/apple`.

EXAMPLE 16

Complete example using rdf:ID and xml:base for shortening URIs (example16.rdf, output example16.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:ex="http://example.org/stuff/1.0/"
            xml:base="http://example.org/here/">

  <rdf:Description rdf:ID="snack">
    <ex:prop rdf:resource="fruit/apple"/>
  </rdf:Description>

</rdf:RDF>
```

2.15 Container Membership Property Elements: rdf:li and rdf:_n

RDF has a set of container membership properties and corresponding property elements that are mostly used with instances of the rdf:Seq, rdf:Bag and rdf:Alt classes which may be written as typed node elements. The list properties are rdf:_1, rdf:_2 etc. and can be written as property elements or property attributes as shown in Example 17. There is an rdf:li special property element that is equivalent to rdf:_1, rdf:_2 in order, explained in detail in section 7.4. The mapping to the container membership properties is always done in the order that the rdf:li special property elements appear in XML — the document order is significant. The equivalent RDF/XML to Example 17 written in this form is shown in Example 18.

EXAMPLE 17

Complex example using RDF list properties (example17.rdf, output example17.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

  <rdf:Seq rdf:about="http://example.org/favourite-fruit">
    <rdf:_1 rdf:resource="http://example.org/banana"/>
    <rdf:_2 rdf:resource="http://example.org/apple"/>
    <rdf:_3 rdf:resource="http://example.org/pear"/>
  </rdf:Seq>

</rdf:RDF>
```

EXAMPLE 18

Complete example using rdf:li property element for list properties (example18.rdf, output example18.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

  <rdf:Seq rdf:about="http://example.org/favourite-fruit">
    <rdf:li rdf:resource="http://example.org/banana"/>
    <rdf:li rdf:resource="http://example.org/apple"/>
    <rdf:li rdf:resource="http://example.org/pear"/>
  </rdf:Seq>

</rdf:RDF>
```

2.16 Collections: rdf:parseType="Collection"

RDF/XML allows an rdf:parseType="Collection" attribute on a property element to let it contain multiple node elements. These contained node elements give the set of subject nodes of the collection. This syntax form corresponds to a set of triples connecting the collection of subject nodes, the exact triples generated are described in detail in Section 7.2.19 Production parseTypeCollectionPropertyElt. The collection construction is always done in the order that the node elements appear in the XML document. Whether the order of the collection of nodes is significant is an application issue and not defined here.

Example 19 shows a collection of three nodes elements at the end of the ex:hasFruit property element using this form.

EXAMPLE 19

Complete example of a RDF collection of nodes using rdf:parseType="Collection" (example19.rdf, output example19.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:ex="http://example.org/stuff/1.0/">

  <rdf:Description rdf:about="http://example.org/basket">
    <ex:hasFruit rdf:parseType="Collection">
      <rdf:Description rdf:about="http://example.org/banana"/>
      <rdf:Description rdf:about="http://example.org/apple"/>
      <rdf:Description rdf:about="http://example.org/pear"/>
    </ex:hasFruit>
  </rdf:Description>

</rdf:RDF>
```

2.17 Reifying Statements: `rdf:ID`

The `rdf:ID` attribute can be used on a property element to reify the triple that it generates (See section 7.3 Reification Rules for the full details). The identifier for the triple should be constructed as a IRI made from the relative IRI `#` concatenated with the `rdf:ID` attribute value, resolved against the in-scope base URI. So for example if `rdf:ID="triple"`, that would be equivalent to the IRI formed from relative IRI `#triple` against the base URI. Each (`rdf:ID` attribute value, base URI) pair has to be unique in an RDF/XML document, see constraint-id.

Example 20 shows a `rdf:ID` being used to reify a triple made from the `ex:prop` property element giving the reified triple the IRI `http://example.org/triples/#triple1`.

EXAMPLE 20

Complete example of `rdf:ID` reifying a property element (example20.rdf, output example20.nt)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:ex="http://example.org/stuff/1.0/"
            xml:base="http://example.org/triples/">
  <rdf:Description rdf:about="http://example.org/">
    <ex:prop rdf:ID="triple1">blah</ex:prop>
  </rdf:Description>

</rdf:RDF>
```

3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

All use of string without further qualification refers to a Unicode [UNICODE] character string; a sequence of characters represented by a code point in Unicode.

4. RDF MIME Type, File Extension and Macintosh File Type

The Internet media type / MIME type for RDF/XML is `application/rdf+xml` — RFC 3023 [RFC3023], section 8.18.

NOTE

(Informative): For the state of the MIME type registration, consult IANA MIME Media Types [IANA-MEDIA-TYPES]

It is recommended that RDF/XML files have the extension `".rdf"` (all lowercase) on all platforms.

It is recommended that RDF/XML files stored on Macintosh HFS file systems be given a file type of `"rdf "` (all lowercase, with a space character as the fourth letter).

5. Global Issues

5.1 The RDF Namespace and Vocabulary

The ***RDF namespace IRI*** (or namespace name) is `http://www.w3.org/1999/02/22-rdf-syntax-ns#` and is typically used in XML with the prefix `rdf` although other prefix strings may be used. The ***RDF Vocabulary*** is identified by this namespace name and consists of the following names only:

Syntax names — not concepts

`RDF Description ID about parseType resource li nodeID datatype`

Class names

`Seq Bag Alt Statement Property XMLLiteral List`

Property names

`subject predicate object type value first rest _`*n*
where *n* is a decimal integer greater than zero with no leading zeros.

Resource names

`nil`

Any other names are not defined and SHOULD generate a warning when encountered, but should otherwise behave normally.

Within RDF/XML documents it is not permitted to use XML namespaces whose namespace name is the ·RDF namespace IRI· concatenated with additional characters.

Throughout this document the terminology `rdf:`*name* will be used to indicate *name* is from the RDF vocabulary and it has a IRI of the concatenation of the ·RDF namespace IRI· and *name*. For example, `rdf:type` has the IRI `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`

5.2 Identifiers

The RDF Concepts document [RDF11-CONCEPTS] defines the three types of RDF data that can act as node and/or predicate:

IRI

IRIs can act as node (both subject and object) and as predicate.

IRIs can be either:

- given as XML attribute values interpreted as relative IRIs that are resolved against the in-scope base URI as described in section 5.3 to give absolute IRIs
- transformed from XML namespace-qualified element and attribute names (QNames)
- transformed from `rdf:ID` attribute values.

Within RDF/XML, XML QNames are transformed into IRIs by appending the XML local name to the namespace name (IRI). For example, if the XML namespace prefix `foo` has namespace name (IRI) `http://example.org/somewhere/` then the QName `foo:bar` would correspond to the IRI `http://example.org/somewhere/bar`. Note that this restricts which IRIs can be made and the same IRI can be given in multiple ways.

The `rdf:ID` values are transformed into IRIs by appending the attribute value to the result of appending "#" to the in-scope base URI which is defined in Section 5.3 Resolving IRIs

Literal

Literals can only act as object nodes.

Literals always have a datatype. Language-tagged strings get the datatype `rdf:langString`. When there is no language tag or datatype specified the literal is assumed to have the datatype `xsd:string`.

Blank Node

Blank nodes can act as subject node and as object node.

Blank nodes have distinct identity in the RDF graph. When the graph is written in a syntax such as RDF/XML, these blank nodes may need graph-local identifiers and a syntax in order to preserve this distinction. These local identifiers are called blank node identifiers and are used in RDF/XML as values of the `rdf:nodeID` attribute with the syntax given in Production nodeIdAttr. Blank node identifiers in RDF/XML are scoped to the XML Information Set document information item.

If no blank node identifier is given explicitly as an `rdf:nodeID` attribute value then one will need to be generated (using generated-blank-node-id, see section 6.3.3). Such generated blank node identifiers must not clash with any blank node identifiers derived from `rdf:nodeID` attribute values. This can be implemented by any method that preserves the distinct identity of all the blank nodes in the graph, that is, the same blank node identifier is not given for different blank nodes. One possible method would be to add a constant prefix to all the `rdf:nodeID` attribute values and ensure no generated blank node identifiers ever used that prefix. Another would be to map all `rdf:nodeID` attribute values to new generated blank node identifiers and perform that mapping on all such values in the RDF/XML document.

5.3 Resolving IRIs

RDF/XML supports XML Base [XMLBASE] which defines a ·base-uri· accessor for each ·root event· and ·element event·. Relative IRIs are resolved into IRIs according to the algorithm specified in [XMLBASE] (and RFC 2396). These specifications do not specify an algorithm for resolving a fragment identifier alone, such as `#foo`, or the empty string `""` into an IRI. In RDF/XML, a fragment identifier is transformed into an IRI by appending the fragment identifier to the in-scope base URI. The empty string is transformed into an IRI by substituting the in-scope base URI.

> NOTE
>
> **Test:** indicated by:
> test001.rdf and test001.nt
> test004.rdf and test004.nt
> test008.rdf and test008.nt

An empty same document reference "" resolves against the URI part of the base URI; any fragment part is ignored. See Uniform Resource Identifiers (URI) [RFC3986].

> NOTE
>
> **Test:** Indicated by test013.rdf and test013.nt

> NOTE

Implementation Note (Informative): When using a hierarchical base URI that has no path component (/), it must be added before using as a base URI for resolving.

NOTE

Test: Indicated by test011.rdf and test011.nt

5.4 Constraints

constraint-id

Each application of production idAttr matches an attribute. The pair formed by the ·string-value· accessor of the matched attribute and the ·base-uri· accessor of the matched attribute is unique within a single RDF/XML document.

The syntax of the names must match the rdf-id production.

NOTE

Test: Indicated by test014.rdf and test014.nt

5.5 Conformance

Definition:
An ***RDF Document*** is a serialization of an RDF Graph into a concrete syntax.
Definition:
An ***RDF/XML Document*** is an RDF Document written in the XML syntax for RDF as defined in this document.
Conformance:
An RDF/XML Document is a ***conforming RDF/XML document*** if it adheres to the specification defined in this document.

6. Syntax Data Model

This document specifies the syntax of RDF/XML as a grammar on an alphabet of symbols. The symbols are called *events* in the style of the XPATH Information Set Mapping. A sequence of events is normally derived from an XML document, in which case they are in document order as defined below in Section 6.2 Information Set Mapping. The sequence these events form are intended to be similar to the sequence of events produced by the [SAX] XML API from the same XML document. Sequences of events may be checked against the grammar to determine whether they are or are not syntactically well-formed RDF/XML.

The grammar productions may include actions which fire when the production is recognized. Taken together these actions define a transformation from any syntactically well-formed RDF/XML sequence of events into an RDF graph represented in the N-Triples [N-TRIPLES] language.

The model given here illustrates one way to create a representation of an RDF Graph from an RDF/XML document. It does not mandate any implementation method — any other method that results in a representation of the same RDF Graph may be used.

In particular:

- This specification permits any representation of an RDF graph; in particular, it does not require the use of N-Triples [N-TRIPLES].
- This specification does not require the use of [XPATH] or [SAX]
- This specification places no constraints on the order in which software transforming RDF/XML into a representation of a graph, constructs the representation of the graph.
- Software transforming RDF/XML into a representation of a graph MAY eliminate duplicate predicate arcs.

The syntax does not support non-well-formed XML documents, nor documents that otherwise do not have an XML Information Set; for example, that do not conform to Namespaces in XML [XML-NAMES].

The Infoset requires support for XML Base [XMLBASE]. RDF/XML uses the information item property [base URI], discussed in section 5.3

This specification requires an XML Information Set [XML-INFOSET] which supports at least the following information items and properties for RDF/XML:

document information item
[document element], [children], [base URI]
element information item
[local name], [namespace name], [children], [attributes], [parent], [base URI]
attribute information item
[local name], [namespace name], [normalized value]
character information item
[character code]

There is no mapping of the following items to data model events:

- processing instruction information item

- unexpanded entity reference information item
- comment information item
- document type declaration information item
- unparsed entity information item
- notation information item
- namespace information item

Other information items and properties have no mapping to syntax data model events.

Element information items with reserved XML Names (See Name in XML 1.0) are not mapped to data model element events. These are all those with property [prefix] beginning with `xml` (case independent comparison) and all those with [prefix] property having no value and which have [local name] beginning with `xml` (case independent comparison).

All information items contained inside XML elements matching the parseTypeLiteralPropertyElt production form XML literals and do not follow this mapping. See parseTypeLiteralPropertyElt for further information.

This section is intended to satisfy the requirements for Conformance in the [XML-INFOSET] specification. It specifies the information items and properties that are needed to implement this specification.

6.1 Events

There are nine types of event defined in the following subsections. Most events are constructed from an Infoset information item (except for IRI, blank node, plain literal and typed literal). The effect of an event constructor is to create a new event with a unique identity, distinct from all other events. Events have accessor operations on them and most have the *string-value* accessor that may be a static value or computed.

6.1.1 Root Event

Constructed from a document information item and takes the following accessors and values.

document-element
: Set to the value of document information item property [document-element].

children
: Set to the value of document information item property [children].

base-uri
: Set to the value of document information item property [base URI].

language
: Set to the empty string.

6.1.2 Element Event

Constructed from an element information item and takes the following accessors and values:

local-name
: Set to the value of element information item property [local name].

namespace-name
: Set to the value of element information item property [namespace name].

children
: Set to the value of element information item property [children].

parent
: Set to the value of element information item property [parent].

base-uri
: Set to the value of element information item property [base URI].

attributes
: Made from the value of element information item property [attributes] which is a set of attribute information items.

  If this set contains an attribute information item `xml:lang` ([namespace name] property with the value "http://www.w3.org/XML/1998/namespace" and [local name] property value "lang") it is removed from the set of attribute information items and the ·language· accessor is set to the [normalized-value] property of the attribute information item.

  All remaining reserved XML Names (see Name in XML 1.0) are now removed from the set. These are, all attribute information items in the set with property [prefix] beginning with `xml` (case independent comparison) and all attribute information items with [prefix] property having no value and which have [local name] beginning with `xml` (case independent comparison) are removed. Note that the [base URI] accessor is computed by XML Base before any `xml:base` attribute information item is deleted.

  The remaining set of attribute information items are then used to construct a new set of Attribute Events which is assigned as the value of this accessor.

URI
: Set to the string value of the concatenation of the value of the namespace-name accessor and the value of the local-name accessor.

URI-string-value
: The value is the concatenation of the following in this order "<", the escaped value of the ·URI· accessor and ">".

  The escaping of the ·URI· accessor uses the N-Triples escapes for IRIs [[N_TRIPLES]].

li-counter

Set to the integer value 1.

language
: Set from the ·attributes· as described above. If no value is given from the attributes, the value is set to the value of the language accessor on the parent event (either a Root Event or an Element Event), which may be the empty string.

subject
: Has no initial value. Takes a value that is an Identifier event. This accessor is used on elements that deal with one node in the RDF graph, this generally being the subject of a statement.

6.1.3 End Element Event

Has no accessors. Marks the end of the containing element in the sequence.

6.1.4 Attribute Event

Constructed from an attribute information item and takes the following accessors and values:

local-name
: Set to the value of attribute information item property [local name].

namespace-name
: Set to the value of attribute information item property [namespace name].

string-value
: Set to the value of the attribute information item property [normalized value] as specified by [XML10] (if an attribute whose normalized value is a zero-length string, then the string-value is also a zero-length string).

URI

If ·namespace-name· is present, set to a string value of the concatenation of the value of the ·namespace-name· accessor and the value of the ·local-name· accessor. Otherwise if ·local-name· is `ID`, `about`, `resource`, `parseType` or `type`, set to a string value of the concatenation of the ·RDF namespace IRI· and the value of the ·local-name· accessor. Other non-namespaced ·local-name· accessor values are forbidden.

The support for a limited set of non-namespaced names is REQUIRED and intended to allow RDF/XML documents specified in [RDFMS] to remain valid; new documents SHOULD NOT use these unqualified attributes and applications MAY choose to warn when the unqualified form is seen in a document.

The construction of IRIs from XML attributes can generate the same IRIs from different XML attributes. This can cause ambiguity in the grammar when matching attribute events (such as when `rdf:about` and `about` XML attributes are both present). Documents that have this are illegal.

URI-string-value

The value is the concatenation of the following in this order "<", the escaped value of the ·URI· accessor and ">".

The escaping of the ·URI· accessor uses the N-Triples escapes for IRIs [N-TRIPLES].

6.1.5 Text Event

Constructed from a sequence of one or more consecutive character information items. Has the single accessor:

string-value
: Set to the value of the string made from concatenating the [character code] property of each of the character information items.

6.1.6 IRI Event

An event for a IRIs which has the following accessors:

identifier
: Takes a string value used as an IRI.

string-value

The value is the concatenation of "<", the escaped value of the ·identifier· accessor and ">"

The escaping of the ·identifier· accessor value uses the N-Triples escapes for IRIs [N-TRIPLES].

These events are constructed by giving a value for the ·identifier· accessor.

For further information on identifiers in the RDF graph, see section 5.2.

6.1.7 Blank Node Identifier Event

An event for a blank node identifier which has the following accessors:

identifier
: Takes a string value.

string-value
: The value is a function of the value of the ·identifier· accessor. The value begins with "_:" and the entire value MUST match the N-Triples BLANK_NODE_LABELD production. The function MUST preserve distinct blank node identity as discussed in in section 5.2 Identifiers.

These events are constructed by giving a value for the ·identifier· accessor.

For further information on identifiers in the RDF graph, see section 5.2.

6.1.8 Plain Literal Event

NOTE

RDF/XML plain literals are in RDF 1.1 treated as syntactic sugar for a literal with datatype `xsd:string` (in case no language tag is present) or as a literal with datatype `rdf:langString` (in case a language tag is present). The mapping to N-Triples as defined in this subsection is not affected by this change.

An event for a plain literal which can have the following accessors:

literal-value
: Takes a string value.

literal-language
: Takes a string value used as a language tag in an RDF plain literal.

string-value
: The value is calculated from the other accessors as follows.

  If ·literal-language· is the empty string then the value is the concatenation of """ (1 double quote), the escaped value of the ·literal-value· accessor and """ (1 double quote).

  Otherwise the value is the concatenation of """ (1 double quote), the escaped value of the ·literal-value· accessor ""@" (1 double quote and a '@'), and the value of the ·literal-language· accessor.

  The escaping of the ·literal-value· accessor value uses the N-Triples escapes for strings as described in [N-TRIPLES] for escaping certain characters such as ".

These events are constructed by giving values for the ·literal-value· and ·literal-language· accessors.

NOTE

Interoperability Note (Informative): Literals beginning with a Unicode combining character are allowed however they may cause interoperability problems. See [CHARMOD] for further information.

6.1.9 Typed Literal Event

An event for a typed literal which can have the following accessors:

literal-value
: Takes a string value.

literal-datatype
: Takes a string value used as an IRI.

string-value
: The value is the concatenation of the following in this order """ (1 double quote), the escaped value of the ·literal-value· accessor, """ (1 double quote), "^^<", the escaped value of the ·literal-datatype· accessor and ">".

  The escaping of the ·literal-value· accessor value uses the N-Triples escapes for strings [N-TRIPLES] for escaping certain characters such as ". The escaping of the ·literal-datatype· accessor value must use the N-Triples escapes for IRI [N-TRIPLES].

These events are constructed by giving values for the ·literal-value· and ·literal-datatype· accessors.

NOTE

Interoperability Note (Informative): Literals beginning with a Unicode combining character are allowed however they may cause interoperability problems. See [CHARMOD] for further information.

NOTE

Implementation Note (Informative): In XML Schema (part 1) [XMLSCHEMA-1], white space normalization occurs during validation according to the value of the whiteSpace facet. The syntax mapping used in this document occurs after this, so the whiteSpace facet formally has no further effect.

6.2 Information Set Mapping

To transform the Infoset into the sequence of events in *document order*, each information item is transformed as described above to generate a tree of events with accessors and values. Each element event is then replaced as described below to turn the tree of events into a sequence in document order.

1. The original element event
2. The value of the children accessor recursively transformed, a possibly empty ordered list of events.
3. An end element event

6.3 Grammar Notation

The following notation is used to describe matching the sequence of data model events as given in Section 6 and the actions to perform for the matches. The RDF/XML grammar is defined in terms of mapping from these matched data model events to triples, using notation of the form:

> number *event-type event-content*
>
> *action*...
>
> `N-Triples`

where the *event-content* is an expression matching *event-types* (as defined in Section 6.1), using notation given in the following sections. The number is used for reference purposes. The grammar *action* may include generating new triples to the graph, written in N-Triples [N-TRIPLES] format.

The following sections describe the general notation used and that for event matching and actions.

6.3.1 Grammar General Notation

Notation	Meaning
event.accessor	The value of an event accessor.
`rdf:`*X*	A URI as defined in section 5.1.
"ABC"	A string of characters A, B, C in order.

6.3.2 Grammar Event Matching Notation

Notation	Meaning
A == B	Event accessor A matches expression B.
A != B	A is not equal to B.
A | B | ...	The A, B, ... terms are alternatives.
A - B	The terms in A excluding all the terms in B.
anyURI.	Any URI.
anyString.	Any string.
list(item1, item2, ...); list()	An ordered list of events. An empty list.
set(item1, item2, ...); set()	An unordered set of events. An empty set.
*	Zero or more of preceding term.
?	Zero or one of preceding term.
+	One or more of preceding term.
root(acc1 == value1, acc2 == value2, ...)	Match a Root Event with accessors.
start-element(acc1 == value1, acc2 == value2, ...) *children* end-element()	Match a sequence of Element Event with accessors, a possibly empty list of events as element content and an End Element Event.
attribute(acc1 == value1, acc2 == value2, ...)	Match an Attribute Event with accessors.
text()	Match a Text Event.

6.3.3 Grammar Action Notation

Notation	Meaning
A := B	Assigns A the value B.
concat(A, B, ..)	A string created by concatenating the terms in order.
resolve(*e*, *s*)	A string created by interpreting string *s* as a relative IRI to the ·base-uri· accessor of 6.1.2 Element Event *e* as defined in Section 5.3 Resolving URIs. The resulting string represents an IRI.
generated-blank-node-id()	A string value for a new distinct generated blank node identifier as defined in section 5.2 Identifiers.
event.accessor := *value*	Sets an event accessor to the given value.
uri(identifier := value)	Create a new URI Reference Event.
bnodeid(identifier := value)	Create a new Blank Node Identifier Event. See also section 5.2 Identifiers.
literal(literal-value := string,	Create a new Plain Literal Event.

literal-language := language, ...)	
typed-literal(literal-value := string, ...)	Create a new Typed Literal Event.

7. RDF/XML Grammar

7.1 Grammar summary

7.2.2 coreSyntaxTerms	`rdf:RDF` | `rdf:ID` | `rdf:about` | `rdf:parseType` | `rdf:resource` | `rdf:nodeID` | `rdf:datatype`
7.2.3 syntaxTerms	coreSyntaxTerms | `rdf:Description` | `rdf:li`
7.2.4 oldTerms	`rdf:aboutEach` | `rdf:aboutEachPrefix` | `rdf:bagID`
7.2.5 nodeElementURIs	anyURI - (coreSyntaxTerms | `rdf:li` | oldTerms)
7.2.6 propertyElementURIs	anyURI - (coreSyntaxTerms | `rdf:Description` | oldTerms)
7.2.7 propertyAttributeURIs	anyURI - (coreSyntaxTerms | `rdf:Description` | `rdf:li` | oldTerms)
7.2.8 doc	root(document-element == RDF, children == list(RDF))
7.2.9 RDF	start-element(URI == `rdf:RDF`, attributes == set()) nodeElementList end-element()
7.2.10 nodeElementList	ws* (nodeElement ws*)*
7.2.11 nodeElement	start-element(URI == nodeElementURIs attributes == set((idAttr | nodeIdAttr | aboutAttr)?, propertyAttr*)) propertyEltList end-element()
7.2.12 ws	A text event matching white space defined by XML [XML10] definition *White Space* Rule [3] S in section Common Syntactic Constructs
7.2.13 propertyEltList	ws* (propertyElt ws*) *
7.2.14 propertyElt	resourcePropertyElt | literalPropertyElt | parseTypeLiteralPropertyElt | parseTypeResourcePropertyElt | parseTypeCollectionPropertyElt | parseTypeOtherPropertyElt | emptyPropertyElt
7.2.15 resourcePropertyElt	start-element(URI == propertyElementURIs), attributes == set(idAttr?)) ws* nodeElement ws* end-element()
7.2.16 literalPropertyElt	start-element(URI == propertyElementURIs), attributes == set(idAttr?, datatypeAttr?)) text() end-element()
7.2.17 parseTypeLiteralPropertyElt	start-element(URI == propertyElementURIs), attributes == set(idAttr?, parseLiteral)) literal end-element()
7.2.18 parseTypeResourcePropertyElt	start-element(URI == propertyElementURIs), attributes == set(idAttr?, parseResource)) propertyEltList end-element()
7.2.19 parseTypeCollectionPropertyElt	start-element(URI == propertyElementURIs), attributes == set(idAttr?, parseCollection)) nodeElementList end-element()
7.2.20 parseTypeOtherPropertyElt	start-element(URI == propertyElementURIs), attributes == set(idAttr?, parseOther)) propertyEltList end-element()
7.2.21 emptyPropertyElt	start-element(URI == propertyElementURIs), attributes == set(idAttr?, (resourceAttr | nodeIdAttr | datatypeAttr)?, propertyAttr*)) end-element()
7.2.22 idAttr	attribute(URI == `rdf:ID`, string-value == rdf-id)
7.2.23 nodeIdAttr	attribute(URI == `rdf:nodeID`, string-value == rdf-id)
7.2.24 aboutAttr	attribute(URI == `rdf:about`, string-value == URI-reference)
7.2.25 propertyAttr	attribute(URI == propertyAttributeURIs, string-value == anyString)
7.2.26 resourceAttr	attribute(URI == `rdf:resource`, string-value == URI-reference)
7.2.27 datatypeAttr	attribute(URI == `rdf:datatype`, string-value == URI-reference)
7.2.28 parseLiteral	attribute(URI == `rdf:parseType`, string-value == "Literal")
7.2.29 parseResource	attribute(URI == `rdf:parseType`, string-value == "Resource")
7.2.30 parseCollection	attribute(URI == `rdf:parseType`, string-value == "Collection")
7.2.31 parseOther	attribute(URI == `rdf:parseType`, string-value == anyString - ("Resource" | "Literal" | "Collection"))
7.2.32 URI-reference	An IRI.
7.2.33 literal	Any XML element content that is allowed according to [XML10] definition *Content of Elements* Rule [43] content. in section 3.1 Start-Tags, End-Tags, and Empty-Element Tags
7.2.34 rdf-id	An attribute ·string-value· matching any legal [XML-NAMES] token NCName

7.2 Grammar Productions

7.2.1 Grammar start

If the RDF/XML is a standalone XML document (identified by presentation as an application/rdf+xml RDF MIME type object, or by some other means) then the grammar may start with production doc or production nodeElement.

If the content is known to be RDF/XML by context, such as when RDF/XML is embedded inside other XML content, then the grammar can either start at Element Event RDF (only when an element is legal at that point in the XML) or at production nodeElementList (only when element content is legal, since this is a list of elements). For such embedded RDF/XML, the ·base-uri· value on the outermost element must be initialized from the containing XML since no Root Event will be available. Note that if such embedding occurs, the grammar may be entered several times but no state is expected to be preserved.

7.2.2 Production coreSyntaxTerms

`rdf:RDF` | `rdf:ID` | `rdf:about` | `rdf:parseType` | `rdf:resource` | `rdf:nodeID` | `rdf:datatype`

A subset of the syntax terms from the RDF vocabulary in section 5.1 which are used in RDF/XML.

7.2.3 Production syntaxTerms

coreSyntaxTerms | `rdf:Description` | `rdf:li`

All the syntax terms from the RDF vocabulary in section 5.1 which are used in RDF/XML.

7.2.4 Production oldTerms

`rdf:aboutEach` | `rdf:aboutEachPrefix` | `rdf:bagID`

These are the names from the RDF vocabulary that have been withdrawn from the language. See the resolutions of Issue rdfms-aboutEach-on-object, Issue rdfms-abouteachprefix and Last Call Issue timbl-01 for further information.

> NOTE
>
> **Error Test:** Indicated by error001.rdf and error002.rdf

7.2.5 Production nodeElementURIs

anyURI - (coreSyntaxTerms | `rdf:li` | oldTerms)

The IRIs that are allowed on node elements.

7.2.6 Production propertyElementURIs

anyURI - (coreSyntaxTerms | `rdf:Description` | oldTerms)

The URIs that are allowed on property elements.

7.2.7 Production propertyAttributeURIs

anyURI - (coreSyntaxTerms | `rdf:Description` | `rdf:li` | oldTerms)

The IRIs that are allowed on property attributes.

7.2.8 Production doc

root(document-element == RDF,
  children == list(RDF))

7.2.9 Production RDF

start-element(URI == `rdf:RDF`,
  attributes == set())
nodeElementList
end-element()

7.2.10 Production nodeElementList

ws* (nodeElement ws*)*

7.2.11 Production nodeElement

start-element(URI == nodeElementURIs
  attributes == set((idAttr | nodeIdAttr | aboutAttr)?, propertyAttr*))
propertyEltList
end-element()

For node element *e*, the processing of some of the attributes has to be done before other work such as dealing with children events or other attributes. These can be processed in any order:

- If there is an attribute *a* with *a*.URI == `rdf:ID`, then *e*.subject := uri(identifier := resolve(*e*, concat("#", *a*.string-value))).
- If there is an attribute *a* with *a*.URI == `rdf:nodeID`, then *e*.subject := bnodeid(identifier:=*a*.string-value).

- If there is an attribute *a* with *a*.URI == `rdf:about` then *e*.subject := uri(identifier := resolve(*e*, *a*.string-value)).

If *e*.subject is empty, then *e*.subject := bnodeid(identifier := generated-blank-node-id()).

The following can then be performed in any order:

- If *e*.URI != `rdf:Description` then the following statement is added to the graph:

  ```
  e.subject.string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> e.URI-string-value .
  ```

- If there is an attribute *a* in propertyAttr with *a*.URI == `rdf:type` then *u*:=uri(identifier:=resolve(*e*, *a*.string-value)) and the following triple is added to the graph:

  ```
  e.subject.string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> u.string-value .
  ```

- For each attribute *a* matching propertyAttr (and not `rdf:type`), the Unicode string *a*.string-value SHOULD be in Normal Form C [NFC], *o* := literal(literal-value := *a*.string-value, literal-language := *e*.language) and the following statement is added to the graph:

  ```
  e.subject.string-value a.URI-string-value o.string-value .
  ```

- Handle the propertyEltList children events in document order.

7.2.12 Production ws

A text event matching white space defined by [XML10] definition *White Space* Rule [3] S in section Common Syntactic Constructs

7.2.13 Production propertyEltList

ws* (propertyElt ws*) *

7.2.14 Production propertyElt

resourcePropertyElt | literalPropertyElt | parseTypeLiteralPropertyElt | parseTypeResourcePropertyElt | parseTypeCollectionPropertyElt | parseTypeOtherPropertyElt | emptyPropertyElt

If element *e* has *e*.URI = `rdf:li` then apply the list expansion rules on element *e*.parent in section 7.4 to give a new URI *u* and *e*.URI := *u*.

The action of this production must be done before the actions of any sub-matches (resourcePropertyElt ... emptyPropertyElt). Alternatively the result must be equivalent to as if it this action was performed first, such as performing as the first action of all of the sub-matches.

7.2.15 Production resourcePropertyElt

start-element(URI == propertyElementURIs),
  attributes == set(idAttr?))
ws* nodeElement ws*
end-element()

For element *e*, and the single contained nodeElement *n*, first *n* must be processed using production nodeElement. Then the following statement is added to the graph:

```
e.parent.subject.string-value e.URI-string-value n.subject.string-value .
```

If the `rdf:ID` attribute *a* is given, the above statement is reified with *i* := uri(identifier := resolve(*e*, concat("#", *a*.string-value))) using the reification rules in section 7.3 and *e*.subject := *i*

7.2.16 Production literalPropertyElt

start-element(URI == propertyElementURIs),
  attributes == set(idAttr?, datatypeAttr?))
text()
end-element()

Note that the empty literal case is defined in production emptyPropertyElt.

For element *e*, and the text event *t*. The Unicode string *t*.string-value SHOULD be in Normal Form C [NFC]. If the `rdf:datatype` attribute *d* is given then *o* := typed-literal(literal-value := *t*.string-value, literal-datatype := *d*.string-value) otherwise *o* := literal(literal-value := *t*.string-value, literal-language := *e*.language) and the following statement is added to the graph:

```
e.parent.subject.string-value e.URI-string-value o.string-value .
```

If the `rdf:ID` attribute *a* is given, the above statement is reified with *i* := uri(identifier := resolve(*e*, concat("#", *a*.string-value))) using the reification rules in section 7.3 and *e*.subject := *i*.

7.2.17 Production parseTypeLiteralPropertyElt

This section is non-normative.

start-element(URI == propertyElementURIs),
  attributes == set(idAttr?, parseLiteral))
literal
end-element()

For element *e* and the literal *l* that is the `rdf:parseType="Literal"` content. *l* is not transformed by the syntax data model mapping into events (as noted in section 6 Syntax Data Model) but remains an XML Infoset of XML Information items.

l is transformed into the lexical form of an XML literal in the RDF graph *x* (a Unicode string) by the following algorithm. This does not mandate any implementation method — any other method that gives the same result may be used.

1. Use *l* to construct an XPath sequence [XPATH-DATAMODEL-30].
2. Apply http://www.w3.org/TR/xpath-functions-30/#func-serialize [XPATH-FUNCTIONS-30] to this sequence to give an xsd:string *x*.
3. The Unicode string *x* is used as the lexical form of *l*
4. This Unicode string *x* SHOULD be in NFC Normal Form C [NFC]

Then *o* := typed-literal(literal-value := *x*, literal-datatype := `http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral`) and the following statement is added to the graph:

```
e.parent.subject.string-value e.URI-string-value o.string-value .
```

> NOTE
>
> **Test:** Empty literal case indicated by test009.rdf and test009.nt

If the `rdf:ID` attribute *a* is given, the above statement is reified with *i* := uri(identifier := resolve(*e*, concat("#", *a*.string-value))) using the reification rules in section 7.3 and *e*.subject := *i*.

7.2.18 Production parseTypeResourcePropertyElt

start-element(URI == propertyElementURIs),
  attributes == set(idAttr?, parseResource))
propertyEltList
end-element()

For element *e* with possibly empty element content *c*.

n := bnodeid(identifier := generated-blank-node-id()).

Add the following statement to the graph:

```
e.parent.subject.string-value e.URI-string-value n.string-value .
```

> NOTE
>
> **Test:** Indicated by test004.rdf and test004.nt

If the `rdf:ID` attribute *a* is given, the statement above is reified with *i* := uri(identifier := resolve(*e*, concat("#", *a*.string-value))) using the reification rules in section 7.3 and *e*.subject := *i*.

If the element content *c* is not empty, then use event *n* to create a new sequence of events as follows:

start-element(URI := `rdf:Description`,
  subject := *n*,
  attributes := set())
c
end-element()

Then process the resulting sequence using production nodeElement.

7.2.19 Production parseTypeCollectionPropertyElt

start-element(URI == propertyElementURIs),
  attributes == set(idAttr?, parseCollection))
nodeElementList
end-element()

For element event *e* with possibly empty nodeElementList *l*. Set *s*:=list().

For each element event *f* in *l*, *n* := bnodeid(identifier := generated-blank-node-id()) and append *n* to *s* to give a sequence of events.

If *s* is not empty, *n* is the first event identifier in *s* and the following statement is added to the graph:

```
e.parent.subject.string-value e.URI-string-value n.string-value .
```

otherwise the following statement is added to the graph:

`e.parent.subject.string-value e.URI-string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#nil> .`

If the `rdf:ID` attribute *a* is given, either of the the above statements is reified with *i* := uri(identifier := resolve(*e*, concat("#", *a*.string-value))) using the reification rules in section 7.3.

If *s* is empty, no further work is performed.

For each event *n* in *s* and the corresponding element event *f* in *l*, the following statement is added to the graph:

`n.string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#first> f.string-value .`

For each consecutive and overlapping pair of events (*n*, *o*) in *s*, the following statement is added to the graph:

`n.string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest> o.string-value .`

If *s* is not empty, *n* is the last event identifier in *s*, the following statement is added to the graph:

`n.string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest> <http://www.w3.org/1999/02/22-rdf-syntax-ns#nil> .`

7.2.20 Production parseTypeOtherPropertyElt

start-element(URI == propertyElementURIs),
    attributes == set(idAttr?, parseOther))
propertyEltList
end-element()

All `rdf:parseType` attribute values other than the strings "Resource", "Literal" or "Collection" are treated as if the value was "Literal". This production matches and acts as if production parseTypeLiteralPropertyElt was matched. No extra triples are generated for other `rdf:parseType` values.

7.2.21 Production emptyPropertyElt

start-element(URI == propertyElementURIs),
    attributes == set(idAttr?, (resourceAttr | nodeIdAttr | datatypeAttr)?, propertyAttr*))
end-element()

- If there are no attributes **or** only the optional `rdf:ID` attribute *i* then *o* := literal(literal-value:="", literal-language := *e*.language) and the following statement is added to the graph:

  `e.parent.subject.string-value e.URI-string-value o.string-value .`

  and then if *i* is given, the above statement is reified with uri(identifier := resolve(*e*, concat("#", *i*.string-value))) using the reification rules in section 7.3.

  > NOTE
  >
  > **Test:** Indicated by test002.rdf and test002.nt

  > NOTE
  >
  > **Test:** Indicated by test005.rdf and test005.nt

- Otherwise

  - If `rdf:resource` attribute *i* is present, then *r* := uri(identifier := resolve(*e*, *i*.string-value))
  - If `rdf:nodeID` attribute *i* is present, then *r* := bnodeid(identifier := *i*.string-value)
  - If neither, *r* := bnodeid(identifier := generated-blank-node-id())

  The following are done in any order:

  - For all propertyAttr attributes *a* (in any order)

    - If *a*.URI == `rdf:type` then *u*:=uri(identifier:=resolve(*e*, *a*.string-value)) and the following triple is added to the graph:

      `r.string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> u.string-value .`

    - Otherwise Unicode string *a*.string-value SHOULD be in Normal Form C [NFC], *o* := literal(literal-value := *a*.string-value, literal-language := *e*.language) and the following statement is added to the graph:

      `r.string-value a.URI-string-value o.string-value .`

    > NOTE
    >
    > **Test:** Indicated by test013.rdf and test013.nt

NOTE

Test: Indicated by test014.rdf and test014.nt

- Add the following statement to the graph:

  `e.parent.subject.string-value e.URI-string-value r.string-value .`

  and then if `rdf:ID` attribute *i* is given, the above statement is reified with uri(identifier := resolve(*e*, concat("#", *i*.string-value))) using the reification rules in section 7.3.

7.2.22 Production idAttr

attribute(URI == `rdf:ID`,
string-value == rdf-id)

Constraint:: constraint-id applies to the values of `rdf:ID` attributes

7.2.23 Production nodeIdAttr

attribute(URI == `rdf:nodeID`,
string-value == rdf-id)

7.2.24 Production aboutAttr

attribute(URI == `rdf:about`,
string-value == URI-reference)

7.2.25 Production propertyAttr

attribute(URI == propertyAttributeURIs,
string-value == anyString)

7.2.26 Production resourceAttr

attribute(URI == `rdf:resource`,
string-value == URI-reference)

7.2.27 Production datatypeAttr

attribute(URI == `rdf:datatype`,
string-value == URI-reference)

7.2.28 Production parseLiteral

attribute(URI == `rdf:parseType`,
string-value == "Literal")

7.2.29 Production parseResource

attribute(URI == `rdf:parseType`,
string-value == "Resource")

7.2.30 Production parseCollection

attribute(URI == `rdf:parseType`,
string-value == "Collection")

7.2.31 Production parseOther

attribute(URI == `rdf:parseType`,
string-value == anyString - ("Resource" | "Literal" | "Collection"))

7.2.32 Production IRI

An IRI.

7.2.33 Production literal

Any XML element content that is allowed according to XML definition *Content of Elements* Rule [43] content. in section 3.1 Start-Tags, End-Tags, and Empty-Element Tags

The string-value for the resulting event is discussed in section 7.2.17.

7.2.34 Production rdf-id

An attribute ·string-value· matching any legal [XML-NAMES] token NCName

7.3 Reification Rules

For the given IRI event *r* and the statement with terms *s*, *p* and *o* corresponding to the N-Triples:

```
s p o .
```

add the following statements to the graph:

```
r.string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#subject> s .
r.string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate> p .
r.string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#object> o .
r.string-value <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement> .
```

7.4 List Expansion Rules

For the given element *e*, create a new IRI *u* := concat("http://www.w3.org/1999/02/22-rdf-syntax-ns#_", *e*.li-counter), increment the *e*.li-counter property by 1 and return *u*.

8. Serializing an RDF Graph to RDF/XML

There are some RDF Graphs as defined in [RDF11-CONCEPTS]that cannot be serialized in RDF/XML. These are those that:

Use property names that cannot be turned into XML namespace-qualified names.
An XML namespace-qualified name (QName) has restrictions on the legal characters such that not all property URIs can be expressed as these names. It is recommended that implementors of RDF serializers, in order to break a URI into a namespace name and a local name, split it after the last XML non-NCName character, ensuring that the first character of the name is a Letter or '_'. If the URI ends in a non-NCName character then throw a "this graph cannot be serialized in RDF/XML" exception or error.

Use inappropriate reserved names as properties
For example, a property with the same URI as any of the syntaxTerms production.

Use the `rdf:HTML` datatype
This datatype as introduced in RDF 1.1 [RDF11-CONCEPTS].

> NOTE
>
> **Implementation Note (Informative):** When an RDF graph is serialized to RDF/XML and has an XML Schema Datatype (XSD), it SHOULD be written in a form that does not require whitespace processing. XSD support is NOT required by RDF or RDF/XML so this is optional.

9. Using RDF/XML with SVG

This section is non-normative.

There is a standardized approach for associating RDF compatible metadata with SVG — the metadata element which was explicitly designed for this purpose as defined in Section 21 Metadata of the Scalable Vector Graphics (SVG) 1.0 Specification [SVG10] and Section 21 Metadata of the Scalable Vector Graphics (SVG) 1.1 Specification [SVG11].

This document contains two example graphs in SVG with such embedded RDF/XML inside the metadata element: figure 1 and figure 2.

A. Acknowledgments

This section is non-normative.

Gavin Carothers provided the RDF 1.1 update for the Production parseTypeLiteralPropertyElt. Ivan Herman provided valuable comments and reworked Figs 1 and 2.

This specification is a product of extended deliberations by the members of the RDFcore Working Group and the RDF and RDF Schema Working Group.

The following people provided valuable contributions to the document:

- Dan Brickley, W3C/ILRT
- Jeremy Carroll, HP Labs, Bristol
- Graham Klyne, Nine by Nine
- Bijan Parsia, MIND Lab at University of Maryland at College Park

This document is a product of extended deliberations by the RDF Core working group, whose members have included: Art Barstow (W3C) Dave Beckett (ILRT), Dan Brickley (W3C/ILRT), Dan Connolly (W3C), Jeremy Carroll (Hewlett Packard), Ron Daniel (Interwoven Inc), Bill dehOra (InterX), Jos De Roo (AGFA), Jan Grant (ILRT), Graham Klyne (Clearswift and Nine by

Nine), Frank Manola (MITRE Corporation), Brian McBride (Hewlett Packard), Eric Miller (W3C), Stephen Petschulat (IBM), Patrick Stickler (Nokia), Aaron Swartz (HWG), Mike Dean (BBN Technologies / Verizon), R. V. Guha (Alpiri Inc), Pat Hayes (IHMC), Sergey Melnik (Stanford University), Martyn Horner (Profium Ltd).

This specification also draws upon an earlier RDF Model and Syntax document edited by Ora Lassilla and Ralph Swick, and RDF Schema edited by Dan Brickley and R. V. Guha. RDF and RDF Schema Working group members who contributed to this earlier work are: Nick Arnett (Verity), Tim Berners-Lee (W3C), Tim Bray (Textuality), Dan Brickley (ILRT / University of Bristol), Walter Chang (Adobe), Sailesh Chutani (Oracle), Dan Connolly (W3C), Ron Daniel (DATAFUSION), Charles Frankston (Microsoft), Patrick Gannon (CommerceNet), RV Guha (Epinions, previously of Netscape Communications), Tom Hill (Apple Computer), Arthur van Hoff (Marimba), Renato Iannella (DSTC), Sandeep Jain (Oracle), Kevin Jones, (InterMind), Emiko Kezuka (Digital Vision Laboratories), Joe Lapp (webMethods Inc.), Ora Lassila (Nokia Research Center), Andrew Layman (Microsoft), Ralph LeVan (OCLC), John McCarthy (Lawrence Berkeley National Laboratory), Chris McConnell (Microsoft), Murray Maloney (Grif), Michael Mealling (Network Solutions), Norbert Mikula (DataChannel), Eric Miller (OCLC), Jim Miller (W3C, emeritus), Frank Olken (Lawrence Berkeley National Laboratory), Jean Paoli (Microsoft), Sri Raghavan (Digital/Compaq), Lisa Rein (webMethods Inc.), Paul Resnick (University of Michigan), Bill Roberts (KnowledgeCite), Tsuyoshi Sakata (Digital Vision Laboratories), Bob Schloss (IBM), Leon Shklar (Pencom Web Works), David Singer (IBM), Wei (William) Song (SISU), Neel Sundaresan (IBM), Ralph Swick (W3C), Naohiko Uramoto (IBM), Charles Wicksteed (Reuters Ltd.), Misha Wolf (Reuters Ltd.), Lauren Wood (SoftQuad).

B. Changes since 2004 Recommendation

This section is non-normative.

Changes for RDF 1.1 Recommendation

- No changes.

Changes for RDF 1.1 Proposed Edited Recommendation:

1. Conversion to ReSpec.
2. RDF 2004 errata handling:
    1. Replaced hard-coded reference to XML and Unicode versions (background info)
    2. Corrected the resolve action with the signature resolve(e, s) (background info)
    3. Added parent accessor to element events (background info)
    4. Allow datatyped empty literals (background info)
    5. Removed ID and datatype exclusion on literal property (background info)
3. Adapted and shortened introduction to reflect RDF 1.1
4. Updated references to RDF 1.1 documents
5. Replaced "(RDF) URI reference" with "IRI"
6. Removed Section on embedding RDF/XML into HTML
7. Removed "Specification" from the title to bring it in line with other RDF 1.1 document titles
8. Updated references to other documents
9. Changed links in Sec. 2 examples from relative URI to absolute URI; same for RELAX schema in Appendix.
10. Added note to section on plain-literal event
11. Updated link to QName definition in XML-NAMES
12. Added diff with 2004 Recommendation
13. Sections concerning `rdf:XMLLiteral` (Sec. 2.8 and Sec. 7.2.17) marked as non-normative.
14. Adapted Production parseTypeLiteralPropertyElt to cater for the non-normative status of `rdf:XMLLiteral`.
15. Improved version of Figs. 1 and 2 (with same content)
16. Removed old changes section
17. Informative notes at start of Sec. 5.1 removed, as these have become irrelevant.
18. Added new datatype `rdf:HTML` to the list of things that cannot be serialized in RDF/XML.
19. Replaced the link to 2004 N-Triples `nodeID` production to the RDF 1.1 N-Triples `BLANK_NODE_LABEL` production.

C. Syntax Schemas

This section is non-normative.

This appendix contains XML schemas for validating RDF/XML forms. These are example schemas for information only and are not part of this specification.

C.1 RELAX NG Compact Schema

This section is non-normative.

This is an example schema in RELAX NG Compact (for ease of reading) for RDF/XML. Applications can also use the RELAX NG XML version. These formats are described in RELAX NG [RELAXNG] and RELAX NG Compact [RELAXNG-COMPACT].

> NOTE
>
> The RNGC schema has been updated to attempt to match the grammar but this has not been checked or used to validate RDF/XML.

```
#
# RELAX NG Compact Schema for RDF/XML Syntax
#
# This schema is for information only and NON-NORMATIVE
```

```
#
# It is based on one originally written by James Clark in
# http://lists.w3.org/Archives/Public/www-rdf-comments/2001JulSep/0248.html
# and updated with later changes.
#

namespace local = ""
namespace rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
datatypes xsd = "http://www.w3.org/2001/XMLSchema-datatypes"

start = doc

# I cannot seem to do this in RNGC so they are expanded in-line

# coreSyntaxTerms = rdf:RDF | rdf:ID | rdf:about | rdf:parseType | rdf:resource | rdf:nodeID | rdf:datatype
# syntaxTerms = coreSyntaxTerms | rdf:Description | rdf:li
# oldTerms    = rdf:aboutEach | rdf:aboutEachPrefix | rdf:bagID
# nodeElementURIs       = * - ( coreSyntaxTerms | rdf:li | oldTerms )
# propertyElementURIs   = * - ( coreSyntaxTerms | rdf:Description | oldTerms )
# propertyAttributeURIs = * - ( coreSyntaxTerms | rdf:Description | rdf:li | oldTerms )

# Also needed to allow rdf:li on all property element productions
# since we can't capture the rdf:li rewriting to rdf_<n> in relaxng

# Need to add these explicitly
xmllang = attribute xml:lang { text }
xmlbase = attribute xml:base { text }
# and to forbid every other xml:* attribute, element

doc =
  RDF | nodeElement

RDF =
  element rdf:RDF {
     xmllang?, xmlbase?, nodeElementList
}

nodeElementList =
  nodeElement*

  # Should be something like:
  #  ws* , (  nodeElement , ws* )*
  # but RELAXNG does this by default, ignoring whitespace separating tags.

nodeElement =
  element * - ( local:* | rdf:RDF | rdf:ID | rdf:about | rdf:parseType |
                rdf:resource | rdf:nodeID | rdf:datatype | rdf:li |
                rdf:aboutEach | rdf:aboutEachPrefix | rdf:bagID ) {
      (idAttr | nodeIdAttr | aboutAttr )?, xmllang?, xmlbase?, propertyAttr*, propertyEltList
  }

  # It is not possible to say "and not things
  # beginning with _ in the rdf: namespace" in RELAX NG.

ws =
  " "

  # Not used in this RELAX NG schema; but should be any legal XML
  # whitespace defined by http://www.w3.org/TR/2000/REC-xml-20001006#NT-S


propertyEltList =
  propertyElt*

  # Should be something like:
  #  ws* , ( propertyElt , ws* )*
  # but RELAXNG does this by default, ignoring whitespace separating tags.

propertyElt =
  resourcePropertyElt |
  literalPropertyElt |
  parseTypeLiteralPropertyElt |
  parseTypeResourcePropertyElt |
  parseTypeCollectionPropertyElt |
  parseTypeOtherPropertyElt |
  emptyPropertyElt

resourcePropertyElt =
  element * - ( local:* | rdf:RDF | rdf:ID | rdf:about | rdf:parseType |
                rdf:resource | rdf:nodeID | rdf:datatype |
                rdf:Description | rdf:aboutEach | rdf:aboutEachPrefix | rdf:bagID |
                xml:* ) {
      idAttr?, xmllang?, xmlbase?, nodeElement
  }

literalPropertyElt =
  element * - ( local:* | rdf:RDF | rdf:ID | rdf:about | rdf:parseType |
                rdf:resource | rdf:nodeID | rdf:datatype |
                rdf:Description | rdf:aboutEach | rdf:aboutEachPrefix | rdf:bagID |
                xml:* ) {
      idAttr? , datatypeAttr?, xmllang?, xmlbase?, text
  }

parseTypeLiteralPropertyElt =
  element * - ( local:* | rdf:RDF | rdf:ID | rdf:about | rdf:parseType |
                rdf:resource | rdf:nodeID | rdf:datatype |
```

```
               rdf:Description | rdf:aboutEach | rdf:aboutEachPrefix | rdf:bagID |
               xml:* ) {
      idAttr?, parseLiteral, xmllang?, xmlbase?, literal
  }

parseTypeResourcePropertyElt =
  element * - ( local:* | rdf:RDF | rdf:ID | rdf:about | rdf:parseType |
               rdf:resource | rdf:nodeID | rdf:datatype |
               rdf:Description | rdf:aboutEach | rdf:aboutEachPrefix | rdf:bagID |
               xml:* ) {
      idAttr?, parseResource, xmllang?, xmlbase?, propertyEltList
  }

parseTypeCollectionPropertyElt =
  element * - ( local:* | rdf:RDF | rdf:ID | rdf:about | rdf:parseType |
               rdf:resource | rdf:nodeID | rdf:datatype |
               rdf:Description | rdf:aboutEach | rdf:aboutEachPrefix | rdf:bagID |
               xml:* ) {
      idAttr?, xmllang?, xmlbase?, parseCollection, nodeElementList
  }

parseTypeOtherPropertyElt =
  element * - ( local:* | rdf:RDF | rdf:ID | rdf:about | rdf:parseType |
               rdf:resource | rdf:nodeID | rdf:datatype |
               rdf:Description | rdf:aboutEach | rdf:aboutEachPrefix | rdf:bagID |
               xml:* ) {
      idAttr?, xmllang?, xmlbase?, parseOther, any
  }

emptyPropertyElt =
   element * - ( local:* | rdf:RDF | rdf:ID | rdf:about | rdf:parseType |
                rdf:resource | rdf:nodeID | rdf:datatype |
                rdf:Description | rdf:aboutEach | rdf:aboutEachPrefix | rdf:bagID |
                xml:* ) {
       idAttr?, (resourceAttr | nodeIdAttr | datatypeAttr )?, xmllang?, xmlbase?, propertyAttr*
   }

idAttr =
  attribute rdf:ID {
      IDsymbol
  }

nodeIdAttr =
  attribute rdf:nodeID {
      IDsymbol
  }

aboutAttr =
  attribute rdf:about {
      URI-reference
  }

propertyAttr =
  attribute * - ( local:* | rdf:RDF | rdf:ID | rdf:about | rdf:parseType |
                 rdf:resource | rdf:nodeID | rdf:datatype | rdf:li |
                 rdf:Description | rdf:aboutEach |
         rdf:aboutEachPrefix | rdf:bagID |
                 xml:* ) {
      string
  }

resourceAttr =
  attribute rdf:resource {
      URI-reference
  }

datatypeAttr =
  attribute rdf:datatype {
      URI-reference
  }

parseLiteral =
  attribute rdf:parseType {
      "Literal"
  }

parseResource =
  attribute rdf:parseType {
      "Resource"
  }

parseCollection =
  attribute rdf:parseType {
      "Collection"
  }

parseOther =
  attribute rdf:parseType {
      text
  }

URI-reference =
  string

literal =
  any
```

```
IDsymbol =
  xsd:NMTOKEN

any =
  mixed { element * { attribute * { text }*, any }* }
```

D. References

D.1 Normative references

[JSON-LD]
Manu Sporny, Gregg Kellogg, Markus Lanthaler, Editors. *JSON-LD 1.0*. 16 January 2014. W3C Recommendation. URL: http://www.w3.org/TR/json-ld/

[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-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/

[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/

[RDF11-SCHEMA]
Dan Brickley, R. V. Guha. *RDF Schema 1.1*. W3C Recommendation, 25 February 2014. URL: http://www.w3.org/TR/2014/REC-rdf-schema-20140225/. The latest published version is available at http://www.w3.org/TR/rdf-schema/.

[RDFA-PRIMER]
Ivan Herman; Ben Adida; Manu Sporny; Mark Birbeck. *RDFa 1.1 Primer - Second Edition*. 22 August 2013. W3C Note. URL: http://www.w3.org/TR/rdfa-primer/

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

[TRIG]
Gavin Carothers, Andy Seaborne. *TriG: RDF Dataset Language*. W3C Recommendation, 25 February 2014. URL: http://www.w3.org/TR/2014/REC-trig-20140225/. The latest edition is available at http://www.w3.org/TR/trig/

[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/

[XML-INFOSET]
John Cowan; Richard Tobin. *XML Information Set (Second Edition)*. 4 February 2004. W3C Recommendation. URL: http://www.w3.org/TR/xml-infoset

[XML-NAMES]
Tim Bray; Dave Hollander; Andrew Layman; Richard Tobin; Henry Thompson et al. *Namespaces in XML 1.0 (Third Edition)*. 8 December 2009. W3C Recommendation. URL: http://www.w3.org/TR/xml-names

[XML10]
Tim Bray; Jean Paoli; Michael Sperberg-McQueen; Eve Maler; François Yergeau et al. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. 26 November 2008. W3C Recommendation. URL: http://www.w3.org/TR/xml

[XMLSCHEMA-2]
Paul V. Biron; Ashok Malhotra. *XML Schema Part 2: Datatypes Second Edition*. 28 October 2004. W3C Recommendation. URL: http://www.w3.org/TR/xmlschema-2/

D.2 Informative references

[CHARMOD]
Martin Dürst; François Yergeau; Richard Ishida; Misha Wolf; Tex Texin et al. *Character Model for the World Wide Web 1.0: Fundamentals*. 15 February 2005. W3C Recommendation. URL: http://www.w3.org/TR/charmod/

[IANA-MEDIA-TYPES]
MIME Media Types. The Internet Assigned Numbers Authority (IANA). The registration for application/rdf+xml is archived at http://www.w3.org/2001/sw/RDFCore/mediatype-registration.

[NFC]
M. Davis, Ken Whistler. *TR15, Unicode Normalization Forms.*. 17 September 2010, URL: http://www.unicode.org/reports/tr15/

[RDFMS]
Ora Lassila; Ralph R. Swick. *Resource Description Framework (RDF) Model and Syntax Specification*. 22 February 1999. W3C Recommendation. URL: http://www.w3.org/TR/1999/REC-rdf-syntax-19990222.

[RELAXNG]
James Clark and Murata Makoto, editors. *RELAX NG Specification*. OASIS Committee Specification, 3 December 2001. Latest version: http://www.oasis-open.org/committees/relax-ng/spec.html.

[RELAXNG-COMPACT]
James Clark, editor. *RELAX NG Compact Syntax*. OASIS Committee Specification, 21 November 2002. URI: http://www.oasis-open.org/committees/relax-ng/compact-20021121.html.

[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

[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

[SAX]
D. Megginson, et al. *SAX: The Simple API for XML*. May 1998. URL: http://www.megginson.com/downloads/SAX/

[STRIPEDRDF]
D. Brickley. *RDF: Understanding the Striped RDF/XML Syntax. W3C, 2001. URI: http://www.w3.org/2001/10/stripes/.*

[SVG10]
Jon Ferraiolo. *Scalable Vector Graphics (SVG) 1.0 Specification*. 4 September 2001. W3C Recommendation. URL: http://www.w3.org/TR/SVG/

[SVG11]
Erik Dahlström; Patrick Dengler; Anthony Grasso; Chris Lilley; Cameron McCormack; Doug Schepers; Jonathan Watt; Jon Ferraiolo; Jun Fujisawa; Dean Jackson et al. *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. 16 August 2011. W3C Recommendation. URL: http://www.w3.org/TR/SVG11/

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

[XMLBASE]
Jonathan Marsh; Richard Tobin. *XML Base (Second Edition)*. 28 January 2009. W3C Recommendation. URL: http://www.w3.org/TR/xmlbase/

[XMLSCHEMA-1]
Henry Thompson; David Beech; Murray Maloney; Noah Mendelsohn et al. *XML Schema Part 1: Structures Second Edition*. 28 October 2004. W3C Recommendation. URL: http://www.w3.org/TR/xmlschema-1/

[XPATH]
James Clark; Steven DeRose. *XML Path Language (XPath) Version 1.0*. 16 November 1999. W3C Recommendation. URL: http://www.w3.org/TR/xpath

[XPATH-DATAMODEL-30]
Norman Walsh; Anders Berglund; John Snelson. *XQuery and XPath Data Model 3.0*. 22 October 2013. W3C Proposed Recommendation. URL: http://www.w3.org/TR/xpath-datamodel-30/

[XPATH-FUNCTIONS-30]
Michael Kay. *XPath and XQuery Functions and Operators 3.0*. 22 October 2013. W3C Proposed Recommendation. URL: http://www.w3.org/TR/xpath-functions-30/