RDFa Core 1.1

Syntax and processing rules for embedding RDF through attributes

W3C Working Draft 26 October 2010

This version:
http://www.w3.org/TR/2010/WD-rdfa-core-20101026/
Latest published version:
http://www.w3.org/TR/rdfa-core/
Latest editor's draft:
http://www.w3.org/2010/02/rdfa/drafts#rdfa-core
Previous version:
http://www.w3.org/TR/2010/WD-rdfa-core-20100803/
Latest recommendation:
http://www.w3.org/TR/rdfa-syntax/
Editors:
Ben Adida, Creative Commons ben@adida.net
Mark Birbeck, webBackplane mark.birbeck@webBackplane.com
Shane McCarron, Applied Testing and Technology, Inc. shane@aptest.com
Ivan Herman, W3C ivan@w3.org

This document is also available in these non-normative formats: Diff from previous Working Draft, PostScript version, and PDF version.

Abstract

The current Web is primarily made up of an enormous number of documents that have been created using HTML. These documents contain significant amounts of structured data, which is largely unavailable to tools and applications. When publishers can express this data more completely, and when tools can read it, a new world of user functionality becomes available, letting users transfer structured data between applications and web sites, and allowing browsing applications to improve the user experience: an event on a web page can be directly imported into a user's desktop calendar; a license on a document can be detected so that users can be informed of their rights automatically; a photo's creator, camera setting information, resolution,

location and topic can be published as easily as the original photo itself, enabling structured search and sharing.

RDFa Core is a specification for attributes to express structured data in any markup language. The embedded data already available in the markup language (e.g., XHTML) can often be reused by the RDFa markup, so that publishers don't need to repeat significant data in the document content. The underlying abstract representation is RDF [*RDF-PRIMER [p.58]*], which lets publishers build their own vocabulary, extend others, and evolve their vocabulary with maximal interoperability over time. The expressed structure is closely tied to the data, so that rendered data can be copied and pasted along with its relevant structure.

The rules for interpreting the data are generic, so that there is no need for different rules for different formats; this allows authors and publishers of data to define their own formats without having to update software, register formats via a central authority, or worry that two formats may interfere with each other.

RDFa shares some of the same goals with microformats [*MICROFORMATS [p.58]*]. Whereas microformats specify both a syntax for embedding structured data into HTML documents and a vocabulary of specific terms for each microformat, RDFa specifies only a syntax and relies on independent specification of terms (often called vocabularies or taxonomies) by others. RDFa allows terms from multiple independently-developed vocabularies to be freely intermixed and is designed such that the language can be parsed without knowledge of the specific vocabulary being used.

This document is a detailed syntax specification for RDFa, aimed at:

- those looking to create an RDFa Processor, and who therefore need a detailed description of the parsing rules;
- those looking to integrate RDFa into a new markup language;
- those looking to recommend the use of RDFa within their organization, and who would like to create some guidelines for their users;
- anyone familiar with RDF, and who wants to understand more about what is happening 'under the hood', when an RDFa Processor runs.

For those looking for an introduction to the use of RDFa and some real-world examples, please consult the RDFa Primer.

How to Read this Document

First, if you are not familiar with either RDFa *or* RDF, and simply want to add RDFa to your documents, then you may find the RDFa Primer [*RDFA-PRIMER [p.59]*] to be a better introduction.

If you are already familiar with RDFa, and you want to examine the processing rules â perhaps to create an RDFa Processor â then you'll find the Processing Model [p.22] section of most interest. It contains an overview of each of the processing steps, followed by more detailed sections, one for each rule.

If you are not familiar with RDFa, but you *are* familiar with RDF, then you might find reading the Syntax Overview [p.6] useful, before looking at the Processing Model [p.22] since it gives a range of examples of markup that use RDFa. Seeing some examples first should make reading the processing rules easier.

If you are not familiar with RDF, then you might want to take a look at the section on RDF Terminology [p.12] before trying to do too much with RDFa. Although RDFa is designed to be easy to author â and authors don't need to understand RDF to use it â anyone writing applications that *consume* RDFa will need to understand RDF. There is a lot of material about RDF on the web, and a growing range of tools that support RDFa, this document only contains enough background on RDF to make the goals of RDFa more clear.

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 is a revision of RDFa Syntax 1.0 [*RDFA-SYNTAX [p.59]*]. Once development is complete, if accepted by the W3C membership, this document will supersede the previous Recommendation. There are a number of substantive differences between this version and its predecessor, including:

1. The removal of the specific rules for XHTML - these are now defined in XHTML+RDFa [*XHTML-RDFA [p.58]*]
2. An expansion of the datatypes of some RDFa attributes so that they can contain Terms, CURIES, or Absolute URIs.
3. The ability to change the default vocabulary when no 'prefix' is specified on a CURIE.
4. The ability to reference RDFa Profiles; these are used to ease authoring by creating collections of terms, prefix definitions, and/or default vocabulary declarations.
5. Host languages are permitted to define collections of default terms, default prefix mappings, and a default vocabulary mapping via a default RDFa Profile.
6. Terms are required to be compared in a case-insensitive manner.

A sample test harness is available. This set of tests is not intended to be exhaustive. Users may find the tests to be useful examples of RDFa usage.

This document was published by the RDFa Working Group as a Last Call Working Draft. This document is intended to become a W3C Recommendation. If you wish to make comments regarding this document, please send them to public-rdfa-wg@w3.org (subscribe, archives). The Last Call period ends 06 December 2010. All feedback is welcome.

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

This is a Last Call Working Draft and thus the Working Group has determined that this document has satisfied the relevant technical requirements and is sufficiently stable to advance through the Technical Recommendation process.

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. Motivation

This section is non-normative.

RDF/XML [*RDF-SYNTAX [p.58]*] provides sufficient flexibility to represent all of the abstract concepts in RDF [*RDF-CONCEPTS [p.58]*]. However, it presents a number of challenges; first it is difficult or impossible to validate documents that contain RDF/XML using XML Schemas or DTDs, which therefore makes it difficult to import RDF/XML into other markup languages. Whilst newer schema languages such as RELAX NG [*RELAXNG-SCHEMA [p.59]*] do provide a way to validate documents that contain arbitrary RDF/XML, it will be a while before they gain wide support.

Second, even if one could add RDF/XML directly into an XML dialect like XHTML, there would be significant data duplication between the rendered data and the RDF/XML structured data. It would be far better to add RDF to a document without repeating the document's existing data. For example, an XHTML document that explicitly renders its author's name in the textâperhaps as a byline on a news siteâshould not need to repeat this name for the RDF expression of the same concept: it should be possible to supplement the existing markup in such a way that it can also be interpreted as RDF.

Another reason for aligning the rendered data with the structured data is that it is highly beneficial to express the web data's structure 'in context'; as users often want to transfer structured data from one application to another, sometimes to or from a non-web-based application, the user experience can be enhanced. For example, information about specific rendered data could be presented to the user via 'right-clicks' on an item of interest.

In the past, many attributes were 'hard-wired' directly into the markup language to represent specific concepts. For example, in XHTML 1.1 [*XHTML11 [p.59]*] and HTML [*HTML401 [p.58]*] there is @cite; the attribute allows an author to add information to a document which is used to indicate the origin of a quote.

However, these 'hard-wired' attributes make it difficult to define a generic process for extracting metadata from any document since an RDFa Processor would need to know about each of the special attributes. One motivation for RDFa has been to devise a means by which documents can be augmented with metadata in a general, rather than hard-wired, manner. This has been achieved by creating a fixed set of attributes and parsing rules, but allowing those attributes to contain properties from any of a number of the growing range of available RDF vocabularies. In most cases the *values* of those properties are the information that is already in an author's document.

RDFa alleviates the pressure on markup language designers to anticipate all the structural requirements users of their language might have, by outlining a new syntax for RDF that relies only on attributes. By adhering to the concepts and rules in this specification, language designers can import RDFa into their environment with a minimum of hassle and be confident that semantic data will be extractable from their documents by conforming processors.

2. Syntax Overview

This section is non-normative.

The following examples are intended to help readers who are not familiar with RDFa to quickly get a sense of how it works. For a more thorough introduction, please read the RDFa Primer [*RDFA-PRIMER [p.59]*].

For brevity, in the following examples and throughout this document, assume that the following vocabulary prefixes [p.20] have been defined:

bibo:	http://purl.org/ontology/bibo/
cc:	http://creativecommons.org/ns#
dbp:	http://dbpedia.org/property/
dbr:	http://dbpedia.org/resource/
dcterms:	http://purl.org/dc/terms/
ex:	http://example.org/
foaf:	http://xmlns.com/foaf/0.1/
rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfa:	http://www.w3.org/ns/rdfa#
rdfs:	http://www.w3.org/2000/01/rdf-schema#
xhv:	http://www.w3.org/1999/xhtml/vocab#
xsd:	http://www.w3.org/2001/XMLSchema#

2.1 The RDFa Attributes

RDFa makes use of a number of commonly found attributes, as well as providing a few new ones. Attributes that already exist in widely deployed languages (e.g., HTML) have the same meaning they always did, although their syntax has been slightly modified in some cases. For example, in (X)HTML, @rel [p.19] already defines the relationship between one document and another. However, in (X)HTML there is no clear way to add new values; RDFa sets out to explicitly solve this problem, and does so by allowing URIs as values. It also introduces the concepts of terms [p.53] and ’compact URIs [p.53] ’ â referred to as CURIEs in this document â which allow a full URI value to be expressed succinctly. For a complete list of RDFa attribute names and syntax, see Attributes and Syntax [p.18] .

2.2 Examples

As an (X)HTML author you will already be familiar with using `meta` and `link` to add additional information to your documents:

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Page 7</title>
    <meta name="author" content="Mark Birbeck" />
    <link rel="prev" href="page6.html" />
    <link rel="next" href="page8.html" />
  </head>
  <body>...</body>
</html>
```

RDFa makes use of this concept, enhancing it with the ability to make use of other vocabularies by using compact URIs:

```
<html
  xmlns="http://www.w3.org/1999/xhtml"
  prefix="foaf: http://xmlns.com/foaf/0.1/
          dcterms: http://purl.org/dc/terms/"
  >
  <head>
    <title>My home-page</title>
    <meta property="dcterms:creator" content="Mark Birbeck" />
    <link rel="foaf:topic" href="http://www.formsPlayer.com/#us" />
  </head>
  <body>...</body>
</html>
```

RDFa supports the use of @rel [p.19] and @rev [p.19] on any element. This is even more useful when with the addition of support for different vocabularies:

```
This document is licensed under a
<a prefix="cc: http://creativecommons.org/ns#"
   rel="cc:license"
   href="http://creativecommons.org/licenses/by-nc-nd/3.0/">
   Creative Commons License
</a>.
```

Not only can URLs in the document be re-used to provide metadata, but so can inline text:

```
<html
  xmlns="http://www.w3.org/1999/xhtml"
  prefix="cal: http://www.w3.org/2002/12/cal/ical#"
  >
  <head><title>Jo’s Friends and Family Blog</title></head>
  <body>
    <p>
      I’m holding
      <span property="cal:summary">
        one last summer Barbecue
      </span>,
```

```
      on September 16th at 4pm.
    </p>
  </body>
</html>
```

If some displayed text is different to the actual 'value' it represents, more precise values can be added, which can optionally include datatypes:

```
<html
  xmlns="http://www.w3.org/1999/xhtml"
  prefix="cal: http://www.w3.org/2002/12/cal/ical#
  xsd: http://www.w3.org/2001/XMLSchema"
  >
  <head><title>Jo's Friends and Family Blog</title></head>
  <body>
    <p>
      I'm holding
      <span property="cal:summary">
        one last summer Barbecue
      </span>,
      on
      <span property="cal:dtstart" content="2015-09-16T16:00:00-05:00"
            datatype="xsd:dateTime">
        September 16th at 4pm
      </span>.
    </p>
  </body>
</html>
```

In many cases a block of markup will contain a number of properties that relate to the same item; it's possible with RDFa to indicate the type of that item:

```
<html
  xmlns="http://www.w3.org/1999/xhtml"
  prefix="cal: http://www.w3.org/2002/12/cal/ical#
          xsd: http://www.w3.org/2001/XMLSchema"
  >
  <head><title>Jo's Friends and Family Blog</title></head>
  <body>
    <p typeof="cal:Vevent">
      I'm holding
      <span property="cal:summary">
        one last summer Barbecue
      </span>,
      on
      <span property="cal:dtstart" content="2015-09-16T16:00:00-05:00"
            datatype="xsd:dateTime">
        September 16th at 4pm
      </span>.
    </p>
  </body>
</html>
```

RDFa allows the document to contain metadata information about other documents and resources:

```
<html
  xmlns="http://www.w3.org/1999/xhtml"
  prefix="bibo: http://purl.org/ontology/bibo/
          dcterms: http://purl.org/dc/terms/"
  >
  <head>
    <title>Books by Marco Pierre White</title>
  </head>
  <body>
    I think White’s book
    ’<span about="urn:ISBN:0091808189" typeof="bibo:Book"
           property="dcterms:title">
      Canteen Cuisine
    </span>’
    is well worth getting since although it’s quite advanced stuff, he
    makes it pretty easy to follow. You might also like
    <span about="urn:ISBN:1596913614" typeof="bibo:Book"
          property="dcterms:description">
      White’s autobiography
    </span>.
  </body>
</html>
```

When dealing with small amounts of markup, it is sometimes easier to use full URIs, rather than CURIEs. The previous example can also be written as follows:

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Books by Marco Pierre White</title>
  </head>
  <body>
    I think White’s book
    ’<span
     about="urn:ISBN:0091808189"
     typeof="http://purl.org/ontology/bibo/Book"
     property="http://purl.org/dc/terms/title"
     >Canteen Cuisine</span>’
    is well worth getting since although it’s quite advanced stuff, he
    makes it pretty easy to follow. You might also like
    <span
     about="urn:ISBN:1596913614"
     typeof="http://purl.org/ontology/bibo/Book"
     property="http://purl.org/dc/terms/description"
     >White’s autobiography</span>.
  </body>
</html>
```

A simple way of defining a portion of a document to use FOAF terms is to use @vocab [p.19] to define a default vocabulary URI:

```
<div vocab="http://xmlns.com/foaf/0.1/" about="#me">
   My name is <span property="name">John Doe</span> and my blog is called
   <a rel="homepage" href="http://example.org/blog/">Understanding Semantics</a>.
</div>
```

the following triples will be generated:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
<#me> foaf:name "John Doe" ;
      foaf:homepage <http://example.org/blog/> .
```

RDFa also permits external definition of collections of prefixes [p.20] . The following (mythical) example RDFa Profile document, with a URI of `http://www.example.org/vocab-rdf-dc.html`, defines the standard RDF prefixes as well as the FOAF and Dublin Core vocabulary prefixes in RDFa.

```
<html xmlns="http://www.w3.org/1999/xhtml"
      prefix="rdfa: http://www.w3.org/ns/rdfa#">
  <head>
     ...
  </head>
  <body>
    <p>This is an example to defining the standard RDF and
       Dublin Core prefixes
    </p>

    <p typeof="">
       The "<span property="rdfa:prefix">rdf</span>" prefix can
       be used for the URI:
       "<span property="rdfa:uri">http://www.w3.org/1999/02/22-rdf-syntax-ns#</span>".</p>

    <p typeof="">
       The "<span property="rdfa:prefix">rdfs</span>" prefix can
       be used for the URI:
       "<span property="rdfa:uri">http://www.w3.org/2000/01/rdf-schema#</span>".</p>

    <p typeof="">
       The "<span property="rdfa:prefix">dcterms</span>" prefix can
       be used for the URI:
       "<span property="rdfa:uri">http://purl.org/dc/terms/</span>".</p>

    <p typeof="">
       The "<span property="rdfa:prefix">foaf</span>" prefix can
       be used for the URI:
       "<span property="rdfa:uri">http://xmlns.com/foaf/0.1/</span>".</p>
</html>
```

Using @profile [p.19] , the following RDFa snippet:

```
<p about="http://www.example.org/doc"
   profile="http://www.example.org/vocab-rdf-dc">
  <span property="dcterms:title">title of the document</span>
  <span property="rdfs:comment">and this is a longer comment
        on the same document</span>
</p>
```

would yield the following triples:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
<http://www.example.org/doc>
  dcterms:title "title of the document" ;
  rdfs:comment "and this is a longer comment on the same document" .
```

It is also possible to define terms. Given the following RDFa Profile document at `http://www.example.org/vocab-foaf-terms.html`:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      prefix="rdfa: http://www.w3.org/ns/rdfa#">
  <head>
    <title>Example RDFa Vocabulary</title>
  </head>
  <body>
   <p>
      This is an example RDFa vocabulary that makes it easier to
      use the foaf:name and foaf:homepage terms.
    </p>
    <p typeof="">
       The "<span property="rdfa:term">name</span>" term can
       be used for the URI:
       "<span property="rdfa:uri">http://xmlns.com/foaf/0.1/name</span>".</p>
    <p typeof="">
       The "<span property="rdfa:term">homepage</span>" term can
       be used for the URI:
       "<span property="rdfa:uri">http://xmlns.com/foaf/0.1/homepage</span>".</p>
  </body>
</html>
```

and the following HTML markup:

```
<div profile="http://www.example.org/vocab-foaf-terms" about="#me">
   My name is <span property="name">John Doe</span> and my blog is called
   <a rel="homepage"
      href="http://example.org/blog/">Understanding Semantics</a>.
</div>
```

the following triples will be generated:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
<#me> foaf:name "John Doe" ;
      foaf:homepage <http://example.org/blog/> .
```

3. RDF Terminology

This section is non-normative.

The previous section gave examples of typical markup in order to illustrate the structure of RDFa markup. However, what RDFa *represents* is RDF. In order to author RDFa you do not need to understand RDF, although it would certainly help. However, if you are building a system that

consumes the RDF output of a language that supports RDFa you will almost certainly need to understand RDF. This section introduces the basic concepts and terminology of RDF. For a more thorough explanation of RDF, please refer to the RDF Concepts document [*RDF-CONCEPTS [p.58]*] and the RDF Syntax Document [*RDF-SYNTAX [p.58]*].

3.1 Statements

The structured data that RDFa provides access to is a collection of *statements*. A statement is a basic unit of information that has been constructed in a specific format to make it easier to process. In turn, by breaking large sets of information down into a collection of statements, even very complex metadata can be processed using simple rules.

To illustrate, suppose we have the following set of facts:

```
Albert was born on March 14, 1879, in the German Empire. There is a picture of him at
the web address, http://en.wikipedia.org/wiki/Image:Albert_Einstein_Head.jpg.
```

This would be quite difficult for a machine to interpret, and it is certainly not in a format that could be passed from one data application to another. However, if we convert the information to a set of statements it begins to be more manageable. The same information could therefore be represented by the following shorter ’statements’:

```
Albert was born on March 14, 1879.
Albert was born in the German Empire.
Albert has a picture at
  http://en.wikipedia.org/wiki/Image:Albert_Einstein_Head.jpg.
```

3.2 Triples

To make this information machine-processable, RDF defines a structure for these statements. A statement is formally called a triple, meaning that it is made up of three components. The first is the *subject* of the triple, and is what we are making our statements *about*. In all of these examples the subject is ’Albert’.

The second part of a triple is the property of the subject that we want to define. In the examples here, the properties would be ’was born on’, ’was born in’, and ’has a picture at’. These are more usually called *predicates* in RDF.

The final part of a triple is called the *object*. In the examples here the three objects have the values ’March 14, 1879’, ’the German Empire’, and ’http://en.wikipedia.org/wiki/Image:Albert_Einstein_Head.jpg’.

3.3 URI references

Breaking complex information into manageable units helps us be specific about our data, but there is still some ambiguity. For example, which ’Albert’ are we talking about? If another system has more facts about ’Albert’, how could we know whether they are about the same person, and so add them to the list of things we know about that person? If we wanted to find people born in

the German Empire, how could we know that the predicate ’was born in’ has the same purpose as the predicate ’birthplace’ that might exist in some other system? RDF solves this problem by replacing our vague terms with *URI references*.

URIs are most commonly used to identify web pages, but RDF makes use of them as a way to provide unique identifiers for concepts. For example, we could identify the subject of all of our statements (the first part of each triple) by using the DBPedia [http://dbpedia.org] URI for Albert Einstein, instead of the ambiguous string ’Albert’:

```
<http://dbpedia.org/resource/Albert_Einstein>
   has the name
   Albert Einstein.
<http://dbpedia.org/resource/Albert_Einstein>
   was born on
   March 14, 1879.
<http://dbpedia.org/resource/Albert_Einstein>
   was born in
   the German Empire.
<http://dbpedia.org/resource/Albert_Einstein>
   has a picture at
   http://en.wikipedia.org/wiki/Image:Albert_Einstein_Head.jpg.
```

URI references are also used to uniquely identify the objects in metadata statements (the third part of each triple). The picture of Einstein is already a URI, but we could also use a URI to uniquely identify the country ’German Empire’. At the same time we’ll indicate that the name and date of birth really are literals (and not URIs), by putting quotes around them:

```
<http://dbpedia.org/resource/Albert_Einstein>
   has the name
   "Albert Einstein".
<http://dbpedia.org/resource/Albert_Einstein>
   was born on
   "March 14, 1879".
<http://dbpedia.org/resource/Albert_Einstein>
   was born in
   <http://dbpedia.org/resource/German_Empire>.
<http://dbpedia.org/resource/Albert_Einstein>
   has a picture at
   <http://en.wikipedia.org/wiki/Image:Albert_Einstein_Head.jpg>.
```

URI references are also used to ensure that predicates are unambiguous; now we can be sure that ’birthplace’, ’place of birth’, ’Lieu de naissance’ and so on, all mean the same thing:

```
<http://dbpedia.org/resource/Albert_Einstein>
  <http://xmlns.com/foaf/0.1/name>
  "Albert Einstein".
<http://dbpedia.org/resource/Albert_Einstein>
  <http://dbpedia.org/property/dateOfBirth>
  "March 14, 1879".
<http://dbpedia.org/resource/Albert_Einstein>
  <http://dbpedia.org/property/birthPlace>
```

```
  <http://dbpedia.org/resource/German_Empire>.
<http://dbpedia.org/resource/Albert_Einstein>
  <http://xmlns.com/foaf/0.1/depiction>
  <http://en.wikipedia.org/wiki/Image:Albert_Einstein_Head.jpg>.
```

3.4 Plain literals

Although URI resources are always used for subjects and predicates, the object part of a triple can be either a URI or a literal. In the example triples, Einstein's name is represented by a plain literal, which means that it is a basic string with no type or language information:

```
<http://dbpedia.org/resource/Albert_Einstein>
  <http://xmlns.com/foaf/0.1/name> "Albert Einstein".
```

3.5 Typed literals

Some literals, such as dates and numbers, have very specific meanings, so RDF provides a mechanism for indicating the type of a literal. A typed literal is indicated by attaching a URI to the end of a plain literal [p.15] , and this URI indicates the literal's datatype. This URI is usually based on datatypes defined in the XML Schema Datatypes specification [*XMLSCHEMA-2 [p.58]*]. The following syntax would be used to unambiguously express Einstein's date of birth as a literal of type `http://www.w3.org/2001/XMLSchema#date`:

```
<http://dbpedia.org/resource/Albert_Einstein>
  <http://dbpedia.org/property/dateOfBirth>
     "1879-03-14"^^<http://www.w3.org/2001/XMLSchema#date>.
```

3.6 Turtle

RDF itself does not have one set way to express triples, since the key ideas of RDF are the triple and the use of URIs, and *not* any particular syntax. However, there are a number of mechanisms for expressing triples, such as RDF/XML [*RDF-SYNTAX-GRAMMAR [p.57]*], Turtle [*TURTLE [p.59]*], and of course RDFa. Many discussions of RDF make use of the *Turtle* syntax to explain their ideas, since it is quite compact. The examples we have just seen are already using this syntax, and we'll continue to use it throughout this document when we need to talk about the RDF that could be generated from some RDFa. Turtle allows long URIs to be abbreviated by using a URI mapping, which can be used to express a compact URI as follows:

```
@prefix dbp: <http://dbpedia.org/property/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
<http://dbpedia.org/resource/Albert_Einstein>
  foaf:name "Albert Einstein" .
<http://dbpedia.org/resource/Albert_Einstein>
  dbp:birthPlace <http://dbpedia.org/resource/German_Empire> .
```

Here 'dbp:' has been mapped to the URI for DBPedia and 'foaf:' has been mapped to the URI for the 'Friend of a Friend' taxonomy.

Any URI in Turtle could be abbreviated in this way. This means that we could also have used the same technique to abbreviate the identifier for Einstein, as well as the datatype indicator:

```
@prefix dbp: <http://dbpedia.org/property/> .
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
dbr:Albert_Einstein dbp:dateOfBirth "1879-03-14"^^xsd:date .
dbr:Albert_Einstein
  foaf:depiction <http://en.wikipedia.org/wiki/Image:Albert_Einstein_Head.jpg> .
```

When writing examples, you will often see the following URI in the Turtle representation:

```
<>
```

This indicates the 'current document', i.e., the document being processed. In reality there would always be a full URI based on the document's location, but this abbreviation serves to make examples more compact. Note in particular that the whole technique of abbreviation is merely a way to make examples more compact, and the actual triples generated would always use the full URIs.

3.7 Graphs

A collection of triples is called a *graph*. All of the triples that are defined by this specification are contained in the default graph [p.17] by an RDFa Processor. For more information on graphs and other RDF concepts, see [*RDF-CONCEPTS [p.58]*].

3.8 Compact URIs

In order to allow for the compact expression of RDF statements, RDFa allows the contraction of most URI references into a form called a 'compact URI', or CURIE [p.53] . A detailed discussion of this mechanism is in the section CURIE and URI Processing [p.25] .

Note that CURIEs are only used in the markup and Turtle examples, and will never appear in the generated triple [p.13] s, which are defined by RDF to use URI references.

Full details on how CURIEs are processed are in the section titled CURIE Processing [p.25] .

3.9 Markup Fragments and RDFa

A growing use of embedded metadata is to take fragments of markup and move them from one document to another. This may happen through the use of tools, such as drag-and-drop in a browser, or through snippets of code provided to authors for inclusion in their documents. (A good example of the latter is the licensing fragment provided by Creative Commons [p.8] .)

However, those involved in creating fragments (either by building tools, or authoring snippets), should be aware that this specification does not say how fragments are processed. Specifically, the processing of a fragment 'outside' of a complete document is undefined because RDFa processing is largely about context. Future versions of this or related specifications may do more

to define this behavior.

Developers of tools that process fragments, or authors of fragments for manual inclusion, should also bear in mind what will happen to their fragment once it is included in a complete document. They should carefully consider the amount of 'context' information that will be needed in order to ensure a correct interpretation of their fragment.

3.10 A description of RDFa in RDF terms

The following is a brief description of RDFa in terms of the RDF terminology introduced here. It may be useful to readers with an RDF background:

The aim of RDFa is to allow a single RDF graph [p.17] to be carried in various types of document markup. An *RDF graph* comprises *node*s linked by relationships. The basic unit of an RDF graph [p.17] is a triple [p.13] , in which a subject node [p.17] is linked to an object node [p.17] via a predicate [p.17] . The *subject* node [p.17] is always either a URI reference or a *blank node (or bnode)*, the *predicate* is *always* a URI reference, and the object of a statement can be a URI reference, a literal [p.15] , or a bnode [p.17] .

In RDFa, a subject URI reference is generally indicated using @about [p.18] or @src [p.19] , and predicates are represented using one of @property [p.19] , @rel [p.19] , or @rev [p.19] . Objects which are URI references are represented using @resource [p.19] , or @href [p.19] , whilst objects that are literal [p.15] s are represented either with @content [p.19] or the content of the element in question (with an optional datatype expressed using @datatype [p.19] , and an optional language expressed using a Host Language-defined mechanism such as @xml:lang).

4. Conformance

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

The key words *must*, *must not*, *required*, *should*, *should not*, *recommended*, *may*, and *optional* in this specification are to be interpreted as described in [*RFC2119 [p.57]*].

4.1 RDFa Processor Conformance

A conforming RDFa Processor *must* make available to a consuming application a single RDF graph [p.17] containing all possible triples generated by using the rules in the Processing Model [p.22] section. This specification uses the term *default graph* to mean all of the triples asserted by a document according to the Processing Model [p.22] section. The *processor graph* term is used to denote the collection of all informational, warning, and error triples that are generated by the RDFa Processor as a result of processing the document. The default graph [p.17] and the processor graph [p.17] are separate graphs and *must not* be stored in the same graph by the RDFa Processor.

A conforming RDFa Processor *may* make available additional triples that have been generated using rules not described here, but these triples *must not* be made available in the default graph [p.17] . (Whether these additional triples are made available in one or more additional RDF graph [p.17] s is implementation-specific, and therefore not defined here.)

A conforming RDFa Processor must preserve white space in both plain literal [p.15] s and XML literals [p.49] . However, it may be the case that the architecture in which a processor operates does not make all white space available. It is therefore advisable for authors who would like to make their documents consumable across different processors, to remove any unnecessary white space in their markup.

4.2 RDFa Host Language Conformance

Host Languages that incorporate RDFa must adhere to the following:

- All of the facilities required in this specification *must* be included in the Host Language.
- The attributes defined in this specification *must* be included in the content model of the Host Language.
- If the Host Language uses XML Namespaces [*XML-NAMES [p.58]*], the attributes in this specification *should* be defined in 'no namespace'. (e.g., when the attributes are used on elements in the Host Language's namespace, they can be used with no qualifying prefix: `<myml:myElement property="next">`).
- If the Host Language has its own definition for any attribute defined in this specification, that definition *must* be such that the processing required by this specification remains possible when the attribute is used in a way consistent with the requirements herein.
- The Host Language *may* define a default RDFa Profile. If it does, the RDFa Profile triples that establish term or URI mappings associated with that profile *must not* change without changing the profile URI. RDFa Processors *may* embed, cache, or retrieve the RDFa Profile triples associated with that profile.

  Host Languages are required to change the URI of their default profile if items are added or removed from the default profile document. The URI change is required to accomodate RDFa Processors that statically embed the terms defined in the profile. Host Languages are expected to change their profiles very rarely.

5. Attributes and Syntax

This specification defines a number of attributes and the way in which the values of those attributes are to be interpreted when generating RDF triples. This section defines the attributes and the syntax of their values.

about
: a SafeCURIEorCURIEorURI [p.53] , used for stating what the data is about (a 'subject' in RDF terminology);

content
: a CDATA string, for supplying machine-readable content for a literal (a 'plain literal object', in RDF terminology);

datatype
: a TERMorCURIEorAbsURI [p.53] representing a datatype, to express the datatype of a literal;

href (optional)
: a traditionally navigable URI for expressing the partner resource of a relationship (a 'resource object', in RDF terminology);

prefix
: a white space separated list of prefix-name URI pairs of the form

```
NCName ':' ' '+ xs:anyURI
```

profile
: a white space separated list of one or more URIs that indicate a profile or terms, prefix mappings, and/or default vocabulary declarations. See RDFa Profiles [p.51] ;

property
: a white space separated list of TERMorCURIEorAbsURIs [p.53] , used for expressing relationships between a subject and some literal text (also a 'predicate');

rel
: a white space separated list of TERMorCURIEorAbsURIs [p.53] , used for expressing relationships between two resources ('predicates' in RDF terminology);

resource
: a SafeCURIEorCURIEorURI [p.53] for expressing the partner resource of a relationship that is not intended to be navigable (e.g., a 'clickable' link) (also an 'object');

rev
: a white space separated list of TERMorCURIEorAbsURIs [p.53] , used for expressing reverse relationships between two resources (also 'predicates');

src (optional)
: a URI for expressing the partner resource of a relationship when the resource is embedded (also a 'resource object');

typeof
: a white space separated list of TERMorCURIEorAbsURIs [p.53] that indicate the RDF type(s) to associate with a subject;

vocab
: A URI that defines the mapping to use when a TERM [p.53] is referenced in an attribute value. See General Use of Terms in Attributes [p.28] .

xmlns:prefix (optional)
: A method of declaring prefix mappings as defined in [*XML-NAMES [p.58]*]. Prefix mappings declared via this attribute are equivalent to those declared using @prefix [p.20] . If this attribute and @prefix [p.20] declare a mapping for the same prefix on the same element, the mapping from @prefix [p.20] *must* take precedence. Document authors *should* use @prefix [p.20] , and *should not* mix @prefix [p.20] and this attribute on the same element.

5.1 White space within attribute values

Many attributes accept a white space separated list of tokens. This specification defines white space as:

```
whitespace      ::=  (#x20 | #x9 | #xD | #xA)+
```

When attributes accept a white space separated list of tokens, an RDFa Processor *must* ignore any leading or trailing white space.

This definition is consistent with the definition found in [*XML10 [p.59]*].

6. CURIE Syntax Definition

The key component of RDF is the URI, but these are usually long and unwieldy. RDFa therefore supports a mechanism by which URIs can be abbreviated, called 'compact URIs' or simply, CURIEs.

A CURIE is comprised of two components, a *prefix* and a *reference*. The prefix is separated from the reference by a colon (`:`). In general use it is possible to omit the prefix, and so create a CURIE that makes use of the 'default prefix' mapping; in RDFa the 'default prefix' mapping is `http://www.w3.org/1999/xhtml/vocab#`. It's also possible to omit both the prefix *and* the colon, and so create a CURIE that contains just a reference which makes use of the 'no prefix' mapping. This specification does not define a default 'no prefix' mapping. However, Host Languages *may* define a default. This mapping *may* be changed via @vocab [p.19] .

The RDFa 'default prefix' should not be confused with the 'default namespace' as defined in [*XML-NAMES [p.58]*]. An RDFa Processor *must not* treat an XML-NAMES 'default namespace' declaration as if it were setting the 'default prefix'.

The general syntax of a CURIE can be summarized as follows:

```
prefix      ::=   NCName

reference   ::=   irelative-ref (as defined in [RFC3987])

curie      ::=   [ [ prefix ] ':' ] reference

safe_curie  ::=   '[' [ [ prefix ] ':' ] reference ']'
```

The production `safe_curie` is not required, even in situations where an attribute value is permitted to be a CURIE or a URI: A URI that uses a scheme that is not an in-scope mapping *cannot* be confused with a CURIE. The concept of a safe_curie is retained for backward compatibility.

In normal evaluation of CURIEs the following context information would need to be provided:

- a set of mappings from prefixes to URIs;
- a mapping to use with the default prefix (for example, `:p`);
- a mapping to use when there is no prefix (for example, p);
- a mapping to use with the '_' prefix, which is used to generate unique identifiers (for example, `_:p`).

In RDFa these values are defined as follows:

- the **set of mappings from prefixes to URIs** is provided by the current in-scope prefix declarations of the current element [p.31] during parsing;
- the **mapping to use with the default prefix** is the current default prefix mapping;
- the **mapping to use when there is no prefix** is not defined (see General Use of Terms in Attributes [p.28] for the way items with no colon can be interpreted in some datatypes) ;
- the **mapping to use with the '_' prefix**, is not explicitly stated, but since it is used to generate bnode [p.17] s, its implementation needs to be compatible with the RDF definition and rules in Referencing Blank Nodes [p.29] . A document *should not* define a mapping for the '_' prefix. A Conforming RDFa Processor *must* ignore any definition of a mapping for the '_' prefix.

A CURIE is a representation of a full URI. The rules for determining that URI are:

- If a CURIE consists of an empty `prefix` and a `reference`, the URI is obtained by taking the current default prefix mapping and concatenating it with the `reference`. If there is no current default prefix mapping, then this is not a valid CURIE and *must* be ignored.
- Otherwise, if a CURIE consists of a non-empty `prefix` and `reference`, and if there is an in-scope mapping for `prefix` (when compared case-insensitively), then the URI is created by using that mapping, and concatenating it with the `reference`.
- Finally, if there is no in-scope mapping for `prefix`, then the value is not a CURIE.

Note that the resulting URI *must* be a syntactically valid IRI [*RFC3987 [p.57]*]. For a more detailed explanation see CURIE and URI Processing [p.25] . Also note that while the *lexical space* of a CURIE is as defined in curie [p.20] above, the *value space* is the set of IRIs.

6.1 Why CURIEs and not QNames?

This section is non-normative.

In many cases, language designers have attempted to use QNames for an extension mechanism [*XMLSCHEMA-2 [p.58]*]. QNames do permit independent management of the name collection, and *can* map the names to a resource. Unfortunately, QNames are unsuitable in most cases because 1) the use of QName as identifiers in attribute values and element content is problematic as discussed in [*QNAMES [p.58]*] and 2) the syntax of QNames is overly restrictive and does not allow all possible URIs to be expressed.

A specific example of the problem this causes comes from attempting to define the name collection for books. In a QName, the part after the colon must be a valid element name, making an example such as the following *invalid*: `isbn:0321154991`

This is not a valid QName simply because "0321154991" is not a valid element name. Yet, in the example given, we don’t really want to define a valid element name anyway. The whole reason for using a QName was to reference an item in a private scope - that of ISBNs. Moreover, in this example, we want the names within that scope to map to a URI that will reveal the meaning of that ISBN. As you can see, the definition of QNames and this (relatively common) use case are in conflict with one another.

This specification addresses the problem by defining CURIEs. Syntactically, CURIEs are a superset of QNames.

Note that this specification is targeted at language designers, not document authors. Any language designer considering the use of QNames as a way to represent URIs or unique tokens should consider instead using CURIEs:

- CURIEs are designed from the ground up to be used in attribute values. QNames are designed for unambiguously naming elements and attributes.
- CURIEs expand to IRIs, and any IRI can be represented by such a mapping. QNames are treated as value pairs, but even if those pairs are combined into a string, only a subset of IRIs can be represented.
- CURIEs can be used in non-XML grammars, and can even be used in XML languages that do not support XML Namespaces. QNames are limited to XML Namespace-aware XML Applications.

7. Processing Model

This section looks at a generic set of processing rules for creating a set of triples that represent the structured data present in an RDFa document. Processing need not follow the DOM traversal technique outlined here, although the effect of following some other manner of processing must be the same as if the processing outlined here were followed. The processing model is explained using the idea of DOM traversal which makes it easier to describe (particularly in relation to the evaluation context [p.30]).

Note that in this section, explanations about the processing model or guidance to implementors are enclosed in sections like this.

7.1 Overview

Evaluating a document for RDFa triples is carried out by starting at the document object, and then visiting each of its child elements in turn, in document order, applying processing rules. Processing is recursive in that for each child element the processor also visits each of *its* child elements, and applies the same processing rules.

In some environments there will be little difference between starting at the root element of the document, and starting at the document object itself. It is defined this way because in some environments important information is present at the document object level which is not present on the root element.

As processing continues, rules are applied which may generate triples, and may also change the evaluation context [p.30] information that will then be used when processing descendant elements.

This specification does not say anything about what should happen to the triples generated, or whether more triples might be generated during processing than are outlined here. However, to be conforming, an RDFa Processor *must* act as if at a minimum the rules in this section are applied, and a single RDF graph [p.17] produced. As described in the RDFa Processor Conformance [p.17] section, any additional triples generated *must not* appear in the default graph [p.17] .

7.2 Evaluation Context

During processing, each rule is applied using information provided by an evaluation context [p.30] . An initial context is created when processing begins. That context has the following members:

- The *base*. This will usually be the URL of the document being processed, but it could be some other URL, set by some other mechanism, such as the (X)HTML `base` element. The important thing is that it establishes a URL against which relative paths can be resolved.
- The *parent subject*. The initial value will be the same as the initial value of base [p.23] , but it will usually change during the course of processing.
- The *parent object*. In some situations the object of a statement becomes the subject of any nested statements, and this property is used to convey this value. Note that this value may be a bnode [p.17] , since in some situations a number of nested statements are grouped together on one bnode [p.17] . This means that the bnode [p.17] must be set in the containing statement and passed down, and this property is used to convey this value.
- A list of current, in-scope *URI mappings*.
- A list of *incomplete triple*s. A triple can be incomplete when no object resource is provided alongside a predicate that requires a resource (i.e., @rel [p.19] or @rev [p.19]). The triples can be completed when a resource becomes available, which will be when the next subject is specified (part of the process called chaining [p.24]).
- The *language*. Note that there is no default language.
- The *term mappings*, a list of terms and their associated URIs. This specification does not define an initial list. Host Languages *may* define an initial list. If a Host Language provides an initial list, it *should* do so via an RDFa Profile.
- The *default vocabulary*, a value to use as the prefix URI when a term [p.53] is used. This specification does not define an initial setting for the default vocabulary. Host Languages *may* define an initial setting. If a Host Language defines an initial setting, it *should* do so via an RDFa Profile document.

During the course of processing, new evaluation context [p.30] s are created which are passed to each child element. The rules described below will determine the values of the items in the context. Additionally, some rules will cause new triples to be created by combining information provided by an element with information from the evaluation context [p.30] .

During the course of processing a number of locally scoped values are needed, as follows:

- An initially empty list of URI mapping [p.31] s, called the *local list of URI mappings*.
- An initially empty *list of incomplete triples*, called the *local list of incomplete triples*.
- An initially empty language [p.23] value.
- A *skip element* flag, which indicates whether the current element [p.31] can safely be ignored since it has no relevant RDFa attributes. Note that descendant elements will still be processed.
- A *new subject* value, which once calculated will set the parent subject [p.23] property in an evaluation context [p.30] , as well as being used to complete any incomplete triple [p.23] s, as described in the next section.
- A value for the *current object literal*, the literal to use when creating triples that have a literal object.
- A value for the *current object resource*, the resource to use when creating triples that have a resource object.
- The *local term mappings*, a list of terms and their associated URIs.
- A *local default vocabulary*, a URI to use as a prefix mapping when a term [p.53] is used.

7.3 Chaining

Statement *chaining* is an RDFa feature that allows the author to link RDF statements together while avoiding unnecessary repetitive markup. For example, if an author were to add statements as children of an object that was a resource, these statements should be interpreted as being about that resource:

```
<div about="http://dbpedia.org/resource/Albert_Einstein">
  <span property="foaf:name">Albert Einstein</span>
  <span property="dbp:dateOfBirth" datatype="xsd:date">1879-03-14</span>
  <div rel="dbp:birthPlace" resource="http://dbpedia.org/resource/German_Empire">
    <span property="dbp:conventionalLongName">the German Empire</span>
  </div>
</div>
```

In this example we can see that an object resource (’German_Empire’), has become the subject for nested statements. This markup also illustrates the basic chaining pattern of ’A has a B has a C’ (i.e., Einstein has a birth place of the German Empire, which has a long name of "the German Empire").

It’s also possible for the subject of nested statements to provide the object for *containing* statements â essentially the reverse of the example we have just seen. To illustrate, we’ll take an example of the type of chaining just described, and show how it could be marked up more efficiently. To start, we mark up the fact that Albert Einstein had both German and American citizenship:

```
<div about="http://dbpedia.org/resource/Albert_Einstein">
  <div rel="dbp:citizenship" resource="http://dbpedia.org/resource/German_Empire"></div>
  <div rel="dbp:citizenship" resource="http://dbpedia.org/resource/United_States"></div>
</div>
```

Now, we show the same information, but this time we create an incomplete triple [p.23] from the citizenship part, and then use any number of further subjects to 'complete' that triple, as follows:

```
<div about="http://dbpedia.org/resource/Albert_Einstein" rel="dbp:citizenship">
  <span about="http://dbpedia.org/resource/German_Empire"></span>
  <span about="http://dbpedia.org/resource/United_States"></span>
</div>
```

In this example, the incomplete triple [p.23] actually gets completed twice, once for the German Empire and once for the USA, giving exactly the same information as we had in the earlier example:

```
<http://dbpedia.org/resource/Albert_Einstein>
  dbp:citizenship <http://dbpedia.org/resource/German_Empire> .
<http://dbpedia.org/resource/Albert_Einstein>
  dbp:citizenship <http://dbpedia.org/resource/United_States> .
```

Chaining can sometimes involve elements containing relatively minimal markup, for example showing only one resource, or only one predicate. Here the `img` element is used to carry a picture of Einstein:

```
<div about="http://dbpedia.org/resource/Albert_Einstein">
  <div rel="foaf:depiction">
    <img src="http://en.wikipedia.org/wiki/Image:Albert_Einstein_Head.jpg" />
  </div>
</div>
```

When such minimal markup is used, any of the resource-related attributes could act as a subject or an object in the chaining:

```
<div about="http://dbpedia.org/resource/Albert_Einstein">
  <div rel="dbp:citizenship">
    <span about="http://dbpedia.org/resource/German_Empire"></span>
    <span about="http://dbpedia.org/resource/United_States"></span>
  </div>
</div>
```

7.4 CURIE and URI Processing

Since RDFa is ultimately a means for transporting RDF, a key concept is the *resource* and its manifestation as a URI. RDF deals with complete URIs (not relative paths); when converting RDFa to triples, any relative URIs *must* be resolved relative to the base URI, using the algorithm defined in section 5 of RFC 3986 [*URI [p.58]*], *Reference Resolution*. The values of RDFa attributes [p.18] that refer to URIs use three different datatypes: URI, SafeCURIEorCURIEorURI [p.53] , or TERMorCURIEorAbsURI [p.53] . All these attributes are mapped, after processing, to URIs. The handling of these attributes is as follows:

URI
: The content is a URI, and is used as such.

SafeCURIEorCURIEorURI

- When the value is surrounded by square brackets, then the content within the brackets is evaluated as a CURIE according to the CURIE Syntax definition [p.20] . If it is not a valid CURIE, the value *must* be ignored.
- Otherwise, the value is evaluated as a CURIE. If it is a valid CURIE, the resulting URI is used; otherwise, the value is processed as a URI.

TERMorCURIEorAbsURI

- If the value is an NCName, then it is evaluated as a term according to General Use of Terms in Attributes [p.28] . Note that this step may mean that the value is to be ignored.
- If the value is a valid CURIE, then the resulting URI is used.
- If the value is an absolute URI, that value is used.
- Otherwise, the value is ignored.

Note that it is possible for all values in an attribute to be ignored. When that happens, the attribute *must* be treated as if it were empty.

For example, the full URI for Albert Einstein on DBPedia is:

```
http://dbpedia.org/resource/Albert_Einstein
```

This can be shortened by authors to make the information easier to manage, using a CURIE. The first step is for the author to create a prefix mapping that links a prefix to some leading segment of the URI. In RDFa these mappings are expressed using the XML namespace syntax:

```
<div prefix="db: http://dbpedia.org/">
  ...
</div>
```

Once the prefix has been established, an author can then use it to shorten a URI as follows:

```
<div prefix="db: http://dbpedia.org/">
  <div about="db:resource/Albert_Einstein">
    ...
  </div>
</div>
```

The author is free to split the URI at any point, as long as it begins at the left end. However, since a common use of CURIEs is to make available libraries of terms and values, the prefix will usually be mapped to some common segment that provides the most re-use, often provided by those who manage the library of terms. For example, since DBPedia contains an enormous list of resources, it is more efficient to create a prefix mapping that uses the base location of the resources:

```
<div prefix="dbr: http://dbpedia.org/resource/">
  <div about="dbr:Albert_Einstein">
    ...
  </div>
  <div about="dbr:Baruch_Spinoza">
    ...
  </div>
</div>
```

Note that it is generally considered a bad idea to use relative paths in prefix declarations. Since it is possible that an author may ignore this guidance, it is further possible that the URI obtained from a CURIE is relative. However, since all URIs must be resolved relative to base [p.23] before being used to create triples, the use of relative paths should not have any effect on processing.

7.4.1 Scoping of Prefix Mappings

CURIE prefix mappings are defined on the current element and its descendants. The inner-most mapping for a given prefix takes precedence. For example, the URIs expressed by the following two CURIEs are different, despite the common prefix, because the prefix mappings are locally scoped:

```
<div prefix="dbr: http://dbpedia.org/resource/">
  <div about="dbr:Albert_Einstein">
    ...
  </div>
</div>
<div prefix="dbr: http://someotherdb.org/resource/">
  <div about="dbr:Albert_Einstein">
    ...
  </div>
</div>
```

7.4.2 General Use of CURIEs in Attributes

There are a number of ways that attributes make use of CURIEs, and they need to be dealt with differently. These are:

1. An attribute may allow one or more values that are a mixture of TERMs, CURIEs, and absolute URIs.
2. An attribute may allow one or more values that are a mixture of CURIEs and URIs. In this case any value that is not a CURIE, as outlined in section CURIE Syntax Definition [p.20] , will be processed as a URI.
3. If the value *is* surrounded by square brackets, then the content within the brackets is always evaluated according to the rules in CURIE Syntax Definition [p.20] - and if that content is not a CURIE, then the content *must* be ignored.

An empty attribute value (e.g., `typeof=''`) is *still* a CURIE, and is processed as such. The rules for this processing are defined in Sequence [p.30] . Specifically, however, an empty attribute value is *never* treated as a relative URI by this specification.

An example of an attribute that can contain a CURIEorURI is @about [p.18] . To express a URI directly, an author might do this:

```
<div about="http://dbpedia.org/resource/Albert_Einstein">
  ...
</div>
```

whilst to express the URL above as a CURIE they would do this:

```
<div about="dbr:Albert_Einstein">
  ...
</div>
```

The author could also use a safe CURIE, as follows:

```
<div about="[dbr:Albert_Einstein]">
  ...
</div>
```

Since non-CURIE values *must* be ignored, the following value in @about [p.18] would *not* set a new subject, since @about [p.18] does not permit the use of TERM [p.53] s, and the CURIE has no prefix separator.

```
<div about="[Albert_Einstein]">
  ...
</div>
```

However, this markup *would* set a subject, since it is not a CURIE, but a valid relative URI:

```
<div about="Albert_Einstein">
  ...
</div>
```

Note that several RDFa attributes are able to also take TERMS [p.53] as their value. This is discussed in the next section.

7.4.3 General Use of Terms in Attributes

Some RDFa attributes have a datatype that permits a *term* to be referenced. RDFa defines the syntax of a term as:

```
term     ::=  NCName
```

When an RDFa attribute permits the use of a term, and the value being evaluated matches the production for term above, it is transformed to a URI using the following logic:

- Check if the `term` matches an item in the list of local term mappings [p.24] . First compare against the list *case-sensitively*, and if there is no match then compare *case-insensitively*. If there is a match, use the associated URI.
- Otherwise, if there is a local default vocabulary [p.24] the URI is obtained by concatenating that value and the `term`.
- Finally, if there is no local default vocabulary [p.24] , the `term` has no associated URI and *must* be ignored.

7.4.4 Use of CURIEs in Specific Attributes

The general rules discussed in the previous sections apply to the RDFa attributes in the following ways:

- @about [p.18] and @resource [p.19] support either a SafeCURIE, a CURIE, or a URI.
- @href [p.19] and @src [p.19] are as defined in the Host Language (e.g., XHTML), and support only a URI.
- @profile [p.19] and @vocab [p.19] also only support a URI.
- @property [p.19] , @datatype [p.19] , @typeof [p.19] , @rel [p.19] , and @rev [p.19] support Terms, CURIEs, or URIs.

Any value that matches a defined term *must* be expanded into a reference to the corresponding URI. For example in [*XHTML-RDFA [p.58]*] the following examples:

```
<link rel="next" href="http://example.org/page2.html" />
<link rel="xhv:next" href="http://example.org/page2.html" />
```

would each generate the following triple:

```
<> <http://www.w3.org/1999/xhtml/vocab#next> <http://example.org/page2.html> .
```

7.4.5 Referencing Blank Nodes

In RDFa, it is possible to establish relationships using various types of resource references, including bnode [p.17] s. If a subject or object is defined using a CURIE, and that CURIE explicitly names a bnode [p.17] , then a Conforming Processor *must* create the bnode [p.17] when it is encountered during parsing. The RDFa Processor *must* also ensure that no bnode [p.17] created automatically (as a result of chaining [p.24]) has a name that collides with a bnode [p.17] that is defined by explicit reference in a CURIE.

Consider the following example:

```
<link about="_:john" rel="foaf:mbox"
  href="mailto:john@example.org" />
<link about="_:sue" rel="foaf:mbox"
  href="mailto:sue@example.org" />
<link about="_:john" rel="foaf:knows"
  resource="_:sue" />
```

In the above fragment, two bnodes [p.17] are explicitly created as the subject of triples. Those bnodes [p.17] are then referenced to demonstrate the relationship between the parties. After processing, the following triples will be generated:

```
_:john foaf:mbox <mailto:john@example.org> .
_:sue foaf:mbox <mailto:sue@example.org> .
_:john foaf:knows _:sue .
```

RDFa Processors use, internally, implementation-dependent identifiers for bnodes. When triples are *retrieved*, new bnode indentifiers are used, which usually bear no relation to the original identifiers. However, implementations do ensure that these generated bnode identifiers are consistent: each bnode will have its own identifier, all references to a particular bnode will use the same identifier, and different bnodes will have different identifiers.

As a special case, `_:` is also a valid reference for *one* specific bnode [p.17] .

7.5 Sequence

Processing would normally begin after the document to be parsed has been completely loaded. However, there is no requirement for this to be the case, and it is certainly possible to use a stream-based approach, such as SAX [*SAX [p.59]*] to extract the RDFa information. However, if some approach other than the DOM traversal technique defined here is used, it is important to ensure that Host Language-specific processing rules are applied (e.g., XHTML+RDFa [*XHTML-RDFA [p.58]*] indicates the `base` element can be used, and `base` will affect the interpretation of URIs in `meta` or `link` elements even if those elements are before the `base` element in the stream).

At the beginning of processing, an initial *evaluation context* is created, as follows:

- the base [p.23] is set to the URL of the document (or another value specified in a language specific manner such as the HTML `base` element);
- the parent subject [p.23] is set to the base [p.23] value;
- the parent object [p.23] is set to null;
- the list of incomplete triples [p.24] is empty;
- the language [p.23] is set to null.
- the list of URI mappings [p.23] is empty (or a list defined in the Host Language-defined default RDFa Profile).
- the term mappings [p.23] is set to null (or a list defined in the Host Language-defined default RDFa Profile).
- the default vocabulary [p.23] is set to null (or a list defined in the Host Language-defined default RDFa Profile).

Processing begins by applying the processing rules below to the document object, in the context of this initial evaluation context [p.30] . All elements in the tree are also processed according to the rules described below, depth-first, although the evaluation context [p.30] used for each set of rules will be based on previous rules that may have been applied.

This specification defines processing rules for optional attributes that may not be present in all Host Languages (e.g., `@href`). If these attributes are not supported in the Host Language, then the corresponding processing rules are not relevant for that language.

The working group as not reached consensus as to whether to include the optional attributes in this specification, or whether to have them defined in the relevant Host Language specifications.

The processing rules are:

1. First, the local values are initialized, as follows:
   - the skip element [p.24] flag is set to 'false';
   - new subject [p.24] is set to null;
   - current object resource [p.24] is set to null;
   - the local list of URI mappings [p.24] is set to the list of URI mappings from the evaluation context [p.30] ;
   - the local list of incomplete triples [p.24] is set to null;
   - the current language [p.35] value is set to the language [p.23] value from the evaluation context [p.30] .
   - the local term mappings [p.24] is set to the term mappings [p.23] from the evaluation context [p.30] .
   - the local default vocabulary [p.24] is set to the default vocabulary [p.23] from the evaluation context [p.30] .

   Note that some of the local variables are temporary containers for values that will be passed to descendant elements via an evaluation context [p.30] . In some cases the containers will have the same name, so to make it clear which is being acted upon in the following steps, the local version of an item will generally be referred to as such.
2. Next the current element [p.31] is parsed for any updates to the local term mappings [p.24] , default vocabulary [p.23] , and local list of URI mappings [p.24] via @profile [p.19] . If @profile [p.19] is present, its value is processed as defined in RDFa Profiles [p.51] . If any referenced RDFa Profile is not recognized [p.52] , then the current element [p.31] and its children *must not* place any triples in the default graph [p.17] .
   Any new terms or URI mappings are merged into respective local lists. They are in effect for this element and for its children.
3. Next the *current element* is examined for any change to the default vocabulary [p.23] via @vocab [p.19] . If @vocab [p.19] is present and contains a value, its value updates the local default vocabulary [p.24] . If the value is empty, then the local default vocabulary [p.24] *must* be reset to the Host Language defined default.
   A Host Language is not required to define a default vocabulary. In such a case, setting @vocab [p.19] to the empty value has the effect of clearing the local default vocabulary [p.24] .
4. Next, the current element [p.31] is then examined for *URI mapping*s and these are added to the local list of URI mappings [p.24] . Note that a URI mapping [p.31] will simply overwrite any current mapping in the list that has the same name;
   Mappings are defined via @prefix [p.20] . For backward compatibility, some Host Languages *may* also permit the definition of mappings via @xmlns [p.19] . In this case, the value to be mapped is set by the XML namespace prefix, and the value to map is the value of the attribute â a URI. Regardless of how the mapping is declared, the value to be mapped *must* be converted to lower case, and the URI is not processed in any way; in particular if it is a relative path it *must not* be resolved against the current base [p.23] . Authors *should not* use relative paths as the URI.
5. The current element [p.31] is also parsed for any language information, and if present, current language [p.35] is set accordingly;
   Host Languages that incorporate RDFa *may* provide a mechanism for specifying the natural

language of an element and its contents (e.g., XML provides the general-purpose XML attribute @xml:lang).

6. If the current element [p.31] contains no @rel [p.19] or @rev [p.19] attribute, then the next step is to establish a value for new subject [p.24] . Any of the attributes that can carry a resource can set new subject [p.24] ;
   new subject [p.24] is set to the URI obtained from the first match from the following rules:
   - by using the URI from @about [p.18] , if present, obtained according to the section on CURIE and URI Processing [p.25] ;
   - *otherwise*, by using the URI from @src [p.19] , if present, obtained according to the section on CURIE and URI Processing [p.25] .
   - *otherwise*, by using the URI from @resource [p.19] , if present, obtained according to the section on CURIE and URI Processing [p.25] ;
   - *otherwise*, by using the URI from @href [p.19] , if present, obtained according to the section on CURIE and URI Processing [p.25] .

   If no URI is provided by a resource attribute, then the first match from the following rules will apply:

   - if @typeof [p.19] is present, then new subject [p.24] is set to be a newly created bnode [p.17] .
   - *otherwise*, if parent object [p.23] is present, new subject [p.24] is set to the value of parent object [p.23] . Additionally, if @property [p.19] is *not* present then the skip element [p.24] flag is set to 'true';
7. If the current element [p.31] *does* contain a @rel [p.19] or @rev [p.19] attribute, then the next step is to establish *both* a value for new subject [p.24] and a value for current object resource [p.24] :
   new subject [p.24] is set to the URI obtained from the first match from the following rules:
   - by using the URI from @about [p.18] , if present, obtained according to the section on CURIE and URI Processing [p.25] ;
   - *otherwise*, by using the URI from @src [p.19] , if present, obtained according to the section on CURIE and URI Processing [p.25] .

   If no URI is provided then the first match from the following rules will apply:

   - if @typeof [p.19] is present, then new subject [p.24] is set to be a newly created bnode [p.17] ;
   - *otherwise*, if parent object [p.23] is present, new subject [p.24] is set to that.

   Then the current object resource [p.24] is set to the URI obtained from the first match from the following rules:

   - by using the URI from @resource [p.19] , if present, obtained according to the section on CURIE and URI Processing [p.25] ;
   - *otherwise*, by using the URI from @href [p.19] , if present, obtained according to the section on CURIE and URI Processing [p.25] .

Note that final value of the current object resource [p.24] will either be null (from initialization) or a full URI.

8. If in any of the previous steps a new subject [p.24] was set to a non-null value, it is now used to provide a subject for type values;
   One or more 'types' for the new subject [p.24] can be set by using @typeof [p.19] . If present, the attribute may contain one or more URIs, obtained according to the section on URI and CURIE Processing [p.25] , each of which is used to generate a triple as follows:
   subject
   new subject [p.24]
   predicate
   http://www.w3.org/1999/02/22-rdf-syntax-ns#type
   object
   full URI of 'type'
   Note that none of this block is executed if there is no new subject [p.24] value, i.e., new subject [p.24] remains null.
9. If in any of the previous steps a current object resource [p.24] was set to a non-null value, it is now used to generate triples:
   Predicates for the current object resource [p.24] can be set by using one or both of the @rel [p.19] and @rev [p.19] attributes:
   - If present, @rel [p.19] may contain one or more URIs, obtained according to the section on CURIE and URI Processing [p.25] each of which is used to generate a triple as follows:
     subject
     new subject [p.24]
     predicate
     full URI
     object
     current object resource [p.24]
   - If present, @rev [p.19] may contain one or more URIs, obtained according to the section on CURIE and URI Processing [p.25] each of which is used to generate a triple as follows:
     subject
     current object resource [p.24]
     predicate
     full URI
     object
     new subject [p.24]
10. If however current object resource [p.24] was set to null, but there are predicates present, then they must be stored as incomplete triple [p.23] s, pending the discovery of a subject that can be used as the object. Also, current object resource [p.24] should be set to a newly created bnode [p.17] ;
    Predicates for incomplete triple [p.23] s can be set by using one or both of the @rel [p.19] and @rev [p.19] attributes:
    - If present, @rel [p.19] must contain one or more URIs, obtained according to the section on CURIE and URI Processing [p.25] each of which is added to the local list of

incomplete triples [p.24] as follows:
predicate
full URI
direction
forward

- If present, @rev [p.19] must contain one or more URIs, obtained according to the section on CURIE and URI Processing [p.25] , each of which is added to the local list of incomplete triples [p.24] as follows:
predicate
full URI
direction
reverse

11. The next step of the iteration is to establish any current object literal [p.24] ; Predicates for the current object literal [p.24] can be set by using @property [p.19] . If present, one or more URIs are obtained according to the section on CURIE and URI Processing [p.25] , and then the actual literal value is obtained as follows:
    - as a typed literal [p.15] if @datatype [p.19] is present, does not have an empty value according to the section on CURIE and URI Processing [p.25] , and is not set to `XMLLiteral` in the vocabulary `http://www.w3.org/1999/02/22-rdf-syntax-ns#`.

      The actual literal is either the value of @content [p.19] (if present) *or* a string created by concatenating the value of all descendant text nodes, of the current element [p.31] in turn. The final string includes the datatype URI, as described in [*RDF-CONCEPTS [p.58]*], which will have been obtained according to the section on CURIE and URI Processing [p.25] .

    - as an XML literal [p.49] if @datatype [p.19] is present and is set to `XMLLiteral` in the vocabulary `http://www.w3.org/1999/02/22-rdf-syntax-ns#`.

      The value of the XML literal [p.49] is a string created by serializing to text, all nodes that are descendants of the current element [p.31] , i.e., not including the element itself, and giving it a datatype of `XMLLiteral` in the vocabulary `http://www.w3.org/1999/02/22-rdf-syntax-ns#`. The format of the resulting serialized content is as defined in Exclusive XML Canonicalization Version [*XML-EXC-C14N [p.59]*].

      In order to maintain maximum portability of this literal, any children of the current node that are elements *must* have the current in scope profiles, default vocabulary, prefix mappings, and XML namespace declarations (if any) declared on the serialized element using their respective attributes. Since the child element node could also declare new prefix mappings or XML namespaces, the RDFa Processor *must* be careful to merge these together when generating the serialized element definition. For avoidance of doubt, any re-declarations on the child node *must* take precedence over declarations that were active on the current node.

- otherwise as a plain literal [p.15] .

  Additionally, if there is a value for *current language* then the value of the plain literal [p.15] should include this language information, as described in [*RDF-CONCEPTS [p.58]*]. The actual literal is either the value of @content [p.19] (if present) *or* a string created by concatenating the text content of each of the descendant elements of the current element [p.31] in document order.

The current object literal [p.24] is then used with each predicate to generate a triple as follows:

subject
    new subject [p.24]
predicate
    full URI
object
    current object literal [p.24]

12. If the skip element [p.24] flag is 'false', *and* new subject [p.24] was set to a non-null value, then any incomplete triple [p.23] s *within the current context* should be completed:
    The list of incomplete triples [p.24] from the current evaluation context [p.30] (*not* the local list of incomplete triples [p.24]) will contain zero or more predicate URIs. This list is iterated, and each of the predicates is used with parent subject [p.23] and new subject [p.24] to generate a triple. Note that at each level there are *two* lists of incomplete triple [p.23] s; one for the current processing level (which is passed to each child element in the previous step), and one that was received as part of the evaluation context [p.30] . It is the latter that is used in processing during this step.
    Note that each incomplete triple [p.23] has a *direction* value that it used to determine what will become the subject, and what will become the object, of each generated triple:
    - If direction [p.35] is 'forward' then the following triple is generated:
      subject
          parent subject [p.23]
      predicate
          the predicate from the iterated incomplete triple [p.23]
      object
          new subject [p.24]
    - If direction [p.35] is not 'forward' then this is the triple generated:
      subject
          new subject [p.24]
      predicate
          the predicate from the iterated incomplete triple [p.23]
      object
          parent subject [p.23]
13. Next, all elements that are children of the current element [p.31] are processed using the rules described here, using a new evaluation context [p.30] , initialized as follows:
    - If the skip element [p.24] flag is 'true' then the new evaluation context [p.30] is a copy of the current context that was passed in to this level of processing, with the language [p.23] and list of URI mappings [p.23] values replaced with the local values;

- Otherwise, the values are:
  - the base [p.23] is set to the base [p.23] value of the current evaluation context [p.30] ;
  - the parent subject [p.23] is set to the value of new subject [p.24] , if non-null, *or* the value of the parent subject [p.23] of the current evaluation context [p.30] ;
  - the parent object [p.23] is set to value of current object resource [p.24] , if non-null, *or* the value of new subject [p.24] , if non-null, *or* the value of the parent subject [p.23] of the current evaluation context [p.30] ;
  - the list of URI mappings [p.23] is set to the local list of URI mappings [p.24] ;
  - the list of incomplete triples [p.24] is set to the local list of incomplete triples [p.24] ;
  - language [p.23] is set to the value of current language [p.35] .
  - the term mappings [p.23] is set to the value of the local term mappings [p.24] .
  - the default vocabulary [p.23] is set to the value of the local default vocabulary [p.24] .

7.6 Processor Status

The processing rules covered in the previous section are designed to extract as many triples as possible from a document. The RDFa Processor is designed to continue processing, even in the event of errors. For example, failing to resolve a prefix mapping or term [p.53] would result in the RDFa Processor skipping the generation of a triple and continuing with document processing. There are cases where knowing each RDFa Processor warning or error would be beneficial to authors. The processor graph [p.17] is designed as a mechanism to capture all informational, warning, and error messages as triples from the RDFa Processor. These status triples may be retrieved and used to aid RDFa authoring or automated error detection.

If an RDFa Processor supports the generation of a processor graph, then it *must* generate a set of triples when the following processing issues occur:

- An ERROR *must* be generated when the document fails to be fully processed as a result of non-conformant host language markup.
- An ERROR *must* be generated when a referenced RDFa Profile is not recognized [p.52] , as described in RDFa Profiles [p.51] .
- A WARNING *must* be generated when a CURIE prefix fails to be resolved.
- A WARNING *must* be generated when a Term fails to be resolved.

Other implementation-specific INFORMATIONAL, WARNING, or ERROR triples *may* be generated by the RDFa Processor.

7.6.1 Accessing the Processor Graph

Accessing the processor graph [p.17] may be accomplished in a variety of ways and is dependent on the type of RDFa Processor and access method that the developer is utilizing.

SAX-based processors or processors that utilize function or method callbacks to report the generation of triples are classified as *event-based RDFa Processor*s. For Event-based RDFa Processors, the software *must* allow the developer to register a function or callback that is called when a triple is generated for the processor graph [p.17] . The callback *may* be the same as the one that is used for the default graph [p.17] as long as it can be determined if a generated triple belongs in the processor graph [p.17] or the default graph [p.17] .

A *whole-graph RDFa Processor* is defined as any RDFa Processor that processes the entire document and only allows developer access to the triples after processing has completed. RDFa Processors that typically fall into this category express their output via a single call using RDF/XML, N3, TURTLE, or N-Triples notation. For whole-graph RDFa Processors, the software *must* allow the developer to specify if they would like to retrieve the default graph [p.17] , the processor graph [p.17] , or both graphs as a single, combined graph from the RDFa Processor. If the graph preference is not specified, the default graph [p.17] *must* be returned.

An *web service RDFa Processor* is defined as any RDFa Processor that is capable of processing a document by performing an HTTP GET, POST or similar action on an RDFa Processor URL. For this class of RDFa Processor, the software *must* allow the caller to specify if they would like to retrieve the default graph [p.17] , the processor graph [p.17] , or both graphs as a single, combined graph from the web service. The `rdfagraph` query parameter *must* be used to specify the value. The allowable values are `default`, `processor` or both values, in any order, separated by a comma character. If the graph preference is not specified, the default graph [p.17] *must* be returned.

8. RDFa Processing in detail

This section provides an in-depth examination of the processing steps described in the previous section. It also includes examples which may help clarify some of the steps involved.

The key to processing is that a triple is generated whenever a predicate/object combination is detected. The actual triple generated will include a subject that may have been set previously, so this is tracked in the current evaluation context [p.30] and is called the parent subject [p.23] . Since the subject will default to the current document if it hasn’t been set explicitly, then a predicate/object combination is always enough to generate one or more triples.

The attributes for setting a predicate are @rel [p.19] , @rev [p.19] and @property [p.19] , whilst the attributes for setting an object are @resource [p.19] , @href [p.19] , @content [p.19] , and @src [p.19] . @typeof [p.19] is unique in that it sets *both* a predicate and an object at the same time (and also a subject when it appears in the absence of other attributes that would set a subject). Inline content might also set an object, if @content [p.19] is not present, but @property [p.19] is present.

There are many examples in this section. The examples are all written using XHTML+RDFa. However, the explanations are relevant regardless of the Host Language.

8.1 Changing the evaluation context

8.1.1 Setting the current subject

When triples are created they will always be in relation to a subject resource which is provided either by new subject [p.24] (if there are rules on the current element that have set a subject) or parent subject [p.23] , as passed in via the evaluation context [p.30] . This section looks at the specific ways in which these values are set. Note that it doesn't matter how the subject is set, so in this section we use the idea of the *current subject* which may be *either* new subject [p.24] or parent subject [p.23] .

8.1.1.1 The current document

When parsing begins, the current subject [p.38] will be the URI of the document being parsed, or a value as set by a Host Language-provided mechanism (e.g., the `base` element in (X)HTML). This means that by default any metadata found in the document will concern the document itself:

```
<html profile="http://www.example.org/vocab-rdf-dc.html">
  <head>
    <title>Jo's Friends and Family Blog</title>
    <link rel="foaf:primaryTopic" href="#bbq" />
    <meta property="dcterms:creator" content="Jo" />
  </head>
  <body>
    ...
  </body>
</html>
```

This would generate the following triples:

```
<> foaf:primaryTopic <#bbq> .
<> dcterms:creator "Jo" .
```

It is possible for the data to appear elsewhere in the document:

```
<html profile="http://www.example.org/vocab-rdf-dc.html">
  <head>
    <title>Jo's Blog</title>
  </head>
  <body>
    <h1><span property="dcterms:creator">Jo</span>'s blog</h1>
    <p>
      Welcome to my blog.
    </p>
  </body>
</html>
```

which would still generate the triple:

```
<> dcterms:creator "Jo" .
```

In (X)HTML the value of `base` may change the initial value of current subject [p.38] :

```
<html profile="http://www.example.org/vocab-rdf-dc.html">
  <head>
    <base href="http://www.example.org/jo/blog" />
    <title>Jo’s Friends and Family Blog</title>
    <link rel="foaf:primaryTopic" href="#bbq" />
    <meta property="dcterms:creator" content="Jo" />
  </head>
  <body>
    ...
  </body>
</html>
```

An RDFa Processor should now generate the following triples, regardless of the URL from which the document is served:

```
<http://www.example.org/jo/blog> foaf:primaryTopic <#bbq> .
<http://www.example.org/jo/blog> dcterms:creator "Jo" .
```

8.1.1.2 Using @about

As processing progresses, any @about [p.18] attributes will change the current subject [p.38] . The value of @about [p.18] is a URI or a CURIE. If it is a relative URI then it needs to be resolved against the current base [p.23] value. To illustrate how this affects the statements, note in this markup how the properties inside the (X)HTML `body` element become part of a new calendar event object, rather than referring to the document as they do in the head of the document:

```
<html profile="http://www.example.org/vocab-rdf-dc.html"
      prefix="cal: http://www.w3.org/2002/12/cal/ical#">
  <head>
    <title>Jo’s Friends and Family Blog</title>
    <link rel="foaf:primaryTopic" href="#bbq" />
    <meta property="dcterms:creator" content="Jo" />
  </head>
  <body>
    <p about="#bbq" typeof="cal:Vevent">
      I’m holding
      <span property="cal:summary">
        one last summer barbecue
      </span>,
      on
      <span property="cal:dtstart" content="2015-09-16T16:00:00-05:00"
            datatype="xsd:dateTime">
        September 16th at 4pm
      </span>.
    </p>
  </body>
</html>
```

With this markup an RDFa Processor will generate the following triples:

```
<> foaf:primaryTopic <#bbq> .
<> dcterms:creator "Jo" .
<#bbq> rdf:type cal:Vevent .
<#bbq> cal:summary "one last summer barbecue" .
<#bbq> cal:dtstart "2015-09-16T16:00:00-05:00"^^xsd:dateTime .
```

Other kinds of resources can be used to set the current subject [p.38] , not just references to web-pages. Although not advised, email addresses might be used to represent a person:

```
John knows
<a about="mailto:john@example.org"
  rel="foaf:knows" href="mailto:sue@example.org">Sue</a>.

Sue knows
<a about="mailto:sue@example.org"
  rel="foaf:knows" href="mailto:jim@example.org">Jim</a>.
```

This should generate the following triples:

```
<mailto:john@example.org> foaf:knows <mailto:sue@example.org> .
<mailto:sue@example.org> foaf:knows <mailto:jim@example.org> .
```

Similarly, authors may make statements about images:

```
<div about="photo1.jpg">
  this photo was taken by
  <span property="dcterms:creator">Mark Birbeck</span>
</div>
```

which should generate the following triples:

```
<photo1.jpg> dcterms:creator "Mark Birbeck" .
```

8.1.1.3 Using @src

If @about [p.18] is not present, then @src [p.19] is next in priority order, for setting the subject of a statement. A typical use would be to indicate the licensing type of an image:

```
<img src="photo1.jpg" rel="license"
     resource="http://creativecommons.org/licenses/by/2.0/" />
```

Since there is no difference between @src [p.19] and @about [p.18] , then the information expressed in the last example in the section on @about [p.18] (the *creator* of an image), could be expressed as follows:

```
<img src="photo1.jpg"
  rel="license" resource="http://creativecommons.org/licenses/by/2.0/"
  property="dcterms:creator" content="Mark Birbeck"
/>
```

Since normal chaining rules will apply, the image URL can also be used to complete hanging triples:

```
<div about="http://www.blogger.com/profile/1109404" rel="foaf:img">
  <img src="photo1.jpg"
    rel="license" resource="http://creativecommons.org/licenses/by/2.0/"
    property="dcterms:creator" content="Mark Birbeck"
  />
</div>
```

The complete markup yields three triples:

```
<http://www.blogger.com/profile/1109404> foaf:img <photo1.jpg> .
<photo1.jpg> xhv:license <http://creativecommons.org/licenses/by/2.0/> .
<photo1.jpg> dcterms:creator "Mark Birbeck" .
```

8.1.1.4 Creating a new item with @typeof

Whilst @about [p.18] explicitly creates a new context for statements, @typeof [p.19] does so implicitly. @typeof [p.19] works differently to other ways of setting a predicate since the predicate is always `rdf:type`, which means that the processor only requires one attribute, the value of the type.

Since @typeof [p.19] is setting the type of an item, this means that if no item exists one should automatically be created. This involves generating a new bnode [p.17] , and is examined in more detail below; it is mentioned here because the bnode [p.17] used by the new item will become the subject for further statements.

For example, an author may wish to create markup for a person using the FOAF vocabulary, but without having a clear identifier for the item:

```
<div typeof="foaf:Person">
  <span property="foaf:name">Albert Einstein</span>
  <span property="foaf:givenName">Albert</span>
</div>
```

This markup would cause a bnode [p.17] to be created which has a 'type' of `foaf:Person`, as well as name and given name properties:

```
_:a rdf:type foaf:Person .
_:a foaf:name "Albert Einstein" .
_:a foaf:givenName "Albert" .
```

A bnode [p.17] is simply a unique identifier that is only available to the processor, not to any external software. By generating values internally, the processor is able to keep track of properties for `_:a` as being distinct from `_:b`. But by not exposing these values to any external software, it is possible to have complete control over the identifier, as well as preventing further statements being made about the item.

8.1.1.5 Determining the subject with neither @about nor @typeof

As described in the previous two sections, @about [p.18] will always take precedence and mark a new subject, but if no @about [p.18] value is available then @typeof [p.19] will do the same job, although using an implied identifier, i.e., a bnode [p.17] .

But if neither @about [p.18] or @typeof [p.19] are present, there are a number of ways that the subject could be arrived at. One of these is to 'inherit' the subject from the containing statement, with the value to be inherited set either explicitly, or implicitly.

8.1.1.5.1 Inheriting subject from @resource

The most usual way that an inherited subject might get set would be when the parent statement has an object that is a resource. Returning to the earlier example, in which the long name for the German_Empire was added, the following markup was used:

```
<div about="http://dbpedia.org/resource/Albert_Einstein">
  <span property="foaf:name">Albert Einstein</span>
  <span property="dbp:dateOfBirth" datatype="xsd:date">1879-03-14</span>
  <div rel="dbp:birthPlace" resource="http://dbpedia.org/resource/German_Empire" />
  <span about="http://dbpedia.org/resource/German_Empire"
    property="dbp:conventionalLongName">the German Empire</span>
</div>
```

In an earlier illustration the subject and object for the German Empire were elided by removing the @resource [p.19] , relying on the @about [p.18] to set the object:

```
<div about="http://dbpedia.org/resource/Albert_Einstein">
  <span property="foaf:name">Albert Einstein</span>
  <span property="dbp:dateOfBirth" datatype="xsd:date">1879-03-14</span>
  <div rel="dbp:birthPlace">
    <span about="http://dbpedia.org/resource/German_Empire"
      property="dbp:conventionalLongName">the German Empire</span>
  </div>
</div>
```

but it is also possible for authors to achieve the same effect by removing the @about [p.18] and leaving the @resource [p.19] :

```
<div about="http://dbpedia.org/resource/Albert_Einstein">
  <span property="foaf:name">Albert Einstein</span>
  <span property="dbp:dateOfBirth" datatype="xsd:date">1879-03-14</span>
  <div rel="dbp:birthPlace" resource="http://dbpedia.org/resource/German_Empire">
    <span property="dbp:conventionalLongName">the German Empire</span>
  </div>
</div>
```

In this situation, all statements that are 'contained' by the object resource representing the German Empire (the value in @resource [p.19]) will have the same subject, making it easy for authors to add additional statements:

```
<div about="http://dbpedia.org/resource/Albert_Einstein">
  <span property="foaf:name">Albert Einstein</span>
  <span property="dbp:dateOfBirth" datatype="xsd:date">1879-03-14</span>
  <div rel="dbp:birthPlace" resource="http://dbpedia.org/resource/German_Empire">
    <span property="dbp:conventionalLongName">the German Empire</span>
    <span rel="dbp:capital" resource="http://dbpedia.org/resource/Berlin" />
  </div>
</div>
```

Looking at the triples that an RDFa Processor would generate, we can see that we actually have two groups of statements; the first group are set to refer to the @about [p.18] that contains them:

```
<http://dbpedia.org/resource/Albert_Einstein> foaf:name "Albert Einstein" .
<http://dbpedia.org/resource/Albert_Einstein> dbp:dateOfBirth "1879-03-14"^^xsd:date .
<http://dbpedia.org/resource/Albert_Einstein> dbp:birthPlace <http://dbpedia.org/resource/German_Empire> .
```

whilst the second group refer to the @resource [p.19] that contains them:

```
<http://dbpedia.org/resource/German_Empire>
  dbp:conventionalLongName "the German Empire" .
<http://dbpedia.org/resource/German_Empire>
  dbp:capital <http://dbpedia.org/resource/Berlin> .
```

Note also that the same principle described here applies to @src [p.19] and @href [p.19] .

8.1.1.5.2 Inheriting an anonymous subject

There will be occasions when the author wants to elide the subject and object as shown above, but is not concerned to name the resource that is common to the two statements (i.e., the object of the first statement, which is the subject of the second). For example, to indicate that Einstein was influenced by Spinoza the following markup could well be used:

```
<div about="http://dbpedia.org/resource/Baruch_Spinoza" rel="dbp:influenced">
  <div about="http://dbpedia.org/resource/Albert_Einstein">
    <span property="foaf:name">Albert Einstein</span>
    <span property="dbp:dateOfBirth" datatype="xsd:date">1879-03-14</span>
  </div>
</div>
```

An RDFa Processor will generate the following triples:

```
<http://dbpedia.org/resource/Baruch_Spinoza>
  dbp:influenced <http://dbpedia.org/resource/Albert_Einstein> .
<http://dbpedia.org/resource/Albert_Einstein> foaf:name "Albert Einstein" .
<http://dbpedia.org/resource/Albert_Einstein> dbp:dateOfBirth "1879-03-14"^^xsd:date .
```

However, an author could just as easily say that Spinoza influenced *something by the name of Albert Einstein, that was born on March 14th, 1879*:

```
<div about="http://dbpedia.org/resource/Baruch_Spinoza" rel="dbp:influenced">
  <div>
    <span property="foaf:name">Albert Einstein</span>
    <span property="dbp:dateOfBirth" datatype="xsd:date">1879-03-14</span>
  </div>
</div>
```

In RDF terms, the item that 'represents' Einstein is *anonymous*, since it has no URI to identify it. However, the item is given an automatically generated bnode [p.17] , and it is onto this identifier that all child statements are attached:

An RDFa Processor will generate the following triples:

```
<http://dbpedia.org/resource/Baruch_Spinoza> dbp:influenced _:a .
_:a foaf:name "Albert Einstein" .
_:a dbp:dateOfBirth "1879-03-14"^^xsd:date .
```

Note that the `div` is superfluous, and an RDFa Processor will create the intermediate object even if the element is removed:

```
<div about="http://dbpedia.org/resource/Baruch_Spinoza" rel="dbp:influenced">
  <span property="foaf:name">Albert Einstein</span>
  <span property="dbp:dateOfBirth" datatype="xsd:date">1879-03-14</span>
</div>
```

An alternative pattern is to *keep* the `div` and move the @rel [p.19] onto it:

```
<div about="http://dbpedia.org/resource/Baruch_Spinoza">
  <div rel="dbp:influenced">
    <span property="foaf:name">Albert Einstein</span>
    <span property="dbp:dateOfBirth" datatype="xsd:date">1879-03-14</span>
  </div>
</div>
```

From the point of view of the markup, this latter layout is to be preferred, since it draws attention to the 'hanging rel'. But from the point of view of an RDFa Processor, all of these permutations need to be supported.

8.2 Completing 'incomplete triples'

When a new subject is calculated, it is also used to complete any incomplete triples that are pending. This situation arises when the author wants to 'chain' a number of statements together. For example, an author could have a statement that Albert Einstein was born in the German Empire:

```
<div about="http://dbpedia.org/resource/Albert_Einstein">
  <span property="foaf:name">Albert Einstein</span>
  <span property="dbp:dateOfBirth" datatype="xsd:date">1879-03-14</span>
  <div rel="dbp:birthPlace" resource="http://dbpedia.org/resource/German_Empire" />
</div>
```

and then a further statement that the 'long name' for this country is *the German Empire*:

```
<span about="http://dbpedia.org/resource/German_Empire"
  property="dbp:conventionalLongName">the German Empire</span>
```

RDFa allows authors to insert this statement as a self-contained unit into other contexts:

```
<div about="http://dbpedia.org/resource/Albert_Einstein">
  <span property="foaf:name">Albert Einstein</span>
  <span property="dbp:dateOfBirth" datatype="xsd:date">1879-03-14</span>
  <div rel="dbp:birthPlace" resource="http://dbpedia.org/resource/German_Empire" />
  <span about="http://dbpedia.org/resource/German_Empire"
    property="dbp:conventionalLongName">the German Empire</span>
</div>
```

But it also allows authors to avoid unnecessary repetition and to 'normalize' out duplicate identifiers, in this case the one for the German Empire:

```
<div about="http://dbpedia.org/resource/Albert_Einstein">
  <span property="foaf:name">Albert Einstein</span>
  <span property="dbp:dateOfBirth" datatype="xsd:date">1879-03-14</span>
  <div rel="dbp:birthPlace">
    <span about="http://dbpedia.org/resource/German_Empire"
      property="dbp:conventionalLongName">the German Empire</span>
  </div>
</div>
```

When this happens the @rel [p.19] for 'birth place' is regarded as a 'hanging rel' because it has not yet generated any triples, but these 'incomplete triples' are completed by the @about [p.18] that appears on the next line. The first step is therefore to store the two parts of the triple that the RDFa Processor *does* have, but without an object:

```
<http://dbpedia.org/resource/Albert_Einstein> dbp:birthPlace ? .
```

Then as processing continues, the RDFa Processor encounters the subject of the statement about the long name for the German Empire, and this is used in two ways. First it is used to complete the 'incomplete triple':

```
<http://dbpedia.org/resource/Albert_Einstein>
      dbp:birthPlace <http://dbpedia.org/resource/German_Empire> .
```

and second it is used to generate its own triple:

```
<http://dbpedia.org/resource/German_Empire>
      dbp:conventionalLongName "the German Empire" .
```

Note that each occurrence of @about [p.18] will complete any incomplete triples. For example, to mark up the fact that Albert Einstein had both German and American citizenship, an author need only specify one @rel [p.19] value that is then used with multiple @about [p.18] values:

```
<div about="http://dbpedia.org/resource/Albert_Einstein" rel="dbp:citizenship">
  <span about="http://dbpedia.org/resource/German_Empire" />
  <span about="http://dbpedia.org/resource/United_States" />
</div>
```

In this example there is one incomplete triple:

```
<http://dbpedia.org/resource/Albert_Einstein> dbp:citizenship ? .
```

When the processor meets each of the @about [p.18] values, this triple is completed, giving:

```
<http://dbpedia.org/resource/Albert_Einstein>
  dbp:citizenship <http://dbpedia.org/resource/German_Empire> .
<http://dbpedia.org/resource/Albert_Einstein>
  dbp:citizenship <http://dbpedia.org/resource/United_States> .
```

These examples show how @about [p.18] completes triples, but there are other situations that can have the same effect. For example, when @typeof [p.19] creates a new bnode [p.17] (as described above), that will be used to complete any 'incomplete triples'. To illustrate, to indicate that Spinoza influenced both Einstein and Schopenhauer, the following markup could be used:

```
<div about="http://dbpedia.org/resource/Baruch_Spinoza">
  <div rel="dbp:influenced">
    <div typeof="foaf:Person">
      <span property="foaf:name">Albert Einstein</span>
      <span property="dbp:dateOfBirth" datatype="xsd:date">1879-03-14</span>
    </div>
    <div typeof="foaf:Person">
      <span property="foaf:name">Arthur Schopenhauer</span>
      <span property="dbp:dateOfBirth" datatype="xsd:date">1788-02-22</span>
    </div>
  </div>
</div>
```

First the following incomplete triple is stored:

```
<http://dbpedia.org/resource/Baruch_Spinoza> dbp:influenced ? .
```

Then when the RDFa Processor processes the two occurrences of @typeof [p.19] , each generates a bnode [p.17] , which is used to both complete the 'incomplete triple', and to set the subject for further statements:

```
<http://dbpedia.org/resource/Baruch_Spinoza"> dbp:influenced _:a .
_:a rdf:type foaf:Person .
_:a foaf:name "Albert Einstein" .
_:a dbp:dateOfBirth "1879-03-14"^^xsd:date .
<http://dbpedia.org/resource/Baruch_Spinoza"> dbp:influenced _:b .
_:b rdf:type foaf:Person .
_:b foaf:name "Arthur Schopenhauer" .
_:b dbp:dateOfBirth "1788-02-22"^^xsd:date .
```

Triples are also 'completed' if any one of @property [p.19] , @rel [p.19] or @rev [p.19] are present. However, unlike the situation when @about [p.18] or @typeof [p.19] are present, all predicates are attached to one bnode [p.17] :

```
<div about="http://dbpedia.org/resource/Baruch_Spinoza" rel="dbp:influenced">
  <span property="foaf:name">Albert Einstein</span>
  <span property="dbp:dateOfBirth" datatype="xsd:date">1879-03-14</span>
  <div rel="dbp:citizenship">
    <span about="http://dbpedia.org/resource/German_Empire" />
    <span about="http://dbpedia.org/resource/United_States" />
  </div>
</div>
```

This example has two 'hanging rels', and so two situations when 'incomplete triples' will be created. Processing would proceed as follows; first an incomplete triple is stored:

```
<http://dbpedia.org/resource/Baruch_Spinoza> dbp:influenced ? .
```

Next, the RDFa Processor processes the predicate values for `foaf:name`, `dbp:dateOfBirth` and `dbp:citizenship`, but note that only the first needs to 'complete' the 'hanging rel'. So processing `foaf:name` generates two triples:

```
<http://dbpedia.org/resource/Baruch_Spinoza> dbp:influenced _:a .
_:a foaf:name "Albert Einstein" .
```

but processing `dbp:dateOfBirth` generates only one:

```
_:a dbp:dateOfBirth "1879-03-14"^^xsd:date .
```

Processing `dbp:citizenship` also uses the same bnode [p.17] , but note that it also generates its own 'incomplete triple':

```
_:a dbp:citizenship ? .
```

As before, the two occurrences of @about [p.18] complete the 'incomplete triple', once each:

```
_:a dbp:citizenship <http://dbpedia.org/resource/German_Empire> .
_:a dbp:citizenship <http://dbpedia.org/resource/United_States> .
```

The entire set of triples that an RDFa Processor should generate are as follows:

```
<http://dbpedia.org/resource/Baruch_Spinoza> dbp:influenced _:a .
_:a foaf:name "Albert Einstein" .
_:a dbp:dateOfBirth "1879-03-14"^^xsd:date .
_:a dbp:citizenship <http://dbpedia.org/resource/German_Empire> .
_:a dbp:citizenship <http://dbpedia.org/resource/United_States> .
```

8.3 Object resolution

Although objects have been discussed in the previous sections, as part of the explanation of subject resolution, chaining, evaluation contexts, and so on, this section will look at objects in more detail.

There are two types of object, URI resource [p.47] s and literal [p.15] s.

A literal [p.15] object can be set by using @property [p.19] to express a predicate [p.17] , and then using either @content [p.19] , or the inline text of the element that @property [p.19] is on. *Note that the use of @content [p.19] prohibits the inclusion of rich markup in your literal. If the inline content of an element accurately represents the object, then documents should rely upon that rather than duplicating that data using the @content [p.19] .*

A *URI resource* object can be set using one of @rel [p.19] or @rev [p.19] to express a predicate [p.17] , and then *either* using one of @href [p.19] , @resource [p.19] or @src [p.19] to provide an object resource explicitly, *or* using the chaining techniques described above to obtain an object

from a nested subject, or from a bnode [p.17] .

8.3.1 Literal object resolution

An *object literal* will be generated when @property [p.19] is present. @property [p.19] provides the predicate, and the following sections describe how the actual literal to be generated is determined.

8.3.1.1 Plain Literals

@content [p.19] can be used to indicate a plain literal [p.15] , as follows:

```
<meta about="http://internet-apps.blogspot.com/"
      property="dcterms:creator" content="Mark Birbeck" />
```

The plain literal [p.15] can also be specified by using the content of the element:

```
<span about="http://internet-apps.blogspot.com/"
      property="dcterms:creator">Mark Birbeck</span>
```

Both of these examples give the following triple:

```
<http://internet-apps.blogspot.com/> dcterms:creator "Mark Birbeck" .
```

The value of @content [p.19] is given precedence over any element content, so the following would give exactly the same triple as shown above:

```
<span about="http://internet-apps.blogspot.com/"
      property="dcterms:creator" content="Mark Birbeck">John Doe</span>
```

8.3.1.1.1 Language Tags

RDF allows plain literal [p.15] s to have a language tag, as illustrated by the following example from [*RDF-TESTCASES [p.58]*]:

```
<http://example.org/node>
  <http://example.org/property> "chat"@fr .
```

In RDFa the Host Language may provide a mechanism for setting the language tag. In XHTML+RDFa [*XHTML-RDFA [p.58]*], for example, the XML language attribute @xml:lang or the attribute @lang is used to add this information, whether the plain literal is designated by @content [p.19] , or by the inline text of the element:

```
<meta about="http://example.org/node"
  property="ex:property" xml:lang="fr" content="chat" />
```

Note that the language value can be inherited as defined in [*XML10-4e [p.58]*], so the following syntax will give the same triple as above:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      prefix="ex: http://www.example.com/ns/" xml:lang="fr">
  <head>
    <title xml:lang="en">Example</title>
    <meta about="http://example.org/node"
      property="ex:property" content="chat" />
  </head>
  ...
</html>
```

8.3.1.2 Typed literals

Literals can be given a data type using @datatype [p.19] .

This can be represented in RDFa as follows:

```
<span property="cal:dtstart" content="2015-09-16T16:00:00-05:00"
      datatype="xsd:dateTime">
  September 16th at 4pm
</span>.
```

The triples that this markup generates include the datatype after the literal:

```
<> cal:dtstart "2015-09-16T16:00:00-05:00"^^xsd:dateTime .
```

8.3.1.3 *XML Literals*

XML documents cannot contain XML markup in their attributes, which means it is not possible to represent XML within @content [p.19] (the following would cause an XML parser to generate an error):

```
<head>
  <meta property="dcterms:title"
    content="E = mc<sup>2</sup>: The Most Urgent Problem of Our Time" />
</head>
```

RDFa therefore supports the use of normal markup to express XML literals, by using @datatype [p.19] :

```
<h2 property="dcterms:title" datatype="rdf:XMLLiteral">
  E = mc<sup>2</sup>: The Most Urgent Problem of Our Time
</h2>
```

This would generate the following triple, with the XML preserved in the literal:

```
<> dcterms:title "E = mc<sup>2</sup>: The Most Urgent Problem of Our Time"^^rdf:XMLLiteral .
```

This requires that a URI mapping for the prefix `rdf` has been defined.

In the examples given here the `sup` element is actually part of the meaning of the literal, but there will be situations where the extra markup means nothing, and can therefore be ignored. In this situation omitting the @datatype [p.19] attribute or specifying an empty @datatype [p.19] value can be used create a plain literal:

```
<p>You searched for <strong>Einstein</strong>:</p>
<p about="http://dbpedia.org/resource/Albert_Einstein">
  <span property="foaf:name" datatype="">Albert <strong>Einstein</strong></span>
  (b. March 14, 1879, d. April 18, 1955) was a German-born theoretical physicist.
</p>
```

Although the rendering of this page has highlighted the term the user searched for, setting @datatype [p.19] to nothing ensures that the data is interpreted as a plain literal, giving the following triples:

```
<http://dbpedia.org/resource/Albert_Einstein> foaf:name "Albert Einstein" .
```

The value of this XML Literal [p.49] is the exclusive canonicalization [*XML-EXC-C14N [p.59]*] of the RDFa element's value.

8.3.2 URI object resolution

Most of the rules governing the processing of objects that are resources are to be found in the processing descriptions given above, since they are important for establishing the subject. This section aims to highlight general concepts, and anything that might have been missed.

One or more *URI object*s are needed when @rel [p.19] or @rev [p.19] is present. Each attribute will cause triples to be generated when used with @href [p.19] , @resource [p.19] or @src [p.19] , or with the subject value of any nested statement if none of these attributes are present.

@rel [p.19] and @rev [p.19] are essentially the inverse of each other; whilst @rel [p.19] establishes a relationship between the current subject [p.38] as subject, and the current object resource [p.24] as the object, @rev [p.19] does the exact opposite, and uses the current object resource [p.24] as the subject, and the current subject [p.38] as the object.

8.3.2.1 Using @resource to set the object

RDFa provides the @resource [p.19] attribute as a way to set the object of statements. This is particularly useful when referring to resources that are not themselves navigable links:

```
<html profile='http://www.example.org/vocab-rdf-dc.html'>
  <head>
    <title>On Crime and Punishment</title>
    <base href="http://www.example.com/candp.xhtml" />
  </head>
  <body>
    <blockquote about="#q1" rel="dcterms:source" resource="urn:ISBN:0140449132" >
      <p id="q1">
        Rodion Romanovitch! My dear friend! If you go on in this way
        you will go mad, I am positive! Drink, pray, if only a few drops!
      </p>
    </blockquote>
  </body>
</html>
```

The `blockquote` element generates the following triple:

```
<http://www.example.com/candp.xhtml#q1>
  <http://purl.org/dc/terms/source> <urn:ISBN:0140449132> .
```

8.3.2.2 Using @href

If no @resource [p.19] is present, then @href [p.19] is next in priority order, for setting the object.

When a predicate has been expressed using @rel [p.19] , the @href [p.19] on the [RDFa statement]'s element is used to identify the object with a [URI reference]. Its type is a URI:

```
<link about="mailto:john@example.org"
      rel="foaf:knows" href="mailto:sue@example.org" />
```

It's also possible to use both @rel [p.19] and @rev [p.19] at the same time on an element. This is particularly useful when two things stand in two different relationships with each other, for example when a picture is taken *by* Mark, but that picture also *depicts* him:

```
<img src="photo1.jpg" rel="dcterms:creator" rev="foaf:img"
   href="http://www.blogger.com/profile/1109404" />
```

which then yields two triples:

```
<photo1.jpg>
  dcterms:creator <http://www.blogger.com/profile/1109404> .
<http://www.blogger.com/profile/1109404>
  foaf:img <photo1.jpg> .
```

8.3.2.3 Incomplete triples

When a triple predicate has been expressed using @rel [p.19] or @rev [p.19] , but no @href [p.19] , @src [p.19] , or @resource [p.19] exists on the same element, there is a 'hanging rel'. This causes the current subject and all possible predicates (with an indicator of whether they are 'forwards, i.e., @rel [p.19] values, or not, i.e., @rev [p.19] values), to be stored as 'incomplete triples' pending discovery of a subject that could be used to 'complete' those triples.

This process is described in more detail in Completing 'Incomplete Triples' [p.44] .

9. RDFa Profiles

RDFa Profiles are collections of terms, prefix mappings, and/or default vocabulary declarations. A profile is either intrinsically known to the parser, or it is loaded as an external document and processed. These documents *must* be defined in an approved RDFa Host Language (currently XHTML+RDFa [*XHTML-RDFA [p.58]*]). They *may* also be defined in other formats (e.g., JSON-LD [*JSON-LD [p.58]*], RDF/XML [*RDF-SYNTAX-GRAMMAR [p.57]*], or Turtle [*TURTLE [p.59]*]). RDFa Profiles are referenced via @profile [p.19] , and can be used by document authors to simplify the task of adding semantic markup. When an RDFa document includes @profile [p.19] , each URI in the value is processed as follows:

1. If the URI is known to the parser then:
   - all prefix mappings are loaded into the local list of URI mappings [p.24] ;
   - all term mappings are loaded into the local term mappings [p.24] ;
   - any default vocabulary setting is used to update the default vocabulary [p.23] .
2. If the URI is not known to the parser, then attempt to retrieve the content of the URI. If the retrieval fails, the referenced profile is considered to be not *recognized* - stop processing any additional URIs, generate an error (see Processor Status [p.36]), and do not perform any potential mapping updates.

   When a profile is not retrievable, an RDFa Processor will not generate triples from the element the profile is referenced from, nor from any of its children. Consequently, any further processing of the triples would be effectively ignored.

3. Otherwise, parse the retrieved content (according to the processing rules for that document type) and extract the triples into a collection associated with that URI. Note: These triples *must not* be co-mingled with the triples being extracted from any other URI.
4. For every extracted triple that is the common subject of an `rdfa:prefix` and an `rdfa:uri` predicate, create a mapping from the object literal of the `rdfa:prefix` predicate to the object literal of the `rdfa:uri` predicate. Add or update this mapping in the local list of URI mappings [p.24] after transforming the 'prefix' component to lower-case.
5. For every extracted triple that is the common subject of an `rdfa:term` and an `rdfa:uri` predicate, create a mapping from the object literal of the `rdfa:term` predicate to the object literal of the `rdfa:uri` predicate. Add or update this mapping in the local term mappings [p.24] .
6. For an extracted triple that has a predicate of `rdfa:vocabulary`, update the default vocabulary [p.23] to be the object literal of the `rdfa:vocabulary` predicate.

When an RDFa Profile is defined using an RDF serialization, it *must* use the vocabulary terms above to declare the components of the profile.

Once all the URIs in the @profile [p.19] value have been processed, continue with the normal processing of the current element [p.31] .

If any conflict arises between two RDFa Profiles associated with URIs in the @profile [p.19] value, the declaration from the RDFa Profile associated with the left-most URI takes precedence.

It is possible that a referenced RDFa document will in turn reference other documents via @profile [p.19] . Regardless of the depth to which such references might go, only the triples in the top level document effect current processing.

Caching of the relevant triples retrieved via this mechanism is *recommended*. Embedding definitions for well known, stable RDFa Profiles in the implementation is *recommended*.

The object literal for the `rdfa:uri` predicate *must* be an absolute URI. The object literal for the `rdfa:term` predicate *must* match the production for term [p.28] . The object literal for the `rdfa:prefix` predicate must match the production for prefix [p.20] . The object literal for the

`rdfa:vocabulary` predicate *must* be an absolute URI. If one of the objects is not a Literal, does not match its associated production, if there is more than one `rdfa:vocabulary` predicate, or if there are additional `rdfa:uri` or `rdfa:term` predicates sharing the same subject, an RDFa Processor *must not* update the associated mapping.

A. CURIE Datatypes

In order to facilitate the use of CURIEs in markup languages, this specification defines some additional datatypes in the XHTML datatype space (`http://www.w3.org/1999/xhtml/datatypes/`). Markup languages that want to import these definitions can find them in the "datatypes" file for their schema grammar:

- DTD xhtml-datatypes.mod
- XML Schema xhtml-datatypes.xsd

Specifically, the following datatypes are defined:

CURIE
: A single curie [p.20]

CURIEs
: A white space separated list of CURIEs

CURIEorURI
: A CURIE [p.53] or a URI

CURIEorURIs
: A white space separated list of CURIEorURI [p.53] s

SafeCURIE
: A single safe_curie [p.20]

SafeCURIEorCURIEorURI
: A single SafeCURIE [p.53] or CURIEorURI [p.53]

SafeCURIEorCURIEorURIs
: A white space separated list of SafeCURIEorCURIEorURI [p.53] s.

TERM
: A single term [p.28]

TERMorCURIEorAbsURI
: A TERM [p.53] or a CURIEorURI [p.53]

TERMorCURIEorAbsURIs
: A white space separated list of TERMorCURIEorAbsURI [p.53] s

A.1 XML Schema Definition

This section is non-normative.

The following *informative* XML Schema definition for these datatypes is included as an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.w3.org/1999/xhtml/datatypes/"
 xmlns:xh11d="http://www.w3.org/1999/xhtml/datatypes/"
 targetNamespace="http://www.w3.org/1999/xhtml/datatypes/"
 elementFormDefault="qualified"
>
    <xs:simpleType name="CURIE">
        <xs:restriction base="xs:string">
            <xs:pattern value="(([\i-[:]][\c-[:]]*)?:)?.+" />
            <xs:minLength value="1"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="CURIEs">
        <xs:list itemType="xh11d:CURIE"/>
    </xs:simpleType>

    <xs:simpleType name="SafeCURIE">
        <xs:restriction base="xs:string">
            <xs:pattern value="\[(([\i-[:]][\c-[:]]*)?:)?.+\]" />
            <xs:minLength value="3"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="SafeCURIEs">
        <xs:list itemType="xh11d:SafeCURIE"/>
    </xs:simpleType>

    <xs:simpleType name="TERM">
        <xs:list itemType="xs:NCName"/>
    <xs:simpleType>

    <xs:simpleType name="CURIEorURI">
        <xs:union memberTypes="xh11d:CURIE xs:anyURI" />
    </xs:simpleType>

    <xs:simpleType name="CURIEorURIs">
        <xs:list itemType="xh11d:CURIEorURI"/>
    </xs:simpleType>

    <xs:simpleType name="SafeCURIEorCURIEorURI">
        <xs:union memberTypes="xh11d:SafeCURIE xh11d:CURIE xs:anyURI" />
    </xs:simpleType>

    <xs:simpleType name="SafeCURIEorCURIEorURIs">
        <xs:list itemType="xh11d:SafeCURIEorCURIEorURI"/>
    </xs:simpleType>

    <xs:simpleType name='AbsURI'>
        <xs:restriction base='xs:string'>
            <xs:pattern value="[\i-[:]][\c-[:]]+:.+" />
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="TERMorCURIEorAbsURI">
```

```
        <xs:union memberTypes="xh11d:TERM xh11d:CURIE xh11d:AbsURI" />
    </xs:simpleType>

    <xs:simpleType name="TERMorCURIEorAbsURIs">
        <xs:list itemType="xh11d:SafeCURIEorCURIEorAbsURI"/>
    </xs:simpleType>
</xs:schema>
```

A.2 XML DTD Definition

This section is non-normative.

The following *informative* XML DTD definition for these datatypes is included as an example:

```
<!ENTITY % CURIE.datatype "CDATA" >
<!ENTITY % CURIEs.datatype "CDATA" >
<!ENTITY % CURIEorURI.datatype "CDATA" >
<!ENTITY % CURIEorURIs.datatype "CDATA" >
<!ENTITY % SafeCURIEorCURIEorURI.datatype "CDATA" >
<!ENTITY % SafeCURIEorCURIEorURIs.datatype "CDATA" >
<!ENTITY % TERMorCURIEorAbsURI.datatype "CDATA" >
<!ENTITY % TERMorCURIEorAbsURIs.datatype "CDATA" >
```

B. The RDFa Vocabulary for Term Assignments

The RDFa Vocabulary is used to modify RDFa processing behavior and to define the terms usable in the processor graph [p.17] . Its URI is `http://www.w3.org/ns/rdfa#`.

The Vocabulary includes the following term definitions (shown here in Turtle [*TURTLE [p.59]*] format):

```
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix dist: <http://www.w3.org/2007/08/pyRdfa/distiller#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfa: <http://www.w3.org/ns/rdfa#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xhv: <http://www.w3.org/1999/xhtml/vocab#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .


rdfa:PrefixMapping a rdfs:Class ;
     dcterms:description "used in conjunction with the definition of prefixes as the domain of the rdfa:prefix property"@en ;
     rdfs:subClassOf rdfa:VocabularyMapping .

rdfa:TermMapping a rdfs:Class ;
     dcterms:description "used in conjunction with the definition of terms as the domain of the rdfa:term property"@en ;
     rdfs:subClassOf rdfa:VocabularyMapping .

rdfa:VocabularyMapping a rdfs:Class .

<http://www.w3.org/ns/rdfa#> a owl:Ontology ;
     dcterms:creator <http://www.ivan-herman.net/foaf#me> ;
     dcterms:date "2010-07-23"@en ;
     dcterms:description "This document describes the RDFa Vocabulary for Term and Prefix Assignment. The Vocabulary is used to modify RDFa1.1 processing behavior."@en ;
     dcterms:title "RDFa Vocabulary for Term and Prefix Assignment"@en ;
     rdfs:isDefinedBy <http://www.w3.org/TR/rdfa-core/> ;
     rdfs:seeAlso <http://www.w3.org/TR/rdfa-core/> ;
     owl:versionInfo "$Date: 2010/10/21 19:24:17 $"@en .

rdfa:prefix a rdf:Property, owl:DatatypeProperty, owl:FunctionalProperty ;
     dcterms:description "defines a prefix"@en ;
     rdfs:domain rdfa:PrefixMapping ;
     rdfs:range xsd:NMTOKEN .

rdfa:term a rdf:Property, owl:DatatypeProperty, owl:FunctionalProperty ;
     dcterms:description "defines a term"@en ;
     rdfs:domain rdfa:TermMapping ;
     rdfs:range xsd:NMTOKEN .

rdfa:uri a rdf:Property, owl:DatatypeProperty, owl:FunctionalProperty ;
     dcterms:description "defines a uri string to be used either with a term or a prefix definition"@en ;
     rdfs:domain rdfa:VocabularyMapping ;
     rdfs:range xsd:anyURI .
```

```
rdfa:vocabulary a rdf:Property, owl:DatatypeProperty ;
     dcterms:description "defines a uri string to be used as a default vocabulary"@en ;
     rdfs:range xsd:anyURI .

<http://www.w3.org/ns/rdfa.html> xhv:stylesheet <http://www.w3.org/StyleSheets/TR/base.css> .

<http://www.ivan-herman.net/foaf#me> a foaf:Person ;
     rdfs:seeAlso <http://www.ivan-herman.net/foaf> ;
     foaf:mbox <mailto:ivan@w3.org> ;
     foaf:name "Ivan Herman"@en ;
     foaf:title "Semantic Web Activity Lead"@en ;
     foaf:workplaceHomepage <http://www.w3.org> .

 [ a owl:AllDisjointClasses ;
        owl:members ( rdfa:PrefixMapping rdfa:TermMapping )].
```

This vocabulary is also available in an separate file in Turtle format and in RDF/XML format.

These predicates can be used to 'pair' URI strings and their usage in the form of a prefix and/or a term as part of, for example, a blank node. An example can be as follows:

```
[] rdfa:uri     "http://xmlns.com/foaf/0.1/name" ;
   rdfa:prefix  "foaf" .
```

which defines a prefix for the foaf URI.

an RDFa version of the vocabulary should be provided - we still need to write it.

C. Changes

This section is non-normative.

C.1 Major differences with RDFa Syntax 1.0

This specification introduces a number of new features, and extends the behavior of some features from the previous version. The following summary may be helpful to RDFa Processor developers, but is *not* meant to be comprehensive.

- Specific rules about XHTML have been moved into a companion specification: [*XHTML-RDFA [p.58]*].
- Prefix mappings can now be declared using @prefix [p.20] in addition to @xmlns [p.19] .
- Prefix names are now required be be converted to lower-case when the mapping is defined. Prefixes are checked in a case-insensitive manner during CURIE expansion.
- You can now use an Absolute URI everywhere you could previously only use a CURIE (e.g., in the value of @datatype [p.19]).
- There is now a concept of a term [p.53] . This concept has replaced the concept of a 'reserved word'. It is possible now to use a 'term' in most places where you could previously only use a CURIE.
- You can define a default prefix (via @vocab [p.19]) that will be used on non-prefixed CURIEs that are not terms.
- You can define collections of prefix mappings, terms, and a default vocabulary in an external RDFa Profile document.
- When a triple would include an object literal, and there is no explicit datatype attribute, the object literal will now be a 'plain literal'. In version 1.0 it would have been an 'XMLLiteral'.

C.2 Major changes during development of version 1.1

2010-07-26: Added the 'vocabulary' term to the RDFa Profile handling.

2010-02-25: Split into RDFa Core and XHTML+RDFa.

2010-01-01: Applied changes to start production of version 1.1. This includes the re-integration of datatype CURIEorURI.

D. Acknowledgments

This section is non-normative.

At the time of publication, the members of the RDFa Working Group were:

- Ben Adida, Creative Commons (Co-Chair)
- Benjamin Adrian, German Research Center for Artificial Intelligence (DFKI) Gmbh
- Mark Birbeck, webBackplane.com (Invited Expert)
- Abhijit Galkward, Rochester Institute of Technology
- Markus Gylling, DAISY Consortium
- Ivan Herman, W3C
- Toby Inkster (Invited Expert)
- Shane McCarron, Applied Testing and Technology, Inc. (Invited Expert)
- Knud MÃ¶ller (DERI Galway at the National University of Ireland)
- John O'Donovan, British Broadcasting Corporation
- Steven Pemberton, Centre for Mathematics and Computer Science (CWI)
- Jeffrey Sonstein, Rochester Institute of Technology
- Manu Sporny, Digital Bazaar (Co-Chair, Invited Expert)
- Robert Weir, IBM Corporation

E. References

E.1 Normative references

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

[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

[RFC3987]
M. DÃ¼rst; M. Suignard. *Internationalized Resource Identifiers (IRIs).* January 2005. Internet RFC 3987. URL: http://www.ietf.org/rfc/rfc3987.txt

[URI]
T. Berners-Lee; R. Fielding; L. Masinter. *Uniform Resource Identifiers (URI): generic syntax.* January 2005. Internet RFC 3986. URL: http://www.ietf.org/rfc/rfc3986.txt
[XHTML-RDFA]
Shane McCarron; et. al. *XHTML+RDFa 1.1.* 3 August 2010. W3C Working Draft. URL: http://www.w3.org/TR/WD-xhtml-rdfa-20100803
[XML-NAMES]
Richard Tobin; et al. *Namespaces in XML 1.0 (Third Edition).* 8 December 2009. W3C Recommendation. URL: http://www.w3.org/TR/2009/REC-xml-names-20091208/
[XML10-4e]
C. M. Sperberg-McQueen; et al. *Extensible Markup Language (XML) 1.0 (Fourth Edition).* 16 August 2006. W3C Recommendation. URL: http://www.w3.org/TR/2006/REC-xml-20060816/
[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/2004/REC-xmlschema-2-20041028/

E.2 Informative references

[HTML401]
David Raggett; Ian Jacobs; Arnaud Le Hors. *HTML 4.01 Specification.* 24 December 1999. W3C Recommendation. URL: http://www.w3.org/TR/1999/REC-html401-19991224
[JSON-LD]
Manu Sporny, et al. *JSON-LD - Linked Data Expression in JSON* 15 October 2010. Unofficial Draft. URL: http://json-ld.org/spec/latest/
[MICROFORMATS]
Microformats. URL: http://microformats.org
[QNAMES]
N. Walsh. *Using Qualified Names (QNames) as Identifiers in XML Content* 17 March, 2004. TAG Finding. URL: http://www.w3.org/2001/tag/doc/qnameids-2004-03-17
[RDF-CONCEPTS]
Graham Klyne; Jeremy J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax.* 10 February 2004. W3C Recommendation. URL: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210
[RDF-PRIMER]
Frank Manola; Eric Miller. *RDF Primer.* 10 February 2004. W3C Recommendation. URL: http://www.w3.org/TR/2004/REC-rdf-primer-20040210/
[RDF-SYNTAX]
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
[RDF-TESTCASES]
Jan Grant; Dave Beckett. *RDF Test Cases.* 10 February 2004. W3C Recommendation. URL: http://www.w3.org/TR/2004/REC-rdf-testcases-20040210

[RDFA-PRIMER]
Mark Birbeck; Ben Adida. *RDFa Primer.* 14 October 2008. W3C Note. URL: http://www.w3.org/TR/2008/NOTE-xhtml-rdfa-primer-20081014

[RDFA-SYNTAX]
Ben Adida, et al. *RDFa in XHTML: Syntax and Processing.* 14 October 2008. W3C Recommendation. URL: http://www.w3.org/TR/2008/REC-rdfa-syntax-20081014

[RELAXNG-SCHEMA]
Information technology -- Document Schema Definition Language (DSDL) -- Part 2: Regular-grammar-based validation -- RELAX NG. ISO/IEC 19757-2:2008. URI: http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

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

[TURTLE]
David Beckett, Tim Berners-Lee. Turtle: Terse RDF Triple Language January 2008. W3C Team Submission. URL: http://www.w3.org/TeamSubmission/turtle/

[XHTML11]
Murray Altheim; Shane McCarron. *XHTMLâ¢ 1.1 - Module-based XHTML.* 31 May 2001. W3C Recommendation. URL: http://www.w3.org/TR/2001/REC-xhtml11-20010531

[XML-EXC-C14N]
Donald E. Eastlake 3rd; Joseph Reagle; John Boyer. *Exclusive XML Canonicalization Version 1.0.* 18 July 2002. W3C Recommendation. URL: http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/

[XML10]
C. M. Sperberg-McQueen; et al. *Extensible Markup Language (XML) 1.0 (Fifth Edition).* 26 November 2008. W3C Recommendation. URL: http://www.w3.org/TR/2008/REC-xml-20081126/