RDFa 1.1 Primer - Second Edition

Rich Structured Data Markup for Web Documents

W3C Working Group Note 22 August 2013

This version:
http://www.w3.org/TR/2013/NOTE-rdfa-primer-20130822/
Latest published version:
http://www.w3.org/TR/rdfa-primer/
Previous version:
http://www.w3.org/TR/2012/NOTE-rdfa-primer-20120607/
Editors:
Ivan Herman, W3C, ivan@w3.org
Ben Adida, Creative Commons, ben@adida.net
Manu Sporny, Digital Bazaar, msporny@digitalbazaar.com
Mark Birbeck, webBackPlane.com, mark.birbeck@webBackplane.com

Please refer to the **errata** for this document, which may include some normative corrections.

Abstract

The last couple of years have witnessed a fascinating evolution: while the Web was initially built predominantly for human consumption, web content is increasingly consumed by machines which expect some amount of structured data. Sites have started to identify a page's title, content type, and preview image to provide appropriate information in a user's newsfeed when she clicks the "Like" button. Search engines have started to provide richer search results by extracting fine-grained structured details from the Web pages they crawl. In turn, web publishers are producing increasing amounts of structured data within their Web content to improve their standing with search engines.

A key enabling technology behind these developments is the ability to add structured data to HTML pages directly. RDFa (Resource Description Framework in Attributes) is a technique that allows just that: it provides a set of markup attributes to augment the

visual information on the Web with machine-readable hints. In this Primer, we show how to express data using RDFa in HTML, and in particular how to mark up existing human-readable Web page content to express machine-readable data.

This document provides only a Primer to RDFa 1.1. The complete specification of RDFa, with further examples, can be found in the RDFa 1.1 Core [rdfa-core], RDFa Lite [rdfa-lite], XHTML+RDFa 1.1 [xhtml-rdfa], and the HTML5+RDFa 1.1 [rdfa-in-html] specifications.

Status of This Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

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

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

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

1. Introduction

The web is a rich, distributed repository of interconnected information. Until recently, it was organized primarily for human consumption. On a typical web page, an HTML author might specify a headline, then a smaller sub-headline, a block of italicized text, a few paragraphs of average-size text, and, finally, a few single-word links. Web browsers will follow these presentation instructions faithfully. However, only the human mind understands what the headline expresses-a blog post title. The sub-headline indicates the author, the italicized text is the article's publication date, and the single-word links are subject categories. Computers do not understand the nuances between the information; the gap between what programs and humans understand is large.

Figure 1: On the left, what browsers see. On the right, what humans see. Can we bridge the gap so that browsers see more of what we see?

What if the browser, or any machine consumer such as a Web crawler, received

information on the meaning of a web page's visual elements? A dinner party announced on a blog could be copied to the user's calendar, an author's complete contact information to the user's address book. Users could automatically recall previously browsed articles according to categorization labels (i.e., tags). A photo copied and pasted from a web site to a school report would carry with it a link back to the photographer, giving him proper credit. A link shared by a user to his social network contacts would automatically carry additional data pulled from the original web page: a thumbnail, an author, and a specific title. When web data meant for humans is augmented with hints meant for computer programs, these programs become significantly more helpful, because they begin to understand the data's structure.

RDFa allows HTML authors to do just that. Using a few simple HTML attributes, authors can mark up human-readable data with machine-readable indicators for browsers and other programs to interpret. A web page can include markup for items as simple as the title of an article, or as complex as a user's complete social network.

1.1 HTML vs. XHTML

Historically, RDFa 1.0 [rdfa-syntax] was specified only for XHTML. RDFa 1.1 [rdfa-core] is the newer version and the one used in this document. RDFa 1.1 is specified for both XHTML [xhtml-rdfa] and HTML5 [rdfa-in-html]. In fact, RDFa 1.1 also works for any XML-based languages like SVG [SVG11]. This document uses HTML in all of the examples; for simplicity, we use the term "HTML" throughout this document to refer to all of the HTML-family languages.

1.2 Validation

RDFa is based on attributes. While some of the HTML attributes (e.g., `href`, `src`) have been re-used, other RDFa attributes are new. This is important because some of the (X)HTML validators may not properly validate the HTML code until they are updated to recognize the new RDFa attributes. This is rarely a problem in practice since browsers simply ignore attributes that they do not recognize. None of the RDFa-specific attributes have any effect on the visual display of the HTML content. Authors do not have to worry about pages marked up with RDFa looking any different to a human being from pages not marked up with RDFa.

2. Using RDFa

2.1 The Basics of RDFa: RDFa Lite

We begin the introduction to RDFa by using a subset of all the possibilities called RDFa Lite 1.1 [rdfa-lite]. The goal, when defining that subset, was to define a set of possibilities that can be applied to most simple to moderate structured data markup tasks, without burdening the authors with additional complexities. Many Web authors will not need to use more than this minimal subset.

2.1.1 The First Steps: Adding Machine-Readable Hints to Web Pages

Consider Alice, a blogger who publishes a mix of professional and personal articles at http://example.com/alice. We will construct markup examples to illustrate how Alice can use RDFa. A more complete markup of these examples is available on a dedicated page.

2.1.1.1 Hints on Social Networking Sites

Alice publishes a blog and would like to provide extra structural information on her pages like the publication date or the title. She would like to use the terms defined in the Dublin Core vocabulary [DC11], a set of terms that are widely used by, for example, the publishing industry or libraries. Her blog already contain that information:

EXAMPLE 1

```
<html>
<head>
  ...
</head>
<body>
  ...
  <h2>The Trouble with Bob</h2>
  <p>Date: 2011-09-10</p>
  ...
</body>
```

This information is, however, aimed at humans only; computers need some sophisticated methods to extract it. But, using RDFa, she can annotate her page to make the *structured data* clear:

EXAMPLE 2

```
<html>
<head>
  ...
</head>
<body>
  ...
  <h2 property="http://purl.org/dc/terms/title">The Trouble with Bob</h2>
  <p>Date: <span property="http://purl.org/dc/terms/created">2011-09-10</span></p>
  ...
</body>
```

(Notice the markup colored in red: these are the RDFa "hints".)

One useful way to visualize the structured data is:

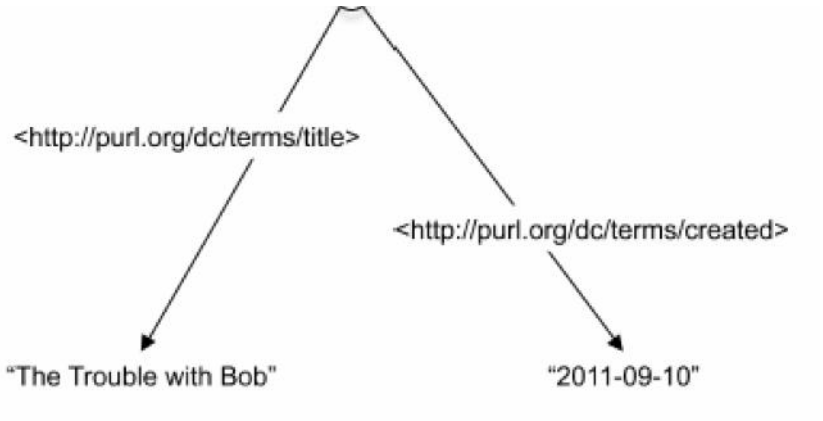


Figure 2: A visualization of the structured data for a blog post with a title of "The Trouble with Bob" and a creation date.

It is worth emphasizing that RDFa uses URLs to identify just about everything. This is why, instead of just using properties like `title` or `created`, we use `http://purl.org/dc/terms/title` and `http://purl.org/dc/terms/created`. The reason behind this design decision is rooted in data portability, consistency, and information sharing. Using URLs removes the possibility for ambiguities in terminology. Without ensuring that there is no ambiguity, the term "title" might mean "the title of a work", "a job title", or "the deed for real-estate property". When each vocabulary term is a URL, a detailed explanation for the vocabulary term is just one click away. It allows anything, humans or machines, to follow the link to find out what a particular vocabulary term means. By using a URL to identify a particular type of title, for example `http://purl.org/dc/terms/title`, both humans and machines can understand that the URL unambiguously refers to the "Date of creation of the resource", such as a web page.

By using URLs as identifiers, RDFa provides a solid way of disambiguating vocabulary terms. It becomes trivial to determine whether or not vocabulary terms used in different documents mean the same thing. If the URLs are the same, the vocabulary terms mean the same thing. It also becomes very easy to create new vocabulary terms and vocabulary documents. If one can publish a document to the Web, one automatically has the power to create a new vocabulary document containing new vocabulary terms.

2.1.1.2 Links with Flavor

The previous example demonstrated how Alice can markup text to make it machine readable. She would also like to mark up the links in a machine-readable way, to express the type of link being described. RDFa lets the publisher add a "flavor", i.e., a label, to an existing clickable link that processors can understand. This makes the same markup help both humans and machines.

In her blog's footer, Alice already declares her content to be freely reusable, as long as she receives due credit when her articles are cited. The HTML includes a link to a Creative Commons [CC-ABOUT] license:

EXAMPLE 3

```
<p>All content on this site is licensed under
   <a href="http://creativecommons.org/licenses/by/3.0/">
     a Creative Commons License</a>. ©2011 Alice Birpemswick.</p>
```

A human clearly understands this sentence, in particular the *meaning* of the link with respect to the current document: it indicates the document's license, the conditions under which the page's contents are distributed. Unfortunately, when Bob visits Alice's blog, his browser sees only a plain link that could just as well point to one of Alice's friends or to her CV. For Bob's browser to understand that this link actually points to the document's licensing terms, Alice needs to add some *flavor*, some indication of what *kind* of link this is.

She can add this flavor using again the `property` attribute. Indeed, when the element contains the `href` (or `src`) attribute, `property` is automatically associated with the value of this attribute rather than the textual content of the `a` element. The value of the attribute is the `http://creativecommons.org/ns#license`, defined by the Creative Commons:

EXAMPLE 4

```
<p>All content on this site is licensed under
   <a property="http://creativecommons.org/ns#license" href="http://creativecommons
     a Creative Commons License</a>. ©2011 Alice Birpemswick.</p>
```

With this small update, Bob's browser will now understand that this link has a flavor: it indicates the blog's license:

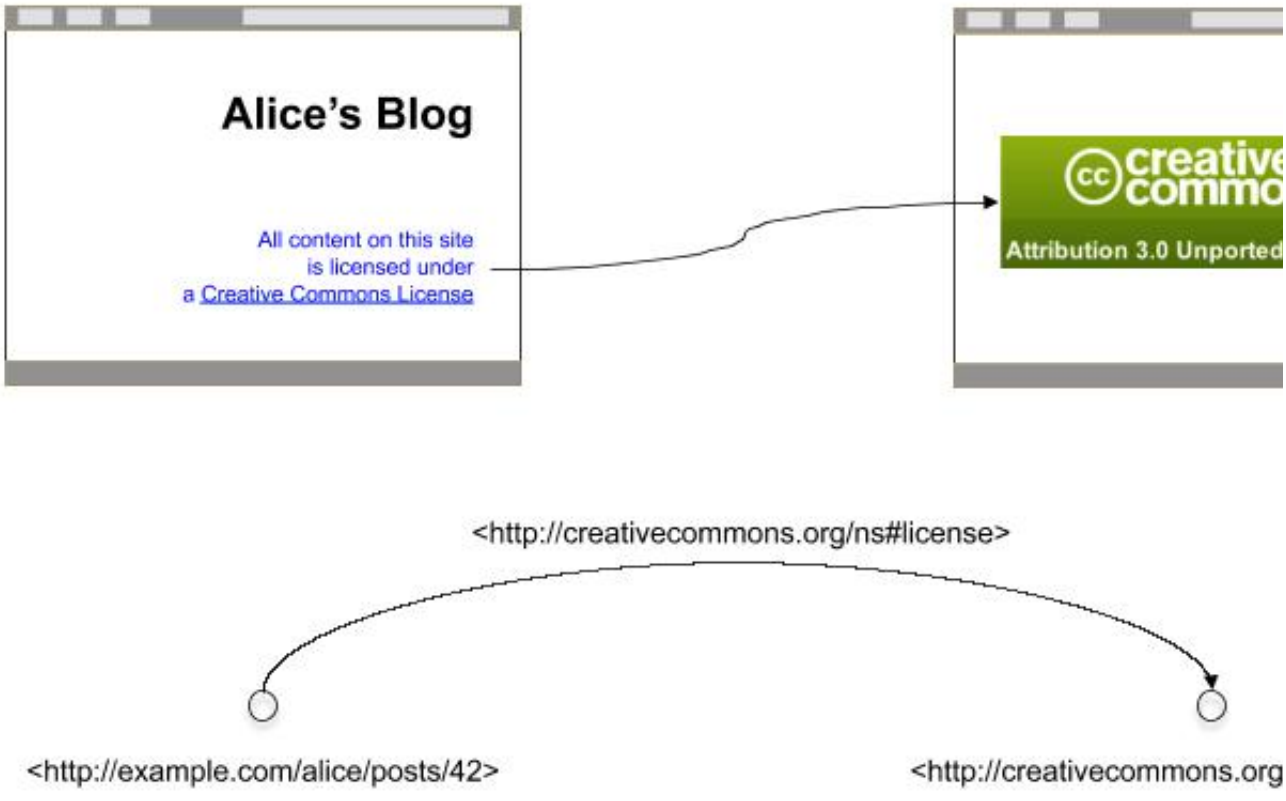


Figure 3: A link with flavor: the link indicates the web page's license. We can represent web pages as nodes, the link as an arrow connecting those nodes, and the link's flavor as the label on that arrow.

Alice is quite pleased that she was able to add only structured-data hints via RDFa, never having to repeat the content of her text or the URL of her clickable links.

2.1.1.3 Setting a Default Vocabulary

In a number of simple use cases, such as our example with Alice's blog, HTML authors will predominantly use a single vocabulary. However, while generating full URLs via a CMS system is not a particular problem, typing these by hand may be error prone and tedious for humans. To alleviate this problem RDFa introduces the `vocab` attribute to let the author declare a single vocabulary for a chunk of HTML. Thus, instead of:

EXAMPLE 5

```
<html>
<head>
  ...
</head>
<body>
  ...
  <h2 property="http://purl.org/dc/terms/title">The Trouble with Bob</h2>
  <p>Date: <span property="http://purl.org/dc/terms/created">2011-09-10</span></p>
  ...
</body>
```

Alice can write:

EXAMPLE 6

```
<html>
<head>
  ...
</head>
<body vocab="http://purl.org/dc/terms/">
  ...
  <h2 property="title">The Trouble with Bob</h2>
  <p>Date: <span property="created">2011-09-10</span></p>
  ...
</body>
```

Note how the property values are single "terms" now; these are simply concatenated to the URL defined via the `vocab` attribute. The attribute can be placed on *any* HTML element (i.e., not only on the `body` element like in the example) and its effect is valid for all the elements below that point.

Default vocabularies and full URIs can be mixed at any time. I.e., Alice could have written:

EXAMPLE 7

```
<html>
<head>
  ...
</head>
<body vocab="http://purl.org/dc/terms/">
  ...
  <h2 property="title">The Trouble with Bob</h2>
  <p>Date: <span property="http://purl.org/dc/terms/created">2011-09-10</span></p>
  ...
</body>
```

Perhaps a more interesting example is the combination of the header with the licensing segment of her web page:

EXAMPLE 8

```
<html>
<head>
  ...
</head>
<body vocab="http://purl.org/dc/terms/">
  ...
  <h2 property="title">The Trouble with Bob</h2>
  <p>Date: <span property="created">2011-09-10</span></p>
  ...
  <p>All content on this site is licensed under
   <a property="http://creativecommons.org/ns#license" href="http://creativecommons
     a Creative Commons License</a>. ©2011 Alice Birpemswick.</p>
</body>
</html>
```

The full URL for the license term is necessary to avoid mixing vocabularies. As an alternative, Alice could have also chosen to use the `vocab` attribute again:

EXAMPLE 9

```
<html>
<head>
  ...
</head>
<body vocab="http://purl.org/dc/terms/">
  ...
  <h2 property="title">The Trouble with Bob</h2>
  <p>Date: <span property="created">2011-09-10</span></p>
  ...
  <p vocab="http://creativecommons.org/ns#">All content on this site is licensed un
    <a property="license" href="http://creativecommons.org/licenses/by/3.0/">
      a Creative Commons License</a>. ©2011 Alice Birpemswick.</p>
</body>
</html>
```

because the vocab in the license paragraph overrides the definition inherited from the body of the document.

> NOTE
>
> The vocab attribute references structured data vocabularies, identified using URLs. RDFa does not limit the form of these URLs or the document formats accessible by de-referencing them; however users SHOULD aim to use widely shared, conventional values for identifying such vocabularies, following conventions of case, spelling etc. established by their publishers.

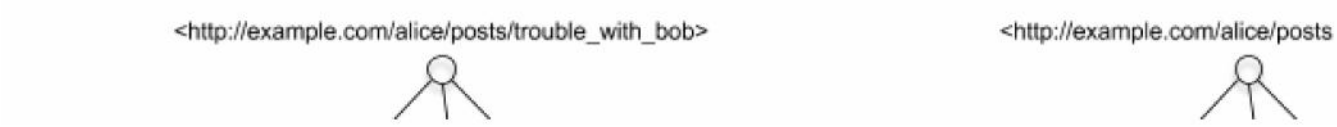
2.1.1.4 Multiple Items per Page

Alice's blog page may contain, of course, multiple entries. Sometimes, Alice's sister Eve guest blogs, too. The front page of the blog lists the 10 most recent entries, each with its own title, author, and introductory paragraph. How, then, should Alice mark up the title of each of these entries individually even though they all appear within the same web page? RDFa provides resource, an attribute for specifying the "context", i.e., the exact URL to which the contained RDFa markup applies:

> EXAMPLE 10

```
<body vocab="http://purl.org/dc/terms/">
   ...
   <div resource="/alice/posts/trouble_with_bob">
      <h2 property="title">The trouble with Bob</h2>
      <p>Date: <span property="created">2011-09-10</span></p>
      <h3 property="creator">Alice</h3>
      ...
   </div>
   ...
   <div resource="/alice/posts/jos_barbecue">
      <h2 property="title">Jo's Barbecue</h2>
      <p>Date: <span property="created">2011-09-14</span></p>
      <h3 property="creator">Eve</h3>
      ...
   </div>
   ...
</body>
```

(Note that we used relative URLs in the example; the value of resource could have been *any* URLs, i.e., relative or absolute.) We can represent this, once again, as a diagram connecting URLs to properties:



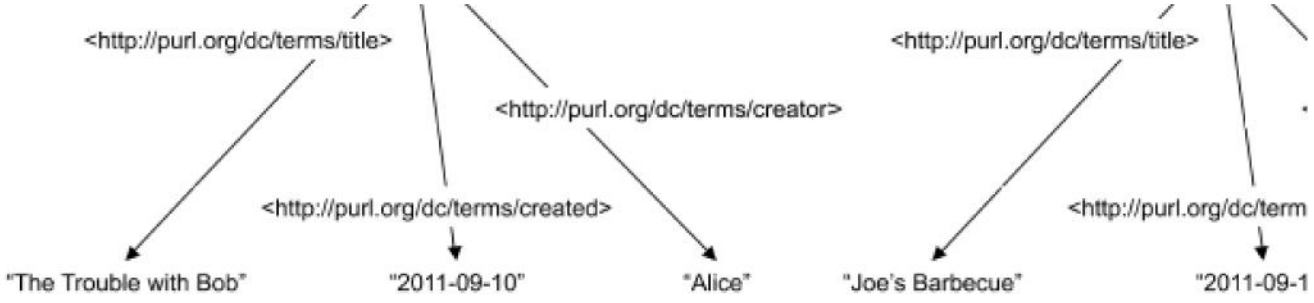


Figure 4: Multiple Items per Page: each blog entry is represented by its own node, with properties attached to each.

Alice can use the same technique to give her friend Bob proper credit when she posts one of his photos:

> EXAMPLE 11
>
> ```
> 
>     <h2 property="title">The trouble with Bob</h2>
>     ...
>     The trouble with Bob is that he takes much better photos than I do:
>     ...
>     
>       <img src="http://example.com/bob/photos/sunset.jpg" />
>       Beautiful Sunset
>       by Bob.
>     
>  
> ```

Notice how the innermost `resource` value, `http://example.com/bob/photos/sunset.jpg`, "overrides" the outer value `/alice/posts/trouble_with_bob` for all markup inside the containing `div`. Once again, here is a diagram that represents the underlying data of this new portion of markup:

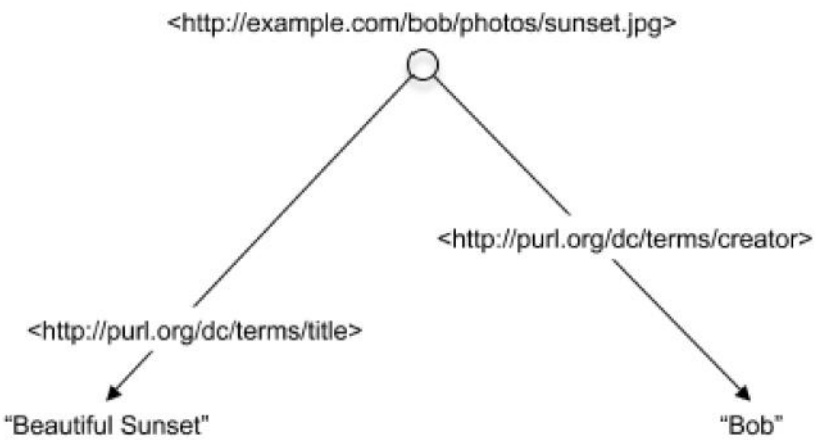


Figure 5: Describing a Photo

2.1.2 Exploring Further: Social networks

2.1.2.1 Contact Information

Alice would also like to make information about herself, such as her email address, phone number, and other details, easily available to her friends' contact management software. This time, instead of describing the properties of a web page, she's going to describe the properties of a person: herself.

Alice already has contact information displayed on her blog.

EXAMPLE 12

```
<div>
  <p>
     Alice Birpemswick,
     Email: <a href="mailto:alice@example.com">alice@example.com</a>,
     Phone: <a href="tel:+1-617-555-7332">+1 617.555.7332</a>
  </p>
</div>
```

The Dublin Core vocabulary does not provide property names for describing contact information, but the Friend-of-a-Friend [FOAF] vocabulary does. Alice therefore decides to use the FOAF vocabulary. As a first step, she declares a FOAF "Person". For this purpose, Alice uses `typeof`, an RDFa attribute that is specifically meant to declare a new data item with a certain type:

EXAMPLE 13

```
<div typeof="http://xmlns.com/foaf/0.1/Person">
  ...
```

Alice realizes that she only intends to use the FOAF vocabulary at this point, so she uses the `vocab` attribute to simplify her markup further (and overriding the effects of any `vocab` attributes that may have been used in, for example, the `body` element at the top).

EXAMPLE 14

```
<div vocab="http://xmlns.com/foaf/0.1/" typeof="Person">
  ...
```

Then, Alice indicates which content on the page represents her full name, email address, and phone number:

EXAMPLE 15

```
<div vocab="http://xmlns.com/foaf/0.1/" typeof="Person"><p>
  <p>
    <span property="name">Alice Birpemswick</span>,
    Email: <a property="mbox" href="mailto:alice@example.com">alice@example.com</a>
    Phone: <a property="phone" href="tel:+1-617-555-7332">+1 617.555.7332</a>
  </p>
</div>
```

Note how Alice did not specify a `resource` like she did when adding blog entry metadata. But, if she is not declaring what she is talking about, how does the RDFa Processor know what she's identifying? In RDFa, in the absence of a `resource` attribute, the `typeof` attribute on the enclosing `div` implicitly sets the subject of the properties marked up within that `div`. That is, the name, email address, and phone number are associated with a new node of type `Person`. This node has no URL to identify it, so it is called a *blank node* as shown on the figure:

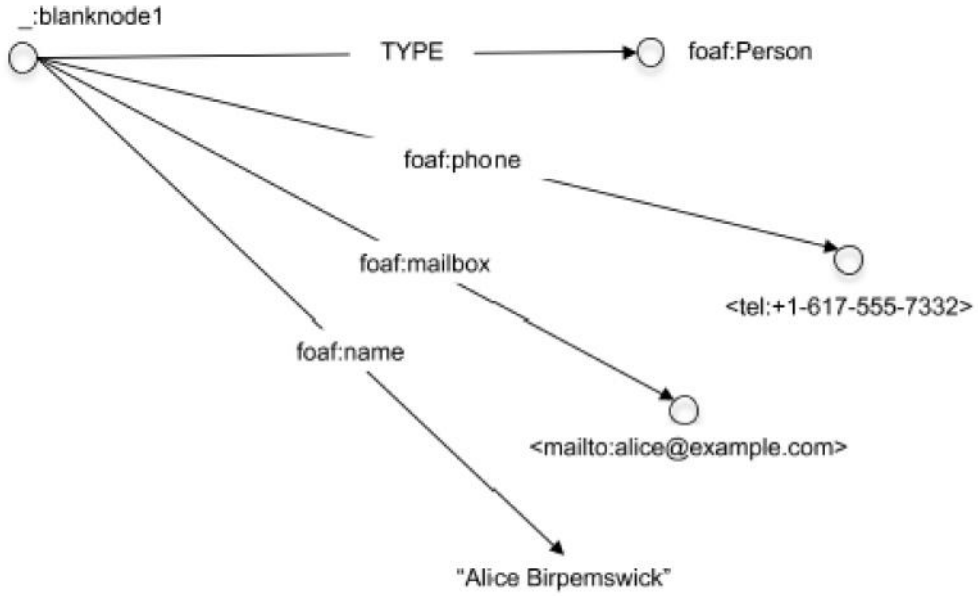


Figure 6: A Blank Node: blank nodes are not identified by URL. Instead, many of them have an RDFa `typeof` attribute that identifies the type of data they represent.
(We've used a short-hands to label the arrows, in order to save space and clarify the diagram. The actual labels are always the full URLs.)

2.1.2.2 Describing Social Networks

Alice continues to mark up her page by adding information about her friends, including at least their names and homepages. She starts with plain HTML:

EXAMPLE 16

```
<div>
   <ul>
      <li>
```

```
        <li>
          <a href="http://example.com/bob/">Bob</a>
        </li>
        <li>
          <a href="http://example.com/eve/">Eve</a>
        </li>
        <li>
          <a href="http://example.com/manu/">Manu</a>
        </li>
    </ul>
</div>
```

First, Alice indicates that the friends she is describing are people, as opposed to animals or imaginary friends, by using again the `Person` type in `typeof` attributes.

EXAMPLE 17

```
<div vocab="http://xmlns.com/foaf/0.1/">
    <ul>
        <li typeof="Person">
          <a href="http://example.com/bob/">Bob</a>
        </li>
        <li typeof="Person">
          <a href="http://example.com/eve/">Eve</a>
        </li>
        <li typeof="Person">
          <a href="http://example.com/manu/">Manu</a>
        </li>
    </ul>
</div>
```

Beyond declaring the type of data we are dealing with, each `typeof` creates a new blank node with its own distinct properties. Thus, Alice can indicate each friend's homepage:

EXAMPLE 18

```
<div vocab="http://xmlns.com/foaf/0.1/">
    <ul>
        <li typeof="Person">
          <a property="homepage" href="http://example.com/bob/">Bob</a>
        </li>
        <li typeof="Person">
          <a property="homepage" href="http://example.com/eve/">Eve</a>
        </li>
        <li typeof="Person">
          <a property="homepage" href="http://example.com/manu/">Manu</a>
        </li>
    </ul>
</div>
```

Alice would also like to improve the markup by expressing each person's name using RDFa, too. That can be done by adding a separate `span` element and the relevant `property`:

EXAMPLE 19

```
<div vocab="http://xmlns.com/foaf/0.1/">
  <ul>
      <li typeof="Person">
        <a property="homepage" href="http://example.com/bob/"><span property="name"
      </li>
      <li typeof="Person">
        <a property="homepage" href="http://example.com/eve/"><span property="name"
      </li>
      <li typeof="Person">
        <a property="homepage" href="http://example.com/manu/"><span property="name
      </li>
   </ul>
</div>
```

Alice is happy that, with so little additional markup, she's able to fully express both a pleasant human-readable page and a machine-readable dataset.

Alice is a member of 5 different social networking sites. She is tired of repeatedly entering information about her friends in each new social networking site, so she decides to list her friends in one place-on her website, combining it with her own FOAF data. With RDFa, she can indicate her friendships on her own web page and let social networking sites read it automatically. So far, Alice has listed three individuals but has not specified her relationship with them; they might be her friends, or they might be her favorite 17th century poets. To indicate that she knows them, she uses the FOAF property `foaf:knows`:

EXAMPLE 20

```
<div vocab="http://xmlns.com/foaf/0.1/" typeof="Person">
   <p>
    <span property="name">Alice Birpemswick</span>,
    Email: <a property="mbox" href="mailto:alice@example.com">alice@example.com</a>
    Phone: <a property="phone" href="tel:+1-617-555-7332">+1 617.555.7332</a>
  </p>
   <ul>
      <li property="knows" typeof="Person">
        <a property="homepage" href="http://example.com/bob/"><span property="name"
      </li>
      <li property="knows" typeof="Person">
        <a property="homepage" href="http://example.com/eve/"><span property="name"
      </li>
      <li property="knows" typeof="Person">
        <a property="homepage" href="http://example.com/manu/"><span property="name
      </li>
   </ul>
</div>
```

With this, Alice could describe here social network:

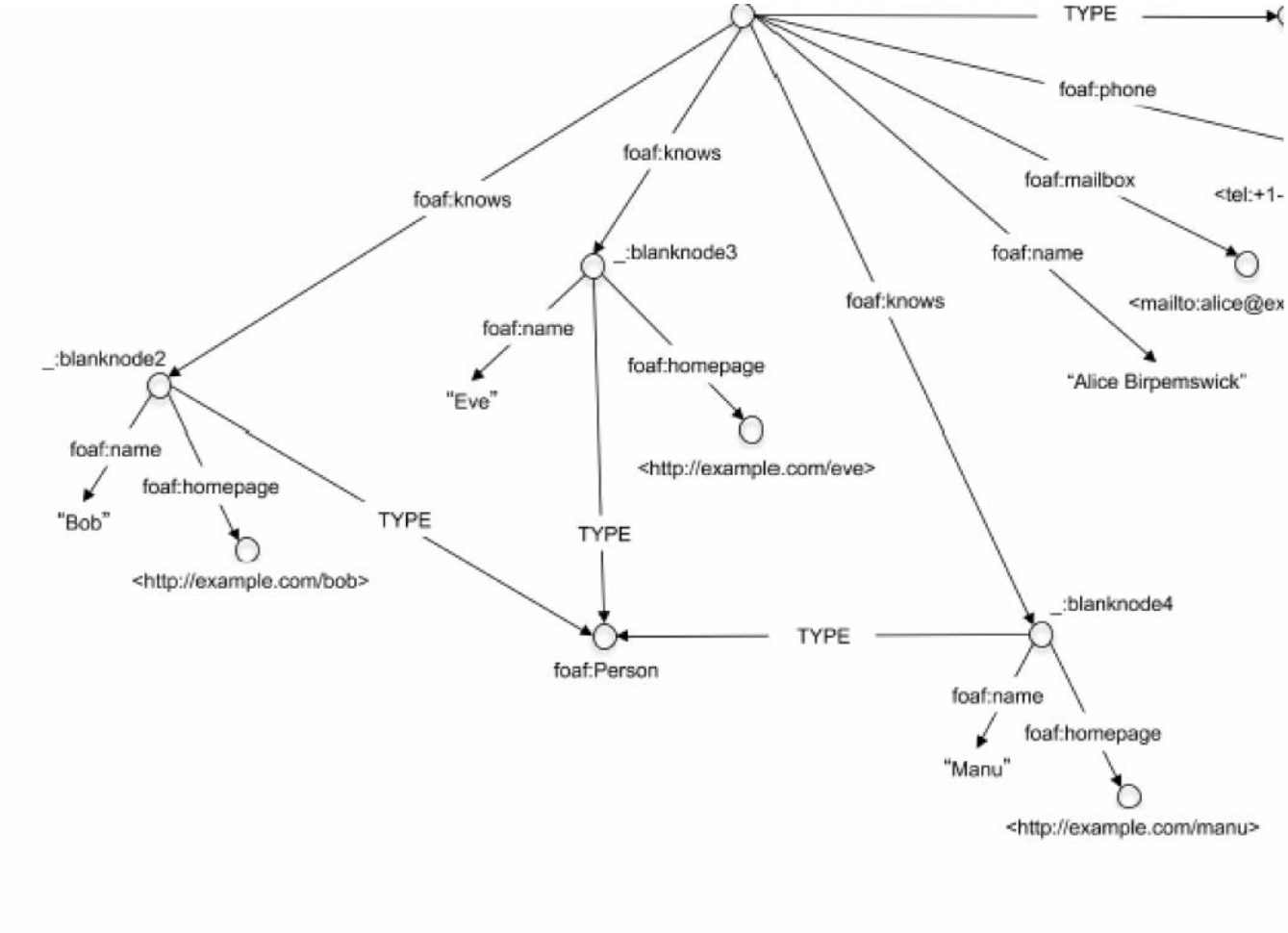


Figure 7: Alice's social network. Note that, with RDFa, Alice could express a fairly complex set of information that others can use.

2.1.3 Repeated Patterns

We have seen, in a previous section, how Alice can use RDFa to include Creative Commons statements on her blog. However, the solution in that section assigned these statements *to the whole page*, and not to individual blog items. This may be an issue if the page includes multiple items. Indeed, Alice may be forced to repeat the relevant statements like this:

EXAMPLE 21

```
<body vocab="http://purl.org/dc/terms/">
   ...
   <div resource="/alice/posts/trouble_with_bob">
      <h2 property="title">The trouble with Bob</h2>
      <p>Date: <span property="created">2011-09-10</span></p>
      <h3 property="creator">Alice</h3>
      ...
      <p vocab="http://creativecommons.org/ns#">All content on this blog item is li
        <a property="license" href="http://creativecommons.org/licenses/by/3.0/">
          a Creative Commons License</a>. <span property="attributionName">©2011 Al
   </div>
   ...
   <div resource="/alice/posts/jims_concert">
```

```
      <h2 property="title">I was at Jim's concert the other day</h2>
      <p>Date: <span property="created">2011-10-22</span></p>
      <h3 property="creator">Alice</h3>
      ...
      <p vocab="http://creativecommons.org/ns#">All content on this blog item is li
        <a property="license" href="http://creativecommons.org/licenses/by/3.0/">
          a Creative Commons License</a>. <span property="attributionName">©2011 Al
   </div>
   ...
</body>
```

which may be tedious and error prone.

HTML+RDFa introduces the notion of "Property copying" to alleviate this situation. Using this feature Alice can "collect" a number of statements as a pattern, and refer to that pattern from other parts of the page. This is done using the magic property `rdfa:copy` and the magic type `rdfa:Pattern` as follows:

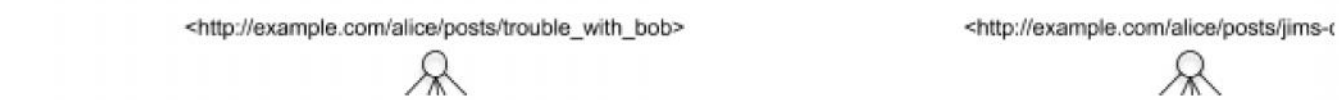
EXAMPLE 22

```
<body vocab="http://purl.org/dc/terms/">
   ...
   <div resource="/alice/posts/trouble_with_bob">
      <h2 property="title">The trouble with Bob</h2>
      <p>Date: <span property="created">2011-09-10</span></p>
      <h3 property="creator">Alice</h3>
      ...
      <link property="rdfa:copy" href="#ccpattern"/>
    </div>
   ...
   <div resource="/alice/posts/jims_concert">
      <h2 property="title">I was at Jim's concert the other day</h2>
      <p>Date: <span property="created">2011-10-22</span></p>
      <h3 property="creator">Alice</h3>
      ...
      <link property="rdfa:copy" href="#ccpattern"/>
   </div>
   ...

   <div resource="#ccpattern" typeof="rdfa:Pattern">
      <p vocab="http://creativecommons.org/ns#">All content on this blog item is li
        <a property="license" href="http://creativecommons.org/licenses/by/3.0/">
          a Creative Commons License</a>. <span property="attributionName">©2011 Al
   </div>

</body>
```

(Alice may choose to use CSS to make the CC statements invisible on the screen if she wants.) The effect of this structure is to, conceptually, "copy" all the RDFa statements appearing in the pattern to replace the `link` element, yielding the following structure:



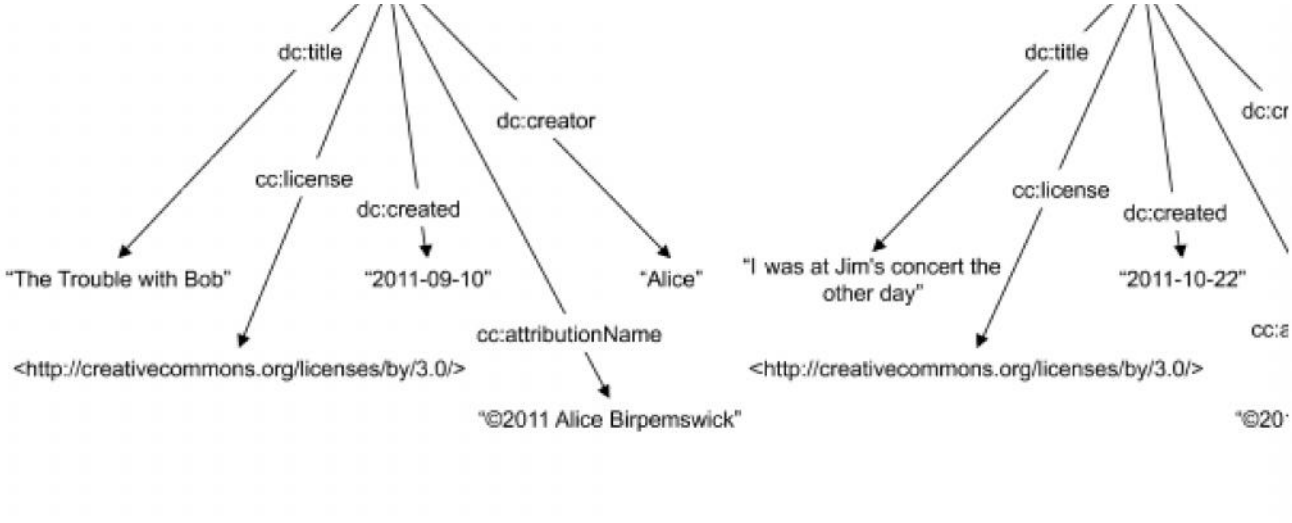


Figure 8: Creative Commons statements added to each blog item separately.

2.1.4 Internal References

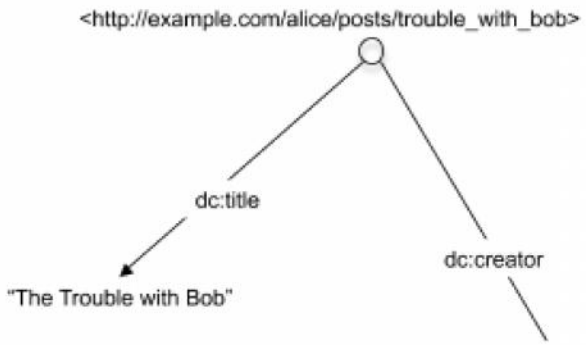
Alice may want to add her personal data to her individual blog items, too. She decides to combine her FOAF data with the blog items, i.e.:

EXAMPLE 23

```
<div vocab="http://purl.org/dc/terms/">

   <div resource="/alice/posts/trouble_with_bob">
      <h2 property="title">The trouble with Bob</h2>
      ...
      <h3 vocab="http://xmlns.com/foaf/0.1/" property="http://purl.org/dc/terms/cre
        <span property="name">Alice Birpemswick</span>,
        Email: <a property="mbox" href="mailto:alice@example.com">alice@example.com
        Phone: <a property="phone" href="tel:+1-617-555-7332">+1 617.555.7332</a>
      </h3>
      ...
   </div>
   ...
</div>
```

The structured data she generates looks like this:



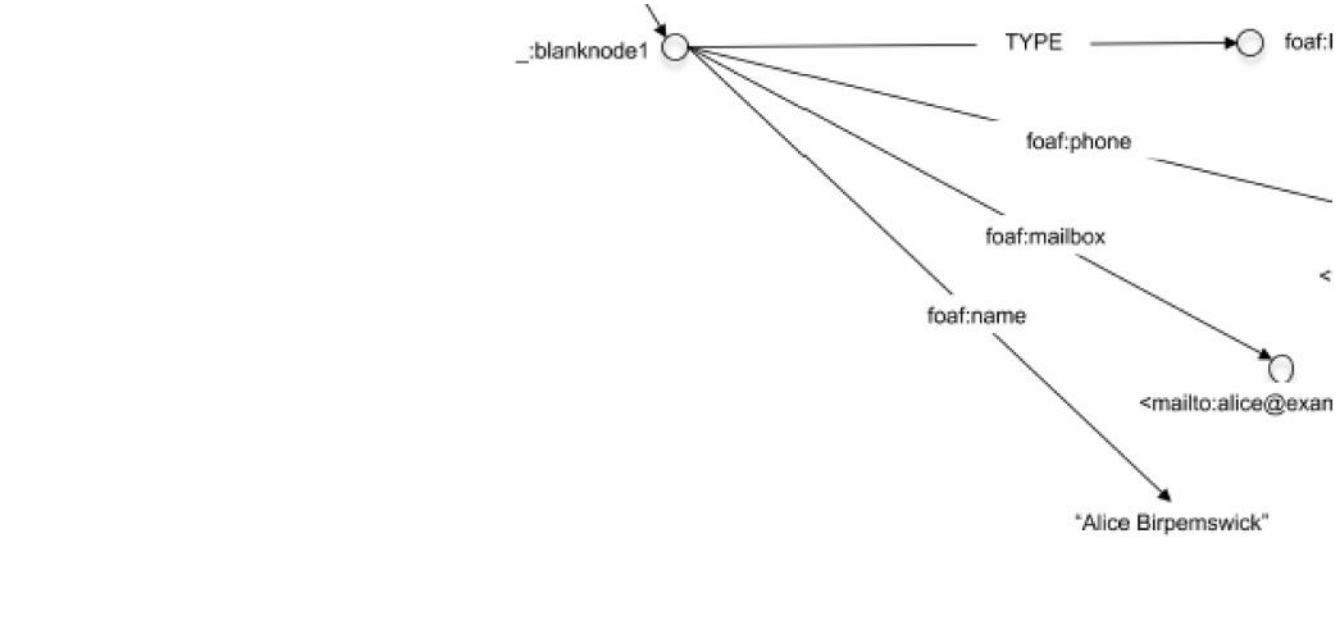


Figure 9: Alice's blog item with data about herself.

Unfortunately, this solution is not optimal in two respects. First of all, notice that Alice had to use the full URI for the `creator` property: this is because the `vocab` attribute is used to set the FOAF terms, i.e., the simple `creator` value would have been misinterpreted. We will come back to the issue of using several vocabularies in another section below.

The other issue is that Alice would like to design her Web page so that her personal data would not appear on the page in each individual blog item but, rather, in one place like a footnote or a sidebar. I.e., what she would like to see is something like:

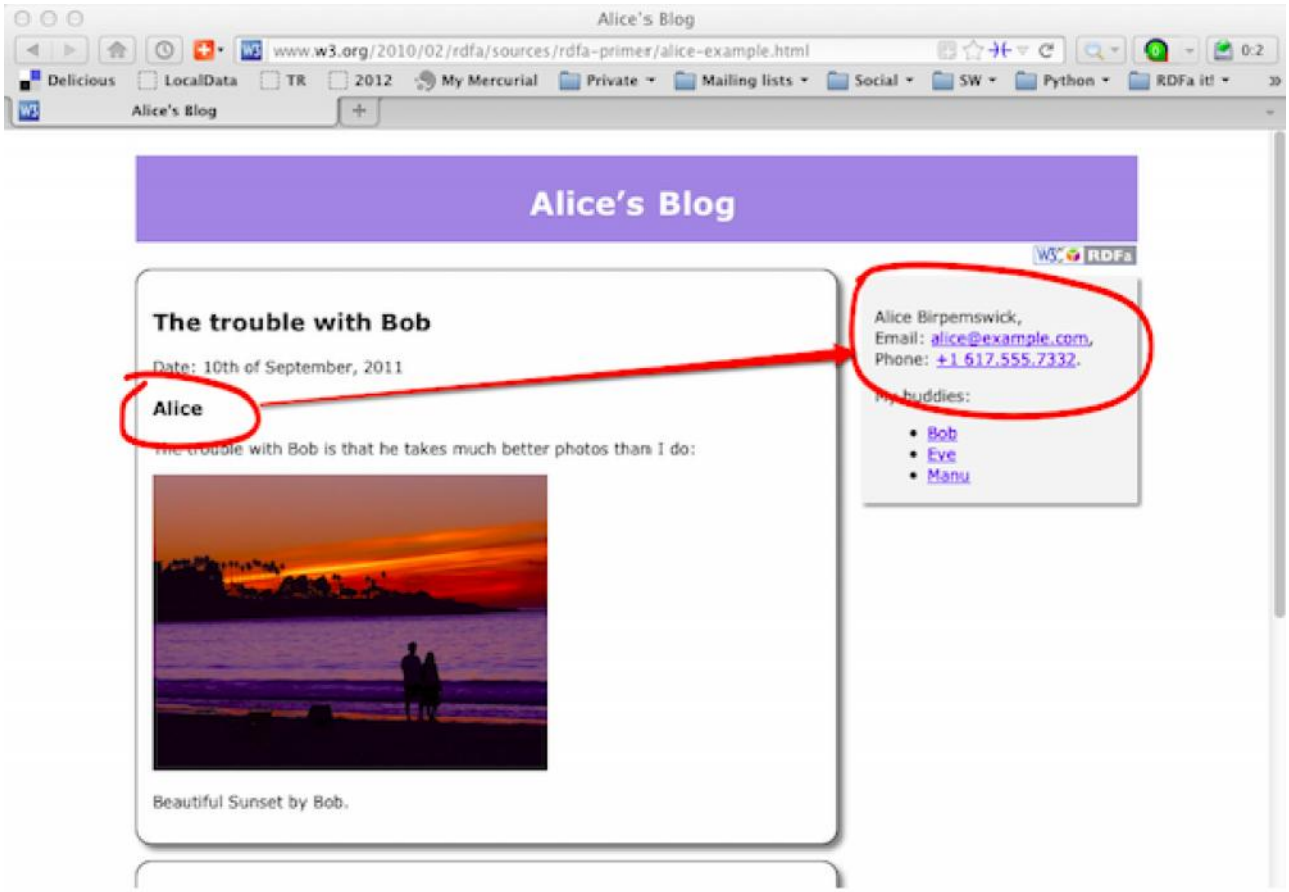


Figure 10: Structure of Alice's Site: individual blog items on the left, personal data, linked from the blog using RDFa terms, in a sidebar.

If the FOAF data was included into each blog item, Alice would have to create a complex set of CSS rules to achieve the visual effect she wants.

To solve this, Alice decides to make use of the structure she already used for her FOAF data but, this time, assigning it a separate URI using the `resource` attribute:

EXAMPLE 24

```
<div vocab="http://xmlns.com/foaf/0.1/" resource="#me" typeof="Person">
  <p>
   <span property="name">Alice Birpemswick</span>,
     Email: <a property="mbox" href="mailto:alice@example.com">alice@example.com</a
     Phone: <a property="phone" href="tel:+1-617-555-7332">+1 617.555.7332</a>
  </p>
  ...
</div>
```

It is actually considered as a good practice to use real URIs whenever possible, i.e., Alice's new alternative should be preferred in general. Indeed, if a real URI is used, then it becomes possible to unambiguously refer to that particular piece of information, whereas that becomes more complicated with blank nodes.

NOTE

The `resource="#me"` markup is a FOAF convention: the URL that represents the *person* Alice is `http://example.com/alice#me`. It should not be confused with Alice's homepage, `http://example.com/alice`. Of course, Alice could have used a different URI if, for example, her blog and her personal homepage were kept separate; e.g., she could have used `resource="http://alice.example.com/alice/home#myself"` instead of `resource="#me"`.

Using the explicit URI for her FOAF data Alice can add a direct reference to the blog item using again the `resource` attribute:

EXAMPLE 25

```
<div vocab="http://purl.org/dc/terms/">
   <div resource="/alice/posts/trouble_with_bob">
      <h2 property="title">The trouble with Bob</h2>
      <h3 property="creator" resource="#me">Alice</h3>
      ...
   </div>
</div>
   ...
<div class="sidebar" vocab="http://xmlns.com/foaf/0.1/" resource="#me" typeof="Pers
```

```
    <p>
     <span property="name">Alice Birpemswick</span>,
       Email: <a property="mbox" href="mailto:alice@example.com">alice@example.com</a
       Phone: <a property="phone" href="tel:+1-617-555-7332">+1 617.555.7332</a>
    </p>
    ...
  </div>
```

The `resource` attribute appears, in this case, together with `property` *on the same element*: in this situation `resource` indicates the "target" of the relation. Usage of this attribute allows Alice to "distribute" the various parts of her structured data on her page. What she gets is a slightly modified version of the previous structure, where the only difference is the usage of an explicit URI instead of a blank node:

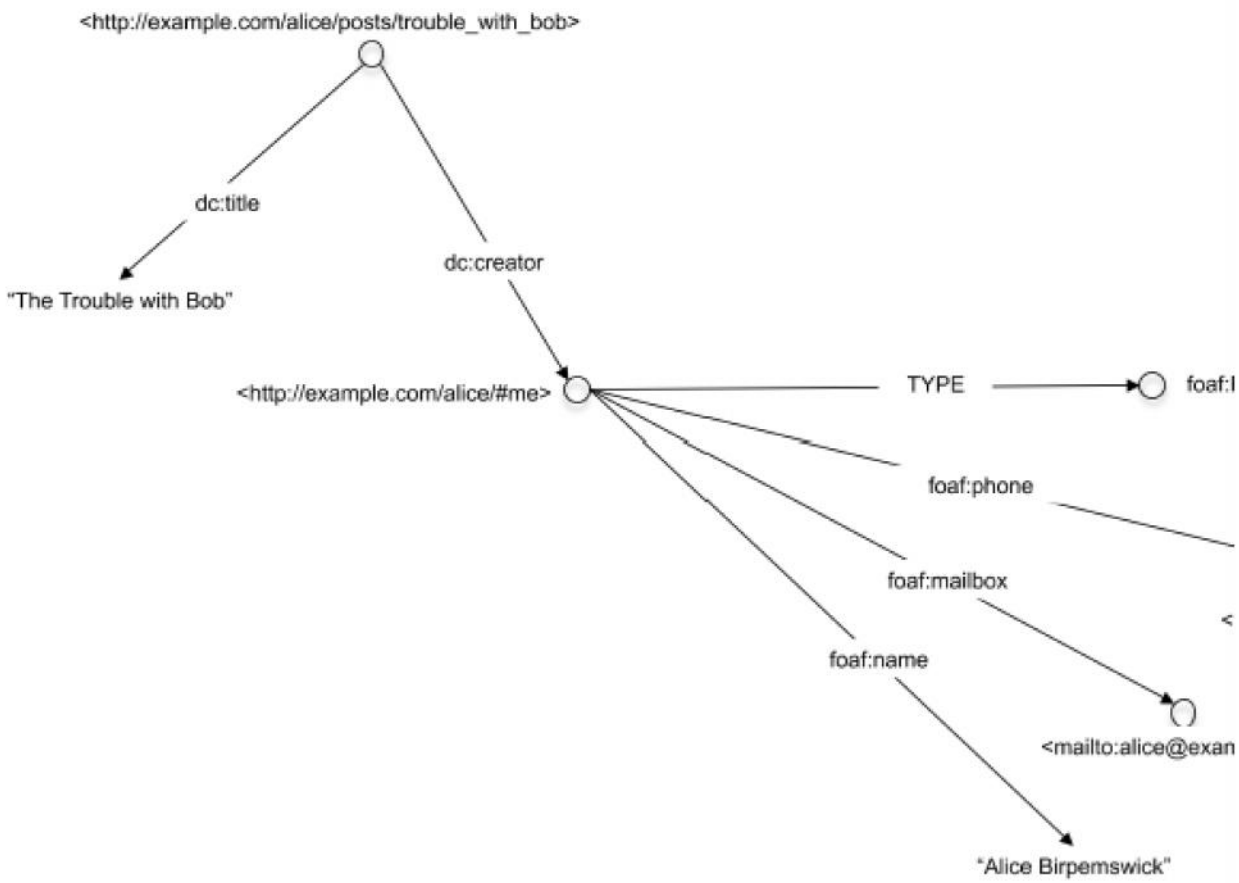


Figure 11: Alice's blog item with data about herself, using an explicit URI for her FOAF data.

Using this approach, it becomes very easy to also add references to the *same* data from different blogs:

EXAMPLE 26

```
<div vocab="http://purl.org/dc/terms/">
   <div resource="/alice/posts/trouble with bob">
```

```
      <h2 property="title">The trouble with Bob</h2>
      <h3 property="creator" resource="#me">Alice</h3>
      ...
   </div>
</div>
   ...
<div vocab="http://purl.org/dc/terms/">
   <div resource="/alice/posts/my_photos">
      <h2 property="title">I will post my photos nevertheless…</h2>
      <h3 property="creator" resource="#me">Alice</h3>
      ...
   </div>
</div>
   ...
<div class="sidebar" vocab="http://xmlns.com/foaf/0.1/" resource="#me" typeof="Pers
  <p>
   <span property="name">Alice Birpemswick</span>,
     Email: <a property="mbox" href="mailto:alice@example.com">alice@example.com</a
     Phone: <a property="phone" href="tel:+1-617-555-7332">+1 617.555.7332</a>
  </p>
  ...
</div>
```

Leading to the following structure:

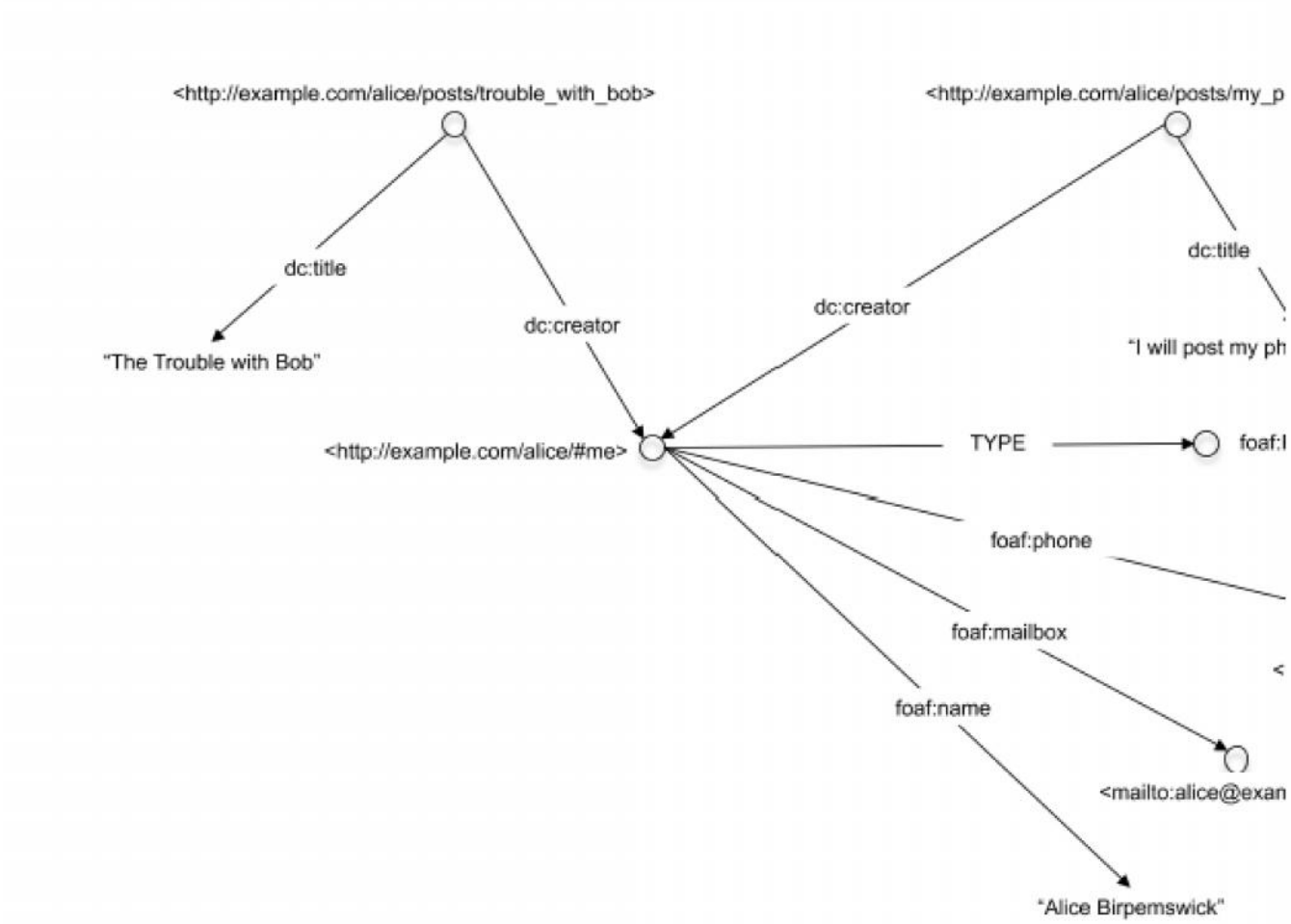


Figure 12: Several of Alice's blog items with data about herself, using an explicit URI for her FOAF data.

NOTE

Combined with `property`, the `resource` attribute plays exactly the same role as `href`, already used for "links with flavor", except that it does not provide a clickable link to the browser like `href` does. Also, the `resource` attribute can be used on *any* HTML element, as opposed to `href` whose usage is restricted, in HTML, to the `a` and `link` elements.

NOTE

There is a similarity between this issue and its solution and the issue and the approach taken in the section on property copying. There is, however, a subtle but important difference between the two. The solution using the `resource` attribute introduces a new node in the graph, as shown on Figure 12, whereas copying the properties does not. Which of the two approaches should be adopted is often based on the vocabulary that is used.

2.1.5 Using Multiple Vocabularies

The previous examples show that, for more complex cases, multiple vocabularies have to be used to express the various aspects of structured data. We have seen Alice using the Dublin Core, as well as the FOAF and the Creative Commons vocabularies, but there may be more. For example. Alice may want to add vocabulary elements defined by search engines on their schema.org site [schema].

Alice can use either full URLs for all the terms, or can use the `vocab` attribute to abbreviate the terms for the predominant vocabulary. But, in some cases, the vocabularies cannot be separated easily, which means that the usage of `vocab` may become awkward. Here is, for example, the kind of HTML she might end up with:

EXAMPLE 27

```
<html>
 <head>
    ...
 </head>
 <body vocab="http://schema.org/">
   <div resource="/alice/posts/trouble_with_bob" typeof="BlogPosting">
      <h2 property="http://purl.org/dc/terms/title">The trouble with Bob</h2>
      ...
      <h3 property="http://purl.org/dc/terms/creator" resource="#me">Alice</h3>
      <div property="articleBody">
        <p>The trouble with Bob is that he takes much better photos than I do:</p>
      </div>
      ...
```

```
      ...
    </div>
    ...
   </body>
  </html>
```

Note that the schema.org and the Dublin Core terms are intertwined for a specific blog, and it becomes an arbitrary choice whether to use the `vocab` attribute for `http://purl.org/dc/terms/` or for `http://schema.org/`. We have seen the same problem in a previous section when FOAF and Dublin Core terms were mixed.

To alleviate this problem, RDFa offers the possibility of using *prefixed* terms: a special `prefix` attribute can assign prefixes to represent URLs and, using those prefixes, the vocabulary elements themselves can be abbreviated. The `prefix:reference` syntax is used: the URL associated with `prefix` is simply concatenated to `reference` to create a full URL. (Note that we have already used this convention to simplify our figures.) Here is how the HTML of the previous example looks like when prefixes are used:

EXAMPLE 28

```
<html>
 <head>
   ...
 </head>
 <body prefix="dc: http://purl.org/dc/terms/ schema: http://schema.org/">
   <div resource="/alice/posts/trouble_with_bob" typeof="schema:BlogPosting">
      <h2 property="dc:title">The trouble with Bob</h2>
      ...
      <h3 property="dc:creator" resource="#me">Alice</h3>
      <div property="schema:articleBody">
        <p>The trouble with Bob is that he takes much better photos than I do:</p>
      </div>
     ...
   </div>
 </body>
</html>
```

The usage of prefixes can greatly reduce possible errors by concentrating the vocabulary choices to one place in the file. Just like `vocab`, the `prefix` attribute can appear anywhere in the HTML file, only affecting the elements below. `prefix` and `vocab` can also be mixed, for example:

EXAMPLE 29

```
<html>
 <head>
   ...
 </head>
 <body vocab="http://purl.org/dc/terms/" prefix="schema: http://schema.org/">
   <div resource="/alice/posts/trouble_with_bob" typeof="schema:BlogPosting">
      <h2 property="title">The trouble with Bob</h2>
      ...
      <h3 property="creator" resource="#me">Alice</h3>
      <div property="schema:articleBody">
```

```
        <div property="schema:articleBody">
          <p>The trouble with Bob is that he takes much better photos than I do:</p>
        </div>
        ...
      </div>
    </body>
  </html>
```

NOTE

An important issue may arise if the `html` element contains a large number of prefix declarations. The character encoding (i.e., UTF-8, UTF-16, ASCII, etc.) used for an HTML5 file is declared using a `meta` element in the header. In HTML5 this meta declaration must fall within the first 512 bytes of the page, or the HTML5 processor (browser, parser, etc.) will try to detect the encoding using some heuristics. A very "long" `html` tag may therefore lead to problems. One way of avoiding the issue is to place most of the prefix declarations on the `body` element.

2.1.5.1 Repeating properties

The previous example, whereby the Dublin Core and the schema.org vocabularies are used within the same blog post, raises another issue. It so happens that not only Dublin Core, but also schema.org has a property called `creator`. Because RDFa uses URIs to denote properties that, by itself, is not a problem. However, if Alice wants to use *both* these properties in the same blog post (e.g., because she wants search engines to manage her blog post but, at the same times, she wants Dublin Core aware applications, like catalogs, to handle her blog post, too) this is what she may have to do:

EXAMPLE 30

```
<html>
  <head>
    ...
  </head>
  <body prefix="dc: http://purl.org/dc/terms/ schema: http://schema.org/">
    <div resource="/alice/posts/trouble_with_bob" typeof="schema:BlogPosting">
      <h2 property="dc:title">The trouble with Bob</h2>
      ...
      <h3 property="dc:creator" resource="#me"><span property="schema:creator" reso
      <div property="schema:articleBody">
        <p>The trouble with Bob is that he takes much better photos than I do:</p>
      </div>
      ...
    </div>
  </body>
</html>
```

Which is a bit awkward. Fortunately, RDFa allows the value of a `property` attribute to be a list of values, i.e., she can also write:

EXAMPLE 31

EXAMPLE 31

```
<html>
 <head>
   ...
 </head>
 <body prefix="dc: http://purl.org/dc/terms/ schema: http://schema.org/">
   <div resource="/alice/posts/trouble_with_bob" typeof="schema:BlogPosting">
      <h2 property="dc:title">The trouble with Bob</h2>
      ...
      <h3 property="dc:creator schema:creator" resource="#me">Alice</h3>
      <div property="schema:articleBody">
        <p>The trouble with Bob is that he takes much better photos than I do:</p>
      </div>
     ...
   </div>
 </body>
</html>
```

yielding the structure:

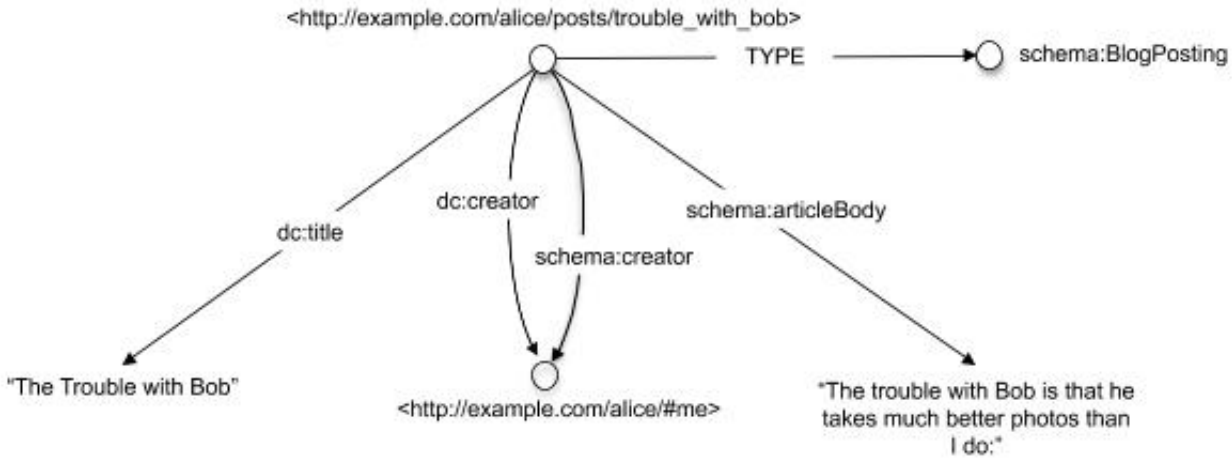


Figure 13: Alice's blog item using two different vocabularies, including two properties with the same context and target.

Similarly to `property`, `typeof` also accepts a list of values. For example, schema.org also has a notion of a Person, similar to FOAF; Alice may choose to use both:

EXAMPLE 32

```
<div class="sidebar" prefix="http://xmlns.com/foaf/0.1/ schema: http://schema.org/"
     resource="#me" typeof="foaf:Person schema:Person">
  <p>
   <span property="foaf:name">Alice Birpemswick</span>,
     Email: <a property="foaf:mbox" href="mailto:alice@example.com">alice@example.c
     Phone: <a property="foaf:phone" href="tel:+1-617-555-7332">+1 617.555.7332</a>
  </p>
  ...
</div>
```

2.1.5.2 Default Prefixes (Initial Context)

A number of vocabularies are very widely used by the Web community with well-known prefixes—the Dublin Core vocabulary is a good example. These common vocabularies tend to be defined over and over again, and sometimes Web page authors forget to declare them altogether.

To alleviate this issue, RDFa introduces the concept of an *initial context* that defines a set of default prefixes. These prefixes, whose list is maintained and regularly updated by the W3C, provide a number of pre-defined prefixes that are known to the RDFa processor. Prefix declarations in a document always override declarations made through the defaults, but if a web page author forgets to declare a common vocabulary such as Dublin Core or FOAF, the RDFa Processor will fall back to those. The list of default prefixes are available on the Web for everyone to read.

For example, the following example does *not* declare the `dc:` prefix using a `prefix` attribute:

EXAMPLE 33

```
<html>
 <head>
   ...
 </head>
 <body>
   <div>
      <h2 property="dc:title">The trouble with Bob</h2>
      ...
      <h3 property="dc:creator" resource="#me">Alice</h3>
       ...
   </div>
 </body>
</html>
```

However, an RDFa processor still recognizes the `dc:title` and `dc:creator` short-hands and expands the values to the corresponding URLs. The RDFa processor is able to do this because the `dc` prefix is part of the default prefixes in the initial context.

NOTE

Default prefixes are used as a mechanism to correct RDFa documents where authors accidentally forgot to declare common prefixes. While authors may rely on these to be available for RDFa documents, the prefixes may change over the course of 5-10 years, although the policy of W3C is that once a prefix is defined as part of a default profile, that particular prefix will *not* be changed or removed. Nevertheless, the best way to ensure that the prefixes that document authors use always map to the intent of the author is to use the `prefix` attribute to declare these prefixes.

these prefixes.

Since default prefixes are meant to be a last-resort mechanism to help novice document authors, the markup above is not recommended. The rest of this document will utilize authoring best practices by declaring all prefixes in order to make the document author's intentions explicit.

2.2 Going Deeper: RDFa Core

As we have seen in the previous sections, RDFa Lite is fairly powerful. Alice could indeed express complex sets of structured information. However, there are cases when the set of attributes presented so far does not cover all the needs, or make the resulting HTML structure a bit awkward and possibly error-prone. In those cases additional RDFa possibilities, provided through additional RDFa attributes, may come to the rescue; some of these will be presented in this section.

NOTE

RDFa Lite does not define a separate class of RDFa processors. In other words conforming RDFa processors are supposed to handle all RDFa features, not only those listed used by RDFa Lite.

2.2.1 Using the `content` attribute

When creating her blog, Alice decided to use this simple structure to add Dublin Core information to her blog post (see also Figure 2):

EXAMPLE 34

```
<html>
 <head>
   ...
 </head>
 <body>
   ...
   <h2 property="http://purl.org/dc/terms/title">The Trouble with Bob</h2>
   <p>Date: <span property="http://purl.org/dc/terms/created">2011-09-10</span></p>
   ...
 </body>
</html>
```

However, to do that, Alice had to accept a small compromise. Indeed, although the string "2011-09-10" unambiguously identifies a date for a machine, it does not looks very natural for a human reader. Surely a native English reader would prefer something like "10th of September, 2011". On the other hand, although it is of course possible for a machine to parse and interpret that string as a date, too, it is clearly more complicated to do so. The problem is that, as a default, RDFa uses the textual content of the element for

the property value. While this works well in most of the cases, sometimes, like in this example, this has awkward consequences.

To alleviate this problem RDFa makes it possible to re-use the `content` attribute of HTML. The blog entry could be written as follows:

EXAMPLE 35

```
<html>
 <head>
   ...
 </head>
 <body>
   ...
   <h2 property="http://purl.org/dc/terms/title">The Trouble with Bob</h2>
   <p>Date: <span property="http://purl.org/dc/terms/created" content="2011-09-10">
   ...
 </body>
</html>
```

The resulting structure is exactly the same as before (i.e., Figure 2). The difference is the presence of the `content` attribute: it instructs the RDFa processor to overrule the default behavior of using the textual content, and to use the value of the `content` attribute instead. Using this attribute Alice could provide a more readable date, while maintaining an unambiguous content for machines using the structured data.

The `content` attribute has another important usage. The "traditional" approach to add simple metadata to a Web page has been to use the document header through the `link` and the `meta` elements. While there is no problem using `link` in RDFa Lite (which uses the `href` attribute, i.e., can be used to define "flavored" links), the fact that, in a conforming HTML file, the `meta` element may have no text content means that the *only* way of using the header for such statements is to use the `content` attribute. For example, using the `meta` element is the approach suggested by Facebook for the Open Graph Protocol [OGP] vocabulary; i.e., if Alice wants to make use of the "Like" button in her posts, this is what she would add to her header:

EXAMPLE 36

```
<html>
 <head prefix="og: http://ogp.me/ns#" >
   ...
   <meta property="og:title" content="The Trouble with Bob" />
   <meta property="og:type"  content="text" />
   <meta property="og:image" content="http://example.com/alice/bob-ugly.jpg" />
   ...
 </head>
 <body>
  ...
 </body>
</html>
```

NOTE

In this example the prefix for the Open Graph Protocol vocabulary is defined via the `prefix` attribute. Alas, many authors forget to do so. Fortunately, the `og` prefix is part of the initial context for RDFa, i.e., the resulting information will be valid even without the prefix declaration…

2.2.2 Datatypes

Alice has already put license information on her page:

EXAMPLE 37

```
<p>All content on this site is licensed under
   <a property="http://creativecommons.org/ns#license" href="http://creativecommons
     a Creative Commons License</a>. ©2011 Alice Birpemswick.</p>
```

but she would like to complete this by recording the date of her copyright statement as a structured data, too. She can use the `date` term of Dublin Core:

EXAMPLE 38

```
<p>All content on this site is licensed under
   <a property="http://creativecommons.org/ns#license" href="http://creativecommons
     a Creative Commons License</a>. ©<span property="dc:date">2011</span> Alice Bi
```

However, the value used for the date may be ambiguous for machines. Of course, if a program "knows" that that `http://purl.org/dc/terms/date` refers to a date, then of course it can find out that the string "2011" stands for a year. But there may be processors that, for example, provide a visual presentation of all the structured data on a specific page, and would like to use a different "widget" to represent a year and again another one to represent, say, an integer number. How would such a processor know which one to choose?

Alice may decide to be helpful by adding an additional information to that item in the form of a *datatype*. This additional information can be conveyed to the RDFa processor using the `datatype` RDFa attribute as follows:

EXAMPLE 39

```
<p>All content on this site is licensed under
   <a property="http://creativecommons.org/ns#license" href="http://creativecommons
     a Creative Commons License</a>. ©<span property="dc:date" datatype="xsd:gYear"
```

where `xsd:gYear` stands for `http://www.w3.org/2001/XMLSchema#gYear`, and is one of the standard datatypes defined by W3C's Datatype specification [xmlschema11-2] which

standard datatypes defined by W3C's Datatype specification [XMLSchema11-2] which contains such types as booleans, integers, dates, or doubles. (`xsd` is one of the default prefixes for RDFa.)

2.2.3 Alternative for setting the context: `about`

Alice has used the following patterns to define structured data for the individual blogs:

> EXAMPLE 40
>
> ```
> 
>    <h2 property="title">The trouble with Bob</h2>
>    <h3 property="creator" resource="#me">Alice</h3>
>    ...
> 
> ```

The role of the `resource` attribute in the `div` element is to set the "context", i.e., the subject for all the subsequent statements. Also, when combined with the `property` attribute, `resource` can be used to set the "target", i.e., the object for the statement (much as `href`).

This pattern is perfectly fine, but it may become too verbose in some cases. Indeed, let us suppose that Alice would like to set up a separate index page for all her blogs, and the only information she would like to put there, as structured data, is references to the titles. Following the same pattern, she would have to do something like:

> EXAMPLE 41
>
> ```
> <ul>
>   <li resource="/alice/posts/trouble_with_bob">The trouble w
>   <li resource="/alice/posts/jos_barbecue">Jo's Barbecue</sp
>   ...
> </ul>
> ```

This of course works, but it is a bit convoluted. Merging the information into one element, i.e.:

> EXAMPLE 42
>
> ```
> <ul resource="/alice/posts/trouble_with_bob">
>   <li resource="/alice/posts/trouble_with_bob" property="title">The trouble with Bol
>   ...
> </ul>
> ```