RDF integration in HTML 5 web pages

Gijs Davis

g.davis@student.utwente.nl

ABSTRACT
The adoption of the Semantic Web would benefit greatly if small chunks of Semantic Web data could be integrated in normal web pages. As of yet there is no standardized way of doing this. We will construct a standard for doing this, based on already existing implementations.

Keywords
OWL, RDF, Semantic Web, N3, N-Triples, XML, RDF/XML
Microformats, eRDF, HTML 5, XHTML 5, W3C, WHATWG

1. INTRODUCTION
It has been more than ten years since Tim Berners-Lee first published his vision of the Semantic Web [1]. The idea behind the semantic web is to create web pages that contain the same data as available now on the ‘normal’ web, but written in a way that they are easily understandable by computers instead of human beings. While a lot has happened in the development of the Semantic Web, the Semantic Web has yet to catch on.

To facilitate the use of Semantic Web several data formats emerged that allow small units of data to be embedded in currently existing HTML web pages. Early 2007 the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG) began working on a new revision of HTML: version 5 and XHTML version 5 [2]. While some considerable steps are being taken in modernizing HTML, nothing remotely resembling Semantic Web integration has been considered in the latest drafts [3].

2. APPROACH
In this paper, we will try to find a way to make modifications to (X)HTML 5 to improve RDF support. This will be accomplished by following the following steps:

1. By comparing microformats, eRDF and RDF. This part is mainly a literature study. The results taken from these comparisons will help determine what features will be needed in the proposed HTML 5 version.
2. Finding out what will change in HTML 5 and in what way these changes related to RDF integration; mainly a literature study. This also includes finding out how the differences between HTML and XHTML will impact the final design.
3. Finding out how HTML5 can be adapted for RDF integration. This is where the main research happens; the results will depend on the results of the previous two points.

3. THE SEMANTIC WEB
As mentioned, the idea behind the semantic web is to create web pages that contain the same data as available now on the ‘normal’ web. However, there is a vast amount of different data on the web, that there cannot be a predefined set of structures in which data should be formatted. Two cooperating formats have been developed to solve this problem. First there is the Web Ontology Language (confusingly abbreviated to OWL), which can be used to describe schemas in which data can be described [4].

The second is the Resource Description Framework (RDF), in which the actual data is represented according to OWL schemas [5, 6]. However, RDF is only a model, not a syntax. By using different syntaxes, RDF can be put to use on a wide variety of places.

The principal concept of RDF is data is stored in triples: {subject, predicate, object}. The subject is always a reference to an existing thing. This thing can be created in RDF, or one could refer to objects by using its Uniform Resource Identifier (URI). The predicate too is always a reference, defined in an ontology. The object however, can be either a reference to a thing, or a literal value.

3.1 Basic Notations
If we would describe this paper, we could do something like in figure 1. This example also demonstrates N-triples: the most basic format in which RDF can be represented. An N-triples file consists of nothing more than triples in the order subject, predicate, object, terminated by a full stop. URIs are enclosed in angle brackets and literal values in quotation marks [7, 8].

Figure 1: RDF example in N-triples notation

The ontology used in this example to define title and creator is the Dublin Core, a much used, standardized set of tags for resource descriptions [9].

3.2 New Subjects
In RDF, one does not need only to describe things that already exist; it is also possible as it where to “create” new things to describe. In N-Triples, this is done by referencing the file the description is in, but adding a unique identifier. If the N-Triples file in figure 2 would be accessible at http://example.org/people, statements could be made about this new subject in other RDF files, such as in figure 1 [10].

Figure 2: RDF example containing a new subject

3.3 Blank Nodes
RDF assumes an open world. Therefore it is important that incomplete data can be put in N-Triples files. In RDF data can be added to so called blank nodes. These blank nodes can be used in both the subject and the object. In N-triples notation these are marked with “:_” combined with an identifier. So for example, if it is unclear who wrote the paper, but their gender is known, one can put this in an RDF statement. In figure 3 the
blank node _author is used as the object of the first statement and as the subject of the second statement. [8]

 pleasures or resource attributes one can use these blank nodes [8].

 RDF in HTML

 4. RDF in HTML

 4.1 Microformats

 Microformats started out as an initiative to develop a way for people to include personal information on their homepages, [24, 25]. A method was developed that allows vCard data in the source of websites. vCard is a popular format for storing business cards, introduced back in 1998 [26]. This new embedded format is called hCards. Users can link these business cards, creating a friends/relations network which is independent of commercial profile sites such as Facebook or MySpace. This network is called the XML Friends Network (XFN) [27].

 After the relative success of the hCard format, the microformat community has developed more formats. The most well-known is iCalendar, based on iCalendar, for embedding calendar data in websites [28].

 The existing microformats are roughly divisible into two categories: Those that describe something that is on a webpage (like hCard en hCalendar) and those describing the webpage itself, such as the rel-tag microformat. This format allows for tags to be set for a webpage, much like many bloggers do to categorize their posts. Although all microformats within these categories share a common philosophy, each microformat requires its own specification. Microformats do not have a connection to RDF and the semantic web, this might only seem this way because XFN is often compared to FOAF, one of the most widely used RDF formats, used to describe people in a similar way [29, 30].
eRDF (embedded RDF) is a format in which the basic features of RDF can be embedded in HTML files. It was created in 2005, partly inspired by microformats. Even though eRDF is based on proper RDF, and microformats are not, their syntaxes are quite alike. They both use the meta and link elements in the HTML head and use class attributes to insert predicates. [23, 32]

To add eRDF data in a HTML file, one first has to announce the presence of eRDF data. This is done by adding the eRDF profile to the HTML head tag:

```html
<head profile="http://purl.org/NRT/erdf/profile"/>
```

Unlike RDF/XML or N3, in prefixes can only be declared at the beginnings of a HTML file, specifically in the head in link tags:

```html
<link rel="schema.dc" href="http://purl.org/dc/elements/1.1/"/>
```

Now the first major difference from proper RDF comes to light: triples in eRDF can only be in one of four following forms:

- The subject of the triple is the current HTML page
- The object of the triple is the current HTML page
- The subject is a unique identifier on the current page
- The object is a unique identifier on the current page

Adding to that, the syntax for these four is completely different. If the subject is the HTML page and the object is a literal value, the triple is added in a meta element in the HTML head:

```html
<meta name="dr.title" content="RDF in HTML"/>
```

If the subject is the HTML page and the object is a reference, the triple is added in a link element in the HTML head:

```html
<link rel="dc.creator" href="#gdavis"/>
```

If the object is the HTML page, the triple is added in a link element as well:

```html
<link rel="foaf.made" href="#gdavis"/>
```

While a bit confusing, the above statements are all conform the HTML standards. The HTML link element is used to add custom links to a HTML file, by using rel or rev these links can go two directions. The meta element is used in HTML to add custom metadata to a HTML file, therefore basically has the same behavior as an RDF triple with the HTML page as subject. Note that there is no rel or rev in the meta element, but this is not a problem because a literal cannot be a subject in RDF.

In eRDF there are no blank nodes, all nodes that are described need to have an identifier. These are added by using the HTML id attribute. It doesn’t matter what kind of HTML element the identifier is in. Like in microformats, the object can be in either the content of a HTML element, or in specific attributes. If the object is a literal, the object goes in the content field of an element and the predicates are added in class attributes. These can only be used with their prefix as declared in the head, there is no way to use full URIs. It is also possible to overwrite the value that is displayed in the HTML page, by using the title attribute.

If the object or the subject is a reference, it goes in the href attribute and the rel or rev attribute contains the predicate, here the object or subject can only be referred to by the full URI.

If any of these elements overlap, they can be combined, as long as remains clear which object belongs to which predicate. This and other discussed features are shown in figure 11 below.

5. (X)HTML 5

Since the dawn of the web, webpages are written in HTML (Hypertext Markup Language). The first version of HTML, created by Tim Berners-Lee himself, dates back to 1991. There have been steady developments, first by the Internet Engineering Task Force (IETF)[33], later by the W3C. But the latest version (HTML 4) dates back to 1998. In 2001 a few amendments were made to HTML 4 and XHTML 1.1 was released. Both introduced only minor changes. XHTML 2 is in development since 2002, but development has almost come to a stop. There is much criticism on XHTML 2 and it has become very unpopular even before coming close to a final release[34]. This means that the language in which today’s webpages are described in, dates back more than 10 years. That is a lot, especially in computer science terms. In 2004 the Web Hypertext Application Technology Working Group (WHATWG)[35] was formed by individuals from three major browser vendors: Opera, Mozilla and Apple; in response to the lack of initiative from the side of the W3C. The WHATWG immediately began working on a new version of HTML. In 2007 the newly formed HTML working group at the W3C
adopted the work done by the WHATWG and continued cooperatively to work on the standard. HTML 5 will continue the HTML 4’s incentive to drop all presentational elements. For example, the large element (used to indicate that text should be larger than normal) will be dropped. The small element however will be kept, but will be redefined to indicate small print i.e. on the bottom of a website. Whether this text is then rendered in a smaller font should be defined with cascading style sheets.

Furthermore HTML 5 will offer more functionality that is useful for today’s webpages. The most important new elements are canvas for client-side 2D drawing, video and audio for embedding video and audio files in HTML and some new document related tags such as section, article, figure, header, footer and more. HTML 5 will also add some new functionality to already existing elements, such as new input types for the input element (email, url, date and time among others) [3, 36].

There are two new elements which have a pure semantic application: the time element can be used to denote times and dates and the meter element can be used to denote numerical values along with a minimum and maximum value [3, 36]. The inclusion of these two new elements is the closest HTML 5 comes to adding semantic web features.

HTML 5 will come in two flavors, HTML and XHTML. To prevent compatibility issues, HTML 5 and XHTML 5 are designed to be as much alike as they could be. XHTML is based on XML, and webpages written in XHTML must therefore be valid XML. XHTML also uses XML attributes where available (i.e. the xml:lang attribute instead of the lang attribute) [3].

### 5.1 Microformats and HTML 5

HTML 5 is currently in development and there has been no decision if anything related to be included in the spec. However, HTML 5 is being developed in dialogue with several browser vendors. Firefox version 3 by Mozilla already has some support for microformats build in and there are rumors that Microsoft is building the same functionality in for their next version of Internet Explorer [37, 38]. Therefore it is not unthinkable that microformat related specs might be added at a later date.

Microformats in their current form will continue to work in HTML 5, unless the HTML 5 specifications change. Microformats currently use the abbr element to denote dates and times, in a way that closely resembles the time element in HTML 5. Microformats specifications will probably change to use the time element when HTML 5 makes it to recommendation status.

### 5.2 eRDF and HTML 5

eRDF in its current form is not compatible with HTML 5. For example, the profile attribute in the HTML head element will be removed. eRDF also relies heavily on rev attributes, they too will be removed. The latest changes to the eRDF specifications date back to 2006, it is as yet unknown if there will be a revision by the time HTML 5 makes it to recommendation status.

### 5.3 REQUIREMENTS

The aim of embedding RDF in webpages is to promote the usage of semantic web elements by individuals who have no experience with semantic web. Normal HTML is used a lot more than XHTML. Implementing RDF in XML is quite easy without making changes to the HTML spec due to the extendable nature of XML. The focus therefore must be on finding an implementation that works for HTML.

#### 6.1 Features

To use the full extent of RDF functionality it is preferred that as many RDF features as possible described in chapter 3 are included. This includes using shortened statements, prefixes, base paths and blank nodes. Furthermore the implementation should be useful from a HTML point of view, meaning that information in RDF should be visible in the resulting webpage, much like the microformats specifications.

##### 6.1.1 Triples

In RDF the subject always is a reference (a link or URI). This has the advantage that this is data that doesn’t need to be visible in HTML. The easiest solution is adding a new attribute to HTML elements in which subjects can be declared. To avoid changing the HTML 5 specifications too much and keeping in line with the way current HTML elements are used, this attribute should be added to existing HTML elements.

The predicate too is always a reference, however here the XML/RDF implementation cannot be mimicked. The only way to add this data in HTML is to add an attribute in which the predicate can be declared. This implementation however raises a few issues. By only using attributes, there is no way to enforce them in being used in the proper order. We want to construct our implementation in a way that if a webpage is written in valid HTML, the contained RDF data is valid RDF as well.

In table 1 below all the possible tag nestings are shown. A “+” denotes the intended order of the tags, a “-” denotes a nesting that results in invalid RDF. This leaves a few special cases:

- A. Using a predicate without a subject would also not result in proper RDF, but this could be used as shortcut for describing the current document.
- B. Nesting a subject in another would normally not result in valid RDF data, but allowing this could be useful in HTML. If a webpage for example contains an article which we want to describe, and that article contains an image which we also want to describe, we’d have this situation. Everything can always be described by defining the subject to be the one referenced in the innermost subject.
- C. A subject without predicate or object can just be ignored.
- D. Used for shortening of statements, see 6.1.3 below.
- E. If there is no object inside a predicate, the value field of the element can be used as object. If there is no object and no value field, the object can be assumed to be a string with zero length. This behavior resembles the behavior of the time element in HTML 5 and can be used as a handy shorthand so that no useless subject elements without attributes are needed.
- F. Used to create blank nodes, see chapter 6.1.6 below

<table>
<thead>
<tr>
<th>Table 1: Possible tag nestings</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1.2</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>Root-node</td>
</tr>
<tr>
<td>Subject</td>
</tr>
<tr>
<td>Predicate</td>
</tr>
<tr>
<td>Object</td>
</tr>
</tbody>
</table>

The only solution for this issue is to add RDF data in new elements instead of in new attributes. Putting the object in an always available attribute and putting the predicate in a new element matches table 1 exactly.

The object can either be a reference, or a literal (no link but string of characters). Usually the object doesn’t need to be displayed in a webpage if it is a reference and should be
displayed if it’s a literal. To make this distinction the simplest solution is to introduce a new element. Like the *time* element in HTML 5, the value is always the one displayed. If there is data present in the element that data should be the object, if there is not the value is the object.

The term *object* is a quite confusing term, especially for someone who has no RDF knowledge. Therefore the object elements will be called “data”. Predicate will be shortened to “pred”. A complete triple will look like the one in figure 12 below.

**Figure 12: A triple in the proposed notation**

This can be shortened by using the rules A and D from table 1 above. The result looks like the triple in figure 13 below.

**Figure 13: A shortened triple in the proposed notation**

### 6.1.3 Shortening

Statement shortening will work just like in RDF/XML. A subject can contain more than one predicate and a predicate can contain more than one objects. There also are some other tricks that can be used to make the RDF easier to read and write:

For example, the *about* attribute can be used on any element, so if there is only one *pred* element needed, these can be combined, as is shown in figure 14 below.

**Figure 14: Combining the new features**

The value for the object is taken from the value field of a *pred* element, but all HTML tags inside this *pred* element should be ignored. This allows microformat-like combinations, like combining a given name and a family name to a name. This is shown in figure 15 below.

**Figure 15: Nesting *pred* elements**

### 6.1.4 Prefixes

In XML or XHTML prefixes would be build in by using the *xmlns* attribute in XML. Since this doesn’t work in HTML, something has to be constructed that works with the syntax normal (non XML) HTML uses. The best place for the prefixes is to put them in the webpage header, so that the HTML code stays clean and as close to HTML as possible. This won’t offer as much flexibility as RDF/XML or N3 where prefixes can be defined and redefined everywhere in the document, but it should suffice for the relatively simple RDF in webpages. The *link* element is the obvious place to put the prefixes, much like is the colon-notation in XML is rather confusing and requires extra steps to parse. This can be done neater by properly using the already available attributes in the *link*. The prefix should go in the *title* attribute, the *href* in the *href* attribute and *rel* should set to “prefix”. An example is given in figure 16 below. When the document is parsed it’s easier to look for all prefixes if the *rel* value is a fixed value.

**Figure 16: Proposed prefix notation**

The declared prefixes can then be used just like prefixes in most other RDF notations, by using the colon-notation.

### 6.1.5 Base paths

HTML already has a method for specifying a base path for URLs in a document. The *base* element can be declared in the HTML *head*. If there is such an element present, all relative URLs in a webpage must use this URL as the base for calculating the complete path. RDF parsers must honor this base path and use it as well. Using this method means that the base path cannot be redefined, just like the prefixes. This is however not considered a problem.

### 6.1.6 Blank Nodes

As mentioned before, blank nodes can be described the same way as in XML/RDF, by putting *pred* elements in a *data* element, see figure 17 below. Note that the *data* element cannot be omitted if blank nodes are used, this would cause a conflict with one of the statement shortening methods, shown in figure 15 above.

**Figure 17: A blank node in the proposed notation**

Not all features of blank nodes are available because of the restrictions HTML has. Blank nodes cannot be referred to by an identifier, because in HTML anything that gets an identifier attribute is automatically exposed, making such a node not “blank” anymore.

### 6.1.7 Number and Boolean literals

The usage of number and boolean literals won’t be possible due to the syntax HTML uses. All attribute values always have to be enclosed in quotation marks ("), so there is no easy way to show the difference between a string “1” and the number 1. This is not considered to be a problem, number and boolean literals are a pure N3 feature, not an RDF feature. Not using number and boolean literals will in the worst case require an extra step for parsers.

### 6.2 Restrictions

All basic RDF features can be used in the proposed notation. Albeit that some features (prefixes, base paths and blank nodes) are a bit restricted. But there are three features implemented in microformats and eRDF that cannot be used in this proposed notation:

First of all, microformats and eRDF both have specific shortcuts for some HTML elements, such as the anchor and *img* elements. These shortcuts are meant to prevent double data on a webpage. Figure 18 demonstrates this feature in both microformats and eRDF.

**Figure 18: Examples of element specific shortcuts in microformats and in eRDF**
To achieve the same result in our proposed notation, the URLs in the src and href attributes have to be duplicated in the data element. An example is given in figure 19.

```html
<html>
<head>
  <title>RDF in HTML</title>
  <link rel=prefix title=foaf href=http://xmlns.com/foaf/0.1/>
  <link rel=prefix title=dc href=http://purl.org/dc/elements/1.1/>
</head>
<body>
  <h1>RDF in HTML</h1>
  <a href="example.org">My Site</a>
</body>
</html>
```

Figure 19: Referencing an image and a URL

In both microformats and eRDF it is possible to display a different value than is actually stored in the subject, even if the subject is a literal. In our proposed notation this is omitted in favor of simplicity and readability. However this effect can still be achieved by using Cascading Style Sheets (CSS).

### 6.3 Putting it all together

To show how a complete page would look like, figure 20 demonstrates how the eRDF from figure 19 would look using the proposed notation and HTML 5.

```html
<!DOCTYPE html>
<html>
<head>
  <title>RDF in HTML</title>
  <link rel=prefix title=foaf href=http://xmlns.com/foaf/0.1/>
</head>
<body>
  <div about="#gdavis" id="gdavis" about="#gdavis">
    by
    <pre>
      <h1><pred rel="dc:title">RDF in HTML</pred></h1>
    </pre>
    <a href="example.org">My Site</a>
  </div>
</body>
</html>
```

Figure 20: Complete HTML 5 page with RDF data

### 6.4 Compatibility with HTML 4

One of the goals of HTML 5 is to create this new version with as much compatibility with HTML 4 as possible. Our design mimics this behavior the following ways:

HTML elements and attributes are used where possible, but without using these in a way they were not intended to be used, like both eRDF and microformats do. The notation is design in a way that all data that should be visible is put in the value fields of tags, all data that should not be visible is put in attributes. So if a browser doesn’t recognize the used tags and attributes, or chooses to ignore them for another reason, the webpage is still rendered correctly. The data element should always be closed with a separate </data> tag, even if the value field is empty: so using <data rel="..." /> in normal HTML is forbidden, just like it is forbidden to close many other HTML elements that way.

Furthermore, there are individuals who think that the anchor element should be dropped in HTML 5 or a later version and allowing the href attribute to be used on each element. This is new feature is already present in the XHTML 2 drafts. To be prepared for this, the rel attribute is used in the pred and data elements, instead of the href attribute.

### 6.5 Parsing

RDF in HTML 5 has no use at all if RDF data is not extractable from HTML 5 pages. This notation therefore not only must be easy to write, but also must be easily parsable. XML and XHTML are easily parsable by the many XML parsers that exist, however we are dealing with HTML here. Since the data resides in webpages, the most data will be extracted by webbrowsers. For example saving contact information from a website to your contact manager or adding events and dates published on website to your agenda.

Browsers keep a model of the webpage in their memory, called the DOM (Document Object Model). The DOM of a webpage can be seen as a tree, with the html element as its root and each nested element as a branch or leaf. The DOM can be parsed in various ways offered by browsers. The most common is ECMAscript (usually known as JavaScript).

In Appendix B an algorithm for extracting RDF data from a HTML 5 DOM is given. This algorithm may look complex, but is in reality simpler than algorithms required for extracting RDF data from RDFe or extracting data from microformats.

The algorithm for eRDF needs a more intelligent system for handling prefixes, it needs to take care of document related triples in the HTML head and it has to look for data in different places in different elements (see Figure 18: Examples of element specific shortcuts in microformats and in eRDF). Microformat parsers face the same obstacles, but also have to deal with the fact that microformats are not based on RDF, so a special parser has to be written for each specific microformat.

### 6.6 HTML/XHTML

This new way of was designed to be used with normal HTML 5, but what about XHTML 5? All added functionality (new elements and attributes) could be used in XHTML the same way as in HTML. The algorithm in appendix B is purely DOM based, so that would work exactly the same. However, if one uses XHTML, it seems a waste not to use the features XML provides. By using proper xml namespaces, RDF/XML data can be inserted into XHTML. This data can be easily extracted by any XML parser.

### 7. CONCLUSIONS

In this paper we present an extension to HTML 5 which can be used to include RDF data in webpages. This format is based on eRDF, microformats and RDF/XML and tries to solve the biggest issues with those formats. The format is designed to work with normal HTML (not XHTML). Because some new HTML elements and attributes are introduced, RDF can be inserted in a way that is easy to write. To keep the format clear and easy to understand some functionality present in microformats and eRDF has been omitted, but there is nothing expressible in RDF that can be described in eRDF and can’t be described in the proposed format. Adding RDF data to a webpage is rather useless if the data can’t be extracted. This is easier to do for our proposed format than from eRDF or microformats. An algorithm to extract data is given in appendix B.

### 8. FURTHER WORK

The work presented in this paper is all theoretical. It needs testing and the best way to do that is to write a sample parser. This parser should take a HTML 5 DOM as input and returns the triples contained in that DOM.

In a later phase, a formal proposal for inclusion of these proposed elements could be written and submitted to the W3C and WHATWG.
ACKNOWLEDGEMENTS

I would like to thank Maarten Fokkinga for his guidance and suggestions and valuable feedback during this course of this research.

REFERENCES


APPENDIX A: RDF IN HTML 5 EXAMPLE FILE

This appendix shows how the example file from the eRDF website [23] would look translated to RDF in HTML 5.

```html
<!DOCTYPE html>
<html>
<head>
  <base href="http://example.com/about" />
  <link rel="prefix" title="dc" href="http://purl.org/dc/elements/1.1/" />
  <link rel="prefix" title="foaf" href="http://xmlns.com/foaf/0.1/" />
  <title>Anna's Homepage</title>
</head>
<body>
<h2>About me...</h2>
<p id="anna" about="#anna">
  Hi, I'm <pred rel="foaf:name">Anna Wilder</pred>.
  <pred rel="foaf:depiction"><data rel="pic.jpg"><img style="float: right" src="pic.jpg" alt="A picture of me" /></data></pred>
  <pred rel="foaf:nick">You might know me from IRC as <data>wildling</data> or sometimes <data>wilda</data>. You can email me at <pred rel="foaf:mbox_sha1sum">anna {at} example.org</pred>.
</p>
<footer style="display:none">
  <pred rel="dc:creator">Anna Wilder</pred>
  <pred rel="dc:title">Anna's Homepage</pred>
  <pred rel="foaf:homepage" about="#anna"> <data rel="#"></data> </pred>
  <pred rel="foaf:made" about="#anna"> <data rel="#"></data> </pred>
  <pred rel="foaf:maker" about="#"> <data rel="#anna"></data> </pred>
</footer>
</body>
</html>
```

APPENDIX B: SAMPLE PARSING ALGORITHM

This appendix describes an algorithm for extracting RDF data from a HTML 5 webpage.

1. Look for a base element in the head.
2. First locate all prefixes by listing all link elements in the head, keeping only those which have their rel attribute set to “prefix”.
3. Do a depth-first search through the DOM, locating all prefix elements. If a prefix element is found, look inside this element as well. If a data element is found inside a prefix element: don’t look inside those.
4. For each prefix element:
   a. Create a new triple
      b. Starting with the prefix element, look for an about attribute, if none is found, recursively look in the nodes parent. If one arrives at the root element and it too also has no about attribute, use the documents URL as the triples subject.
      c. If a about attribute is found, set the value of the about attribute as the triples subject.
      d. Do a breadth-first search for any data elements. If a data element is found, look inside that element too. If a prefix element is found inside a data element do the following:
         i. Create a new blank node.
         ii. Set this blank node as the object of the triple.
         iii. For each prefix element found, create a triple, set the blank node as the subject of the triple.
         iv. Repeat step 4 for each prefix element.
      e. Else if no prefix elements were found, for each data element, make a copy of the triple. If the data element has a rel attribute set, set the object of the triple to the data in the rel attribute, set the triple to be a “reference triple”. If the rel attribute is not set, extract the text from the data element’s value field and set this as the triple’s object. Set the triple to be a “value triple”.
4. For each stored triple do the following:
   a. If the subject of the triple uses a known prefix: apply this prefix to the subject.
   b. If the subject of the triple has a relative path, but is not an id-reference only and a base element was found: apply the base path to the subject to get the absolute path.
   c. Repeat steps 5a and 5b for the predicate and if the triple is a “reference triple” also repeat these steps for the object.