

## CRDF: Cascading RDF

Note: this document is not intended to be a formal specification, but to illustrate a concept and serve as the basis for such specification. It tries to define a CRDF format in an unambiguous, even if informal, way.

### 0. Rationale: Issues with current solutions

Before starting to define a new language or syntax, it is important to check the currently available syntaxes for similar purposes, and see why they fail for the problem at hand. The main approaches to be considered are RDFa and Microformats. There are issues with each of these, and also some which are shared by both. Let's tackle them separately. Some of these approaches' strong points are also reviewed, as they can serve as an inspiration to develop a solution that avoids all of the issues but still gathers most of the strong points.

#### 0.1. Issues and strong points in RDFa

The main issue for introducing RDFa into HTML5 derives from the facts that RDFa was designed for XHTML and the XML language family; and that HTML5 is *not* XHTML (despite it supports an XML serialization). This yields the problem of XML namespaces and CURIEs.

On the other hand, some sort of namespacing or vocabulary disambiguation is required to ensure that distributed definition of vocabularies is allowed without the risk of clashes between vocabularies.

Next, it requires a lot of extraneous mark-up when it comes to describing relatively complex stuff in a document (such as the iguana collection example case posted on the WHATWG mailing lists). This is no surprise once we know that RDFa introduces several attributes and redefines some others on the host language.

To top it up, RDFa subtly fails on one of its purposes: avoiding repetition of the data in human- and machine-readable forms. While it works on the general case, RDFa miserably fails in two specific (but potentially quite common) cases: when a single value should be used for properties for two or more subjects, or when the value is an attribute rather than an element's content (maybe excluding links' href, thanks to the rel/rev overloads). For example, imagine the following document, hosted at

<http://www.example.org/> (add the ugly *xmlns* to the root element if you're too X-centric, otherwise it should be valid as both HTML (4.x Transitional and 5) and XHTML (1.0 Transitional)):

```
<html>
  <head>
    <title>My homepage</title>
  </head>
  <body>
    <h1>Eduard Pascual's projects</h1>
    <ul>
      <li><a href="proj1/">First project</a></li>
      <li><a href="proj2/">Second project</a></li>
    </ul>
    <small>This page and all projects linked from it, by
      <a href="/">Eduard Pascual</a>, are licensed under a
      <a rel="license"
        href="http://creativecommons.org/licenses/by/3.0/">CC
        Attribution license</a>.</small>
  </body>
</html>
```

I'd like to be able to generate the following triples (please, forgive me if my understanding of the Dublin Core and CCREL vocabularies is not accurate; my intention here is just to illustrate a point):

```
<> <http://purl.org/dc/elements/1.1/creator> "Eduard Pascual"
<> <http://creativecommons.org/ns#attributionName> "Eduard Pascual"
<> <http://purl.org/dc/elements/1.1/title> "Eduard Pascual's homepage"
```

```
<> <http://creativecommons.org/ns#attributionURL> <http://www.example.org>
<> <http://www.w3.org/1999/xhtml/vocab#license>
    <http://creativecommons.org/licenses/by/3.0/>
<http://www.example.org/proj1/> <http://www.w3.org/1999/xhtml/vocab#license>
    <http://creativecommons.org/licenses/by/3.0/>
<http://www.example.org/proj2/> <http://www.w3.org/1999/xhtml/vocab#license>
    <http://creativecommons.org/licenses/by/3.0/>
```

The tough ones are the last three triples: is there any RDFa I could add to my mark-up that re-uses the `<a>` with the license to describe all three subjects (the page itself and the two projects)? I know I could wrap `<span>`s around it and define the properties for each subject with them, but this would alter the structure of the document, so it's essentially a hack, and might arise issues on less trivial cases (such as messing up with styles or scripts that rely on the document's structure).

There is also a secondary issue, related to RDFa: for some reason, several proponents of including semantics into HTML5 might be defending this particular solution a bit too strongly: sure, it would be advantageous if an already existing solution could be applied to the problem, reusing also some of the related tools (such as RDFa consuming software). But a solution can't be chosen from just this advantage: it must also solve the problem in the current context (for example, it has to be friendly to non-X HTML), and RDFa's CURIEs are quite a bit of a drawback to compensate for current deployment. It might be a good solution for XHTML, since it was designed to be so; but still remains to be seen if it can be even a passable solution out of the "X-world".

The main strong point of RDFa is that it allows describing virtually anything conceivable. This is due, however, to RDF's own "triples" concept; so the strong point itself for RDFa would be the fact that it's just a serialization of RDF.

## **0.2. Issues and strong points in HTML5's Microdata**

There are many parallels between RDFa and HTML5's Microdata. Actually, Microdata could be described as an attempt to solve the main problems addressed by RDFa with a syntax that is more tagsoup-friendly. As Microdata is still a non-stable part of the HTML5 draft, anything said here could change before anyone gets to read this document; so we will focus only on the main conceptual issues (more specific problems are the ones that are likely to be solved in a near future).

- Ambiguous property identification: Microdata relies on a reversed domain notation to identify properties; in an attempt to keep XML Namespaces out of non-XML markup. However, it is currently not possible to describe that a given vocabulary is based on another, so shared properties should be treated as fully equivalent; or to properly distinguish between multiple vocabularies that are hosted in the same domain.
- Just like RDFa, it is inline-only: there is no way to describe semantics based on the structure of the document; other than explicitly annotating each metadata instance.
- It can't represent the full RDF model. While this may seem purely theoretical, it means that the range of use cases that can be addressed with Microdata is more limited than those addressable by full RDF serializations, such as RDFa or eRDF. More specifically, it has been highlighted on the HTML5 discussion lists that cyclic graphs can't be handled with Microdata.

## **0.3. Issues and strong points in Microformats**

The main issue of relying on Microformats is centralization: it prevents the risk of clashes between different formats (although there is a small risk that a microformat clashes with some of the billions of pages written before it existed); but it also prevents distributed definition of vocabularies (for example, having the iguana collection case go through the Microformats.org community would be, in the best case, nightmarish: the community will focus on stuff more widely needed than iguanas).

A secondary issue with Microformats is that, in principle, it forces the human-readable content to be used as the machine-readable content. While this might work for the general case, there are cases where this is just not possible (for example: try to express a date in a way that both a machine and a

user used to the Chinese calendar can understand). A workaround based on the abbr element and its title attribute were developed to address this, but at least one case is known<sup>1</sup> to have dropped Microformats just because of the accessibility issues arisen from this hack.

On the other hand, Microformats shine by their simplicity: they just reuse the class attribute, and in a way it was, in theory, supposed to be used.

#### **0.4. Issues of shared by all these approaches**

The first shared issue on all these formats is the blatant limitation: they are intended for embedding metadata directly on X/HTML documents, *exclusively*. Ideally, there should be a format that allows direct embedding, of course, but also other forms of defining the metadata. For example, again on the iguana collection case, let's put a collection with 20 or more iguanas. It would be quite reasonable to have a listing with a summary (just the name, a picture, and a link to the iguana's own page); let's assume it's marked up as a two-column table, with one iguana per row, the picture on the left column and the name, linking to the page, on the right column. Each of these formats would require stating in the mark-up "this is an iguana; this is the iguana's picture; this is the iguana's name; and this is the iguana's main page" once for each iguana. The point here is that it would be ideal to be able to state "each row is an iguana; the first column has the iguana's picture; and the second column has the iguana's name, linked to the iguana's main page". Of course, once on the iguana's main page, it might be preferable to explicitly embed the "this is the iguana's picture", "this is the iguana's whatever", etc where appropriate on the page (keep in mind the difference between several items with some properties each and one or few items with several properties).

## **1. Introduction: CSS + RDF = CRDF**

Aware of the benefits RDFa has from its RDF roots; it seems a good idea to use a RDF-based syntax for adding metadata to HTML5. The new approach, however, must be agnostic about the language syntax, since HTML5 supports both tag-soup and XML syntaxes, and should attempt to cause the minimum possible impact on the host language (thus not obstructing potential paths of evolution for future versions of HTML). Due to RDF's nature, namespaces are almost a must; yet the format shouldn't require the host language to deal with the namespace declarations, since this would mess up with the tag-soup form of HTML5.

Next, the format needs to allow defining the metadata inline within the document, but also allow tying it to the structure (see the iguana summary versus iguana main page example above).

To top it up, the format should be as easy to use as possible (ideally, as simple as Microformats), but without sacrificing expressiveness or requiring centralization.

One of the critical points that have inspired this document was the question: "how could a Microformats-like solution be decentralized?" Due to the heavy use Microformats make from the class attribute, the first thing that came to mind was to do something CSS-like. The name CRDF then became obvious, and work started defining how things would work.

CSS syntax is reused as much as possible, most prominently with selectors, declaration blocks, and the "property: value" structure. Furthermore, CSS already provides a solution to deal with RDF's namespaces, since it has support for defining namespace prefixes (these are normally only used for selectors in actual CSS, but reusing the syntax for CRDF properties seemed quite natural and a significant benefit arose from it).

To spice it up, a shorthand syntax was thought for the most common case (using the contents of the selected elements for the value of properties), and some tweaks were made to address some issues.

---

<sup>1</sup> This refers to BBC dropping the hCalendar format as announced on [http://www.bbc.co.uk/blogs/radiolabs/2008/06/removing\\_microformats\\_from\\_bbc.shtml](http://www.bbc.co.uk/blogs/radiolabs/2008/06/removing_microformats_from_bbc.shtml).

## 2. Syntax

The basic syntax is drawn from CSS, and only some tweaks are made to it. A CRDF document or *sheet* is composed of zero or more *statements*. A statement is either an *at-rule* (as defined by CSS3 Syntax)<sup>2</sup> or a *ruleset*. A *ruleset* is formed by an optional *selector* (as defined by CSS3 Syntax and/or CSS3 Selectors), followed by the opening brace “{”, followed by a semicolon “;” separated list of zero or more *declarations*, followed by the closing brace “}”. A declaration is a *namespace prefix*, followed by the namespace separator (vertical bar “|”), followed by a *property*, followed by a colon “:”, followed by a *value*; or a *shorthand declaration*. The value is further described in the next section. A *shorthand declaration* is the same as a *declaration*, but omitting the colon “:” and the *value*; the ‘contents’ keyword (with its CSS meaning) is assumed as the *value*.

More formally:

```
CRDFsheet:      at-rule* ruleset*
at-rule:        (as per CSS3 Syntax)
ruleset:        selector? { declarations }
selector:       (as per CSS3 Selectors)
declarations:   declaration
                declaration ; declarations
declaration:    prefix|property: value
                prefix|property (short-hand syntax)
prefix:         (as per CSS3 Namespaces)
property:       (a valid identifier as per CSS3 Syntax)
value:         (described later in this document)
```

Although not explicitly described, *whitespace* and *comment* tokens are always allowed between any other two tokens.

For obvious reasons, there are some characters that are not allowed un-escaped on some contexts. For selectors, this is exactly as defined by CSS3 Selectors. For namespace prefixes, allowed characters are defined by CSS3 Namespaces. For properties no colon “:”, semicolon “;”, or closing brace “}” can appear un-escaped. In addition, it is recommended escaping any other character that would interfere with readability. For values, only semicolon “;” and closing braces “}” require escaping. In all cases, literal backslashes will always need to be escaped; otherwise they’d be escaping what follows them.

### 2.1. CRDF Values

Properties defined via CRDF can take several forms, as described here:

- **contents** keyword: The actual value for the property will be the contents of the currently selected element. When the contents of the element are just text, the text will be taken as a string literal by default; but when it includes mark-up it will be taken as a XML literal (see section 2.4 for details about how this is handled on non-X mark-up languages). This keyword is the implicit value used for any property when the short-hand syntax is used.
- Literals: Decimal numeric values (either integer or fractionary, using the dot as the decimal separator), and strings enclosed on double quotes can be given as literal values for CRDF properties.
- **attr(name)**: This must take an attribute name as *name*. The value applied for the property will be the value taken from the corresponding attribute on the selected element<sup>3</sup>. If the attribute’s value can’t be computed (for example, *name* is not a valid identifier, or it doesn’t represent any attribute), it will be taken as if an empty value had been given (such as in `<img alt="" ...>`).
- **calc(expr)**: (see <http://www.w3.org/TR/css3-values/#calc> for details; with the exceptions that in CRDF **attr()** expressions can be nested inside **calc()**, and it represents arbitrary numeric values rather than lengths. A complete definition of this notation for CRDF will be provided in later versions of this document.)

<sup>2</sup> It still needs to be defined which at-rules make sense and hence are allowed in CRDF, and which ones don’t.

<sup>3</sup> Should attribute default values be used when no explicit attribute is given in the mark-up?

- **concat(values)**: This takes a comma-separated<sup>4</sup> list of expressions (each of which can be a literal, or the **attr()** or **calc()** notations), which are converted to strings and concatenated in the given order to form a single string.
- **url(URI)**: This must take a valid URI as the parameter; but **calc()** and **concat()** notations can be nested inside it (so, a URI could be generated by concatenating some attributes and literal values, for example).
- **reversed** keyword: This keyword can be added after any of the values described above (whitespace is required if the keyword could otherwise be taken as part of another token). When used, it reverses the subject and object of the property being expressed (so the implicit or given subject becomes the object, and the value given by the property becomes its subject).<sup>5</sup> If no value is given with this keyword, then “**contents**” is assumed (for consistency with the short-hand syntax).
- **prefix|type(value)**: This functional notation accepts any other value as the **value**, and should give **prefix** and **type** values so that they map to an actual type definition. This is used to give an explicit type for an object, overriding any implicit or default type described by this document.

## 2.2. Rationale on the differences with CSS syntax

It’s quite clear that the most common use-case will be reusing the content of an element for one or more properties. The shorthand syntax defined allows minimizing the typing for such cases, but doesn’t force any extra burden for the rest of cases. See the following example:

```
@namespace foo http://www.example.org/foo/
a.fooish {
  foo|description;
  foo|URI: attr(href)
}
```

The “description” property is taken from the contents of the <a> element; but this doesn’t interfere with using the href attribute for the “URI” property (both on the “foo” namespace”).

The “|” in the middle of properties may seem weird at a beginning, but it’s the natural namespace separator introduced by CSS Namespaces, so it is the best fit for qualifying the properties with namespace prefixes. The alternative, using full URIs for properties, is just horrific: all http URIs have, at least, one colon, and tend to have several punctuation characters (mostly slashes and dots, but others may show up from time to time): that would be a nightmare to parse if un-escaped, and a nightmare to author and read if escaping all that stuff. As a side benefit, this syntax yields some fool-proofing: if an author tried to put CSS and CRDF rules in a single sheet, each parser would get the relevant stuff: the CSS parser will puke at the properties containing “|” and skip them, and the CRDF parser would ignore those that have no namespace. Relying on default namespaces is not allowed by the syntax described above, only in order to prevent valid CRDF properties that have no “|” in them.

Last, but not least, it may have gone unnoticed that the syntax allows for selector-less rulesets. When a ruleset has no selector, it matches the same as the preceding ruleset (this allows reusing a selector to define properties for different subjects, see below). If the first ruleset has no explicit selector, it matches the root node.

## 2.3. Additional syntax tweaks

In order to properly handle the RDF model and avoid leaving any case out, CRDF defines some pseudo-properties. They are just like normal properties, but they use the “@” symbol instead of a namespace prefix to denote that they are not real properties (and the “@” is followed by the “|” to avoid any possible conflict with at-rules). Currently, these pseudo-properties are defined:

<sup>4</sup> Would a space-separated list work properly? If so, the commas are overhead and should be avoided.

<sup>5</sup> Should all values be allowed with reversed? For example, does it make sense to give a literal with it?

- **@|subject**: defines the subject for which the current block defines properties. Must appear before any property (otherwise it is an error and is ignored). By default, this pseudo-property is inherited as by CSS cascade and inheritance rules. The initial value is the URI to the document. This pseudo-property accepts:<sup>6</sup>
  - **url (URI)**: This functional notation defines an explicit URI as the subject for the properties defined in the current block.
  - **none**: This keyword will cause a blank node to be generated only for the properties defined on the current block, for each matched element (if the selector matches multiple elements, each will have its own blank node). This blank node will never be used by any other CRDF block.
  - **blank** or **blank ()**: This notation creates an anonymous blank node for each matched element. The blank node is bound to that element, and other selectors that match it and use this keyword for their subject will be referring the same blank node. Also, selectors that match the element's descendant and inherit the subject will inherit that blank node as the subject, rather than creating a new node.
  - **blank (name)**: This functional notation creates a named blank node. The name must be a valid CSS identifier, or a **concat ()** expression that yields a valid identifier, as described under 2.1; and it can be reused from other blocks to refer to the same node, but it has no impact outside of CRDF (for example, no form of RDF output should have any reference to that name). Note that if the block's selector matches multiple elements, they will all share the name, and hence their subject will be the same blank node. See the parameter-less version of **blank ()** or the **none** keyword above to handle cases where this is not desired. Combining this with the **concat ()** and **attr ()** notations may also help creating independent nodes for each matched element that may still be re-used by other rule-sets.
  - **blank ("selector")**: This notation is based on [RDF-EASE's nearest-ancestor](#). It creates a blank node bound to the nearest ancestor element to the currently selected element that matches the given selector, taking any element as its own ancestor. Blank nodes created with this and with the parameter-less **blank ()** notation are shared if they are bound to the same element. If no element can be matched (but the **@|subject** declaration is syntactically valid), an independent blank node will be created as if the "none" keyword had been used.
- **@|typeof**: defines the type of the subject. Must appear before any property (otherwise it's an error and is ignored). It accepts a **url ()** expression, a quoted URI, or a **prefix|type** expression.

Property values' types are normally defined by the context (either by the value, by the vocabulary, or both); but a syntax is available for explicit typing if required, in the form **prefix|type (value)**.

## 2.4. Parsing tag-soup HTML as XML literals

RDF allows for values to be XML literals. However, classic or tag-soup HTML can't easily include XML content to be used as such literals. This section defines some rules for generic mark-up languages (which can also apply to HTML) to convert arbitrary mark-up fragments into its closest XML equivalent.

The source mark-up must be reviewed to ensure it is well-formed: it is assumed that the whole source from which the fragment is taken was well-formed, or it has already been corrected by the relevant

---

<sup>6</sup> The mechanisms to handle blank nodes probably need some more work. My knowledge of RDF, however, is limited, and I haven't yet understood how these nodes are defined and used. Examples and comments about this would be really helpful to complete this part of the document.

error-handling algorithm. Since XML literals can only be retrieved from the content of nodes in a document, and such nodes are the subtrees of a well-formed document tree, it is safe to assume that their contents are also well-formed.

Depending on the language being used, nodes should be classified as “elements” and “attributes”: for languages that have this distinction (such as HTML) or something equivalent (the SGML family, among others), the distinction is kept. For languages that do not differentiate kinds of nodes, all should be taken as elements. More complex languages that have more than two kinds of nodes should separate them into these two groups if they are going to allow XML literals to be extracted from them.

*Note: this will be more elaborate in further versions: what's supposed to happen with "text" or "comment" nodes? While the answer might be intuitive enough, these details should be clear and explicit before this document is turned into something more spec-like.*

Once the nodes on the document tree have been defined as either elements or attributes, the tree can be easily serialized to XML.

In the more specific case of contents of HTML5 elements being used as an XML literal, this entire process is reduced to just re-serializing the relevant parsed mark-up to HTML5's XML representation.

### 3. The host language

The host language refers to the language in which a document is written and bound to CRDF rules, such as X/HTML. This section defines the requirements for host languages from three perspectives: “classic” (a.k.a. tag-soup) HTML, XHTML, and generic XML.

There are basically three requirements for languages: a mechanism to link to external CRDF sheets, a mechanism to embed CRDF sheets (or fragments) within the document, and a mechanism to include inline CRDF rules.

In addition, there are some rules to deal with *pastage* (aggregate composition based on copy-pasting from one or more mark-up sources) and prevent clashes between the different sources.

#### 3.1. Linking to CRDF sheets

The host language **must** include a mechanism for linking to external CRDF sheets. For the X/HTML language family, the <link> element looks like the best match. The ideal code for linking a CRDF sheet would look like:

```
<link rel="metadata" type="text/crdf" href="myFile.crdf">
```

(Add a slash before the closing “>” for XHTML).

For generic XML, the sanest option is to use a XML processing instruction, like this:

```
<?crdf type="text/crdf" href="myFile.crdf"?>
```

#### 3.2. Embedding CRDF sheets

While this is not a hard requirement (ie: it is not strictly needed for any other purpose than convenience) it is highly desirable, and X/HTML languages already have an element ideal for this task: <script>. Using the appropriate type attribute, embedding CRDF sheets should be quite simple and should just be ignored by non CRDF-aware agents.

For XML languages, this would be a per-dialect task: dialects that have something as flexible as <script> could re-use it for CRDF; while other dialects may simply define an element for this very purpose or just omit this feature.

#### 3.3. Embedding inline CRDF

While this could be worked around by using lots of ID's and classes (or similar concepts for other languages), it is still a requirement, since having the metadata as close as possible to the relevant values is often of critical importance. Both for X/HTML and any XML dialect, the language **must** define an attribute whose content model is “CRDF inline content” (other wordings are acceptable, of

course, as long they mean the same). This is quite similar to the style attribute for CSS supported by X/HTML, but there are some exceptions. Although this document doesn't enforce any name for the attribute, the examples in further sections will assume an attribute with the name "crdf", just for the purpose of the examples. The syntax for inline CRDF is:

```
inlineCRDF:      CRDFsheet (as defined in the syntax section above)
                 declarations (as defined in the syntax section above)
```

Unlike CSS's style, which just takes what would be the contents of a rule-set, the attribute for inline CRDF can also take an entire sheet. Any at-rule within the inline CRDF will impact only the node including it and its children; and selectors of inlined blocks will be matched against the current subtree (the subtree that hangs from the node including the attribute), rather than the whole document tree. This means that selector-less blocks in inline CRDF will normally match the element providing them, unless preceded by blocks that do have selectors.

Parsers need to distinguish between full-sheets and simple declaration lists; so here is provided a simple algorithm that can do this job (note that comments should be ignored within this process):

- 1 Check the first non-whitespace character on the attribute value:
  - 1.1 If the first character is a "@", it may be an at-rule or a pseudo-property. Check the second character:
    - 1.1.1 If the second character is the namespace prefix separator "|", the attribute starts with a pseudo-property, so it is a sequence of inline declarations.
    - 1.1.2 Otherwise, the attribute starts with an at-rule, so it is a full CRDF sheet.
  - 1.2 Otherwise, search for a *non-literal* "{" (*non-literal* here means to ignore any escaped character and anything within quoted strings):
    - 1.2.1 If a "{" is found, the brace is opening a rule-set block: the attribute is a full CRDF sheet.
    - 1.2.2 Otherwise, there are no at-rules nor declaration blocks, so the attribute should be taken as a sequence of inline declarations.

**Note:** CRDF needs a clear mechanism to differentiate inline declarations from full sheets on inline attributes. This seems to be the simplest process that yields a correct result (at least for valid input). Any idea to make this process even simpler; or feedback on cases that the process would yield an incorrect result, would be really appreciated.

When computing the specificity of an inlined selector, the current depth on the document tree is prepended to the normal result.

## 4. Examples

These explicit rules might be complete and unambiguous; but let me be honest: they don't show off the idea as well as a working example would. So, here are some of the examples showing off how this proposal would deal with different use cases:

### 4.1. Reusing a value for multiple subjects (with no extra mark-up)

**Note:** the examples on this subsection had become obsolete with the changes under 0.1. They will be re-done once that section becomes more stable.

### 4.2. The CCREL case

Here is a sample of code generated by the Creative Commons website (using RDFa), and the equivalent CRDF version, which would also be suitable for the user to copy and paste on her/his page:

**Note:** due to the significant changes made to the inline content model, a version of this example for the old and new models are provided.

(Original) RDFa in XHTML mark-up :

```
<a rel="license" href="http://creativecommons.org/licenses/by-sa/3.0/us/">

</a>
<br />
<span xmlns:dc="http://purl.org/dc/elements/1.1/" property="dc:title">
  CRDF for CC Example
</span>
by
<a xmlns:cc="http://creativecommons.org/ns#"
  href="http://example.com"
  property="cc:attributionName"
  rel="cc:attributionURL">Author's name</a> is
licensed under a
<a rel="license" href="http://creativecommons.org/licenses/by-sa/3.0/us/">
  Creative Commons Attribution-Share Alike 3.0 United States License
</a>.
CRDF (old version) in HTML equivalent to the above:
<script type="crdf">
  @namespace dc "http://purl.org/dc/elements/1.1/"
  @namespace cc "http://creativecommons.org/ns#"
</script>
<a rel="license" href="http://creativecommons.org/licenses/by-sa/3.0/us/">

</a>
<br>
<span crdf="dc|title">CRDF for CC Example</span> by
<a href="http://example.com"
  crdf="cc|attributionName; cc|attributionURL:attr(href)">Author's name</a>
is licensed under a
<a rel="license" href="http://creativecommons.org/licenses/by-sa/3.0/us/">
  Creative Commons Attribution-Share Alike 3.0 United States License
</a>.
CRDF (new version) in HTML equivalent to the above:
<a rel="license" href="http://creativecommons.org/licenses/by-sa/3.0/us/">

</a>
<br>
<span crdf='@namespace dc "http://purl.org/dc/elements/1.1/" {dc|title}'>CRDF for
CC Example</span> by
<a href="http://example.com"
  crdf='
    @namespace cc "http://creativecommons.org/ns#"
    {
      cc|attributionName;
      cc|attributionURL:attr(href)
    }'>Author's name</a>
is licensed under a
<a rel="license" href="http://creativecommons.org/licenses/by-sa/3.0/us/">
  Creative Commons Attribution-Share Alike 3.0 United States License
</a>.

```

Note that it is no longer required to enclose the code in <div> or <p> elements: prefix clashes just can't happen. If the code were actually enclosed (which would be quite reasonable for this example), it could be suitable to move the @namespace rules to the container element, leaving only the properties in the inner tags, to improve readability.

### 4.3. The iguana collection's index

This is a sample of how the index page could be marked up for an iguana or any other arbitrary collection. The main point of this example is to show how this can be achieved with a fixed amount of CRDF code, regardless of how large the collection might be.

```
<html>
  <head>
    <title>Sample Iguana Collection - Summary</title>
    <script type="text/crdf">
      @namespace iguanas "http://example.com/RDF/iguanas#"
      tr:not(.headings) {
        /* Having a :matches() pseudo-class would make this even
         * simpler, and the class used in the document could be
         * entirely dropped, using a selector like:
         * tr:not(:matches( th)) { ... }
         */
        @|subject: none;
        @|typeof: iguanas|iguana
      }
      tr a {
        iguanas|name;
        iguanas|page: attr(href)
      }
      tr img { iguanas|pict: attr(src) }
    </script>
  </head>
  <body>
    <h1>Collection Summary</h1>
    <table>
      <tr class="headings">
        <th title="Click on the name to visit an iguana's
          main page.">Name & main page.</th>
        <th>Picture</th>
      </tr>
      <tr>
        <td><a href="lolo.html">Lolo</a></td>
        <td></td>
      </tr>
      <tr>
        <td><a href="dino.html">Dino</a></td>
        <td></td>
      </tr>
      <tr>
        <td><a href="ygg.html">Yggdrassil</a></td>
        <td></td>
      </tr>
      <!-- And so on, with any number of entries. -->
    </table>
  </body>
</html>
```

### 4.4. An iguana's main page

On a page that describes a single iguana, the scenario becomes quite different from the previous example: here we aren't restating the same properties over again for each specimen, but we will be stating many *different properties* for a *single subject*. Due to this, the embedding approach becomes more appropriate. This example demonstrates how the embedding is perfectly doable, and may require even less "boilerplate" code than other methods, such as RDFa:

```
<html>
  <head>
```

```

        <title>Dino's main page - Sample Iguana Collection</title>
</head>
<body crdf='@namespace ig "http://example.com/RDF/iguanas#"
        { @|typeof: ig|iguana; ig|page: url() }'>
    <h1>Dino's main page</title>
    <p><span crdf="ig|name">Dino</span> is a cute and playfull <span
    crdf="ig|gender">male</span> <span title="Hoplocercus spinosus"
    crdf="ig|species; ig|sciName:attr(title)">club-tail iguana</span>
    that enjoys eating <span crdf="ig|diet">flies and other
    insects</span>. He was born in <span crdf="ig|PoB">Cuenca,
    Spain</span>, on <time crdf="ig|DoB: attr(datetime)"
    datetime="2007-05-03">May 3rd, 2007</time>.</p>
    <!-- And so on... -->
</body>
</html>

```

Note that the species' scientific name (denoted by the `ig|sciName` property) is given as a title attribute on a span element. It could have given directly in CRDF code to hide it from human readers, but I considered the “tooltip” effect from this attribute appropriate for the situation.

#### 4.5. Mapping a Microformat to a RDF vocabulary

This is a very simple CRDF sheet that maps the classes used by the Geo Microformat to an arbitrary (supposedly equivalent) RDF vocabulary. The intent of this example is to show how CRDF would allow this kind of mapping for any Microformat, using just the class selector and descendant combinator. Such mappings allow the metadata being reused as either Microformats metadata or RDF triples, depending on the kind of agent used to retrieve it.

Keep in mind that this is supposed to be an independent CRDF file, expected to be `<link>`ed from documents that require it.

```

@namespace geo "http://www.w3.org/2003/01/geo/wgs84_pos#"
.geo {
    @|subject: none; /* generates a blank node for the root */
    @|typeof: geo|point /* and gives it the "point" type */
}
.geo .latitude { geo|latitude }
.geo .longitude { geo|longitude }

```

Currently, there is still an issue with this kind of mappings: it would be very tricky (if possible at all), to select *only* the first match for a class, for singular properties. On the other hand, despite Microformats.org defines explicit parsing rules for these cases, it is inherently wrong to have multiple instances of such values (for example, it makes no sense stating two or more latitudes for a single location).

Work is being done to propose a minimal extension to Selectors to handle these cases. The current idea is a pseudo-class with functional notation, in the form

`:singular(selector)`

that only matches the node if it is within a subtree matched by `selector` and is the first match within such subtree. With this, the above CRDF sheet could be converted to this:

```

@namespace geo "http://www.w3.org/2003/01/geo/wgs84_pos#"
.geo {
    @|subject: none; /* generates a blank node for the root */
    @|typeof: geo|point /* and gives it the "point" type */
}
.latitude:singular(.geo) { geo|latitude }
.longitude:singular(.geo) { geo|longitude }

```

And it would be a 100% compatible implementation of the GEO microformat (producing equivalent output even for bad input).

## 5. Inline vs. Linked metadata

CRDF allows defining inline metadata, in a way similar to that of RDFa or HTML5's Microdata, and also to define the semantics based on the structure of a document (through selectors). Both kinds of representation are supported because both have their use cases, and this proposal doesn't advocate for one of them over the other.

As a general rule (which might have some exceptions), when semantics are conveyed to the (human) reader through structure, metadata is more efficiently expressed through selectors that bind it to such structure; but when the semantics are conveyed through prose, inline metadata tends to work better. Reviewing the iguana examples above, on the "list" page the meaning of each entry is described by the visual layout of a table: seeing an image besides each name normally makes the reader understand that the image represents the same (or a closely related) thing as the name; and having the name linking to the relevant page conveys the idea that such page is about the thing denoted by the name. The listing of multiple items in the same way reinforces this perception; and the CRDF sheet makes use of this: it explicitly tells the machine that each row represents an item; and what property of such item each element in the row (the image, the text, and the link's URL) describes.

On the other hand, for the page about a particular iguana, the structure isn't as helpful: there is nothing on the structure that makes the user know that "Cuenca, Spain" is the birthplace of the iguana, or that "flies and other insects" are his diet (they could be his friends, for example). However, reading the text around makes things clearer: "He was born in Cuenca, Spain" leaves no doubt about what relates the iguana to this region, and the sentence "enjoys eating flies and other insects" makes clear enough that the iguana eats them, rather than establishing a lasting friendship. These cases are clear examples of semantics conveyed by the prose of the document, and hence are best described using inline metadata.

## 6. Conformance requirements

### 6.1. Document-level conformance

- A stand-alone CRDF sheet is conformant if it adheres to the syntax described above, doesn't include any non-supported at-rule, and all the properties defined in declarations map to actual properties of the vocabularies represented by the provided namespace declarations.
- An embedded CRDF sheet (like those included with `<script>` on X/HTML) is conformant if, after adding any non-overridden namespace declaration found before it (on previously linked or embedded sheets) would become a conformant stand-alone CRDF sheet.
- An inline CRDF sheet (only full sheet's, not simple declarations) is conformant if, after adding any applicable namespace declaration, it would become a conformant stand-alone CRDF sheet.  
*Note: "applicable" namespace declarations are those defined on linked and embedded sheets, and on inline sheets from the parent elements in the document tree, that are not overridden by later declarations.*  
*Note: this transformation would make the sheet to be conformant as a stand-alone sheet; but using it as such would have different meanings (especially on what is matched by the selectors).*
- An inline declaration set is conformant if, after enclosing it within "{" braces "}" and adding any applicable namespace declaration, it would become a conformant stand-alone CRDF sheet.  
*Note: see the notes above for inline sheets.*
- A document that includes CRDF metadata is conformant if all the elements and attributes providing such metadata are conformant per the rules above.

### 6.2. Tool-level conformance

*Note: this section is far from complete, and it's currently included only to make clear what should be expected from UAs that choose to implement CRDF.*

Conformant tools are required to process linked, embedded, and inline CRDF data as described by this document from any document in a host language they support, and be able to produce on request (from scripts, plug-ins, and/or human users) a list of all the RDF triples defined by the application of the CRDF code to the relevant document.

Specifically for web browsers, they are *not* required to keep a list of the triples and update it whenever a dynamic change happens to the document (ie: through javascript); but only to be able to apply the CRDF to the document on its current state when requested. Of course, maintaining a list of triples and just returning it on request would be a valid way to fulfill this requirement; but browsers can use other means as long as the result is the same.

Tools *may* provide additional functionalities for specific vocabularies (similarly to how IE8 provides some functionality for the WebSlice microformat, for example). They may also support other RDF serialization formats, and allow a combined list of triples to be retrieved from a document that uses multiple formats; but *should not* combine the information from different serializations by default.

**Note:** *It is unclear, for example, what should be produced if the RDFa and CRDF data on a document provide contradictory information. This can't be properly defined, because it would need to separately case each RDF serialization format, and new formats could appear at any moment. The only reasonable approach is to let the tool decide, based on the formats it supports; or even to put the choice on the hands of the user.*