



# <XML/> Without the X The Return of {{Textual}} Markup

Dave Beckett, Yahoo! Inc, 701 1st Avenue, Sunnyvale, CA 94089, USA; [dajobe@yahoo-inc.com](mailto:dajobe@yahoo-inc.com)

## Introduction

The web today has many textual protocols and formats beyond XML used for data, markup, querying and schema languages. This paper discusses textual formats as a trend using a case study of the Turtle RDF Syntax developed by the author and using developing a new JSON textual format for RDF as an example of the tradeoffs that need to be considered.

## Early Web Technologies

The web took off in the early days, primarily because people could “View Source” on web pages and compose new web pages in their favourite *text* editor. Note the key word there: the *format* of the web was *text*. There were plenty of pre-web attempts to let people create content but they were either clumsy or complex in their formats such as SGML, or were closed systems that required using specific tools.

The *protocol* of the web, HTTP was based on a long heritage of the IETF in creating textual applications protocols such as SMTP, NNTP. Despite the long-claimed inefficiency of textual protocols, all the competitors that used binary formats at the application layer, are dead, with few exceptions. Implementing the web protocol was easy for programmers familiar with processing text, and it was not unusual to build a new web service in an afternoon using existing tools, especially open source ones.

Development had proved pretty impossible with alternative formats and protocols such as ASN.1, EDI or 25, X.series and coloured book. These failed to deliver a world-wide information system for a variety of reasons including:

- Binary formats/protocols and not human readable.
- Required complex encoding, decoding and processing.
- The standards were proprietary or expensive to buy.
- Had little existing / open source software that could be used with them.

Which were all significant barriers for adoption. The early web took off because the technologies involved were easy for people to get involved with, for at least a certain set of technically experienced people. HTML was good enough and was simple enough to teach people, simple enough to be explainable in a consumer book.

## Web Markup

Markup is a method of annotating (primarily) text content, to add additional information about the annotated text. Text is meant here to be a Unicode string. The annotations in markup languages used on the web are primarily *inline*; that is to say, markers are present in the text around the annotated region, to delimit where the annotation applies. The alternative, *external* markup is much rarer than inline. There are two important types of markup on the web:

### Presentational Markup

The annotation describes how the text should be perceived with some human sense such as visually (viewed, read on a display, read on a print out), or aurally (heard). In (X)HTML these are inline markup types are elements such as `<b>`, `<font>`, `<i>` and `<u>` that correspond to physical printing terms.

### **Descriptive or Semantic Markup**

The annotation describes what the text means in some fashion, such as the author's intent, the importance of the text or a more formal description. In (X)HTML these are elements such as `<blockquote>`, `<em>` and `<strong>` which for example say, there is emphasis, but do not tell you how the emphasis is presented.

Semantic markup also covers the use of markup for encoding data that is a primary use case for XML and many other formats and is discussed in the following sections.

## **XML**

XML is the main web data language and also the main markup language if you pretend XHTML is more widespread than HTML tag soup. At least the angle-bracket tag XML/HTML *style* of markup is the most popular. It is based on an earlier and much larger, more complex markup language called SGML which itself is based on other markup languages going back for decades.

XML markup uses two types of *tags* to annotate the content: *elements*, which must be used before and after the content and *attributes*, which are used along with elements to add extra information. The element names are written in angle-brackets like `<name>...</name>` and the attributes inside the start element as pairs of `key="value"` with no duplicate keys like:

```
<name key="value">...</name>
```

It has two built-in notions of validation, the required well-formedness check where the element tags must be balanced, and the optional DTD validation that can constrain both the elements as well as the content in the elements.

XML has many other supporting technologies such as a way to arrange sets of element names into a *namespace*, to link to parts of a document, APIs to manipulate it, transformation technologies, advanced schema languages, databases to store it and entire query and programming languages designed for XML alone.

XML is good for describing **documents** (text mixed with markup, likely validating just the markup if at all) as well as **data** (validating the structure, types and syntax of content) – these were the two uses that drove its formation.

At this point in time, XML is a very mature technology with widespread infrastructure support and deployment. It is well understood and easy to use for developers, and especially, for web developers. As with anything that has a long standards process, it is not without criticism or things that could have been done differently. This paper is not focused at listing all the XML criticisms, but the main points relevant to this paper on textual markup are as follows:

### **Hard for humans to read**

The angle brackets around elements make it hard to see the encoded information.

### **Hard for humans to write**

Balancing start/end elements is error-prone and hard to scan quickly.

### **Ugly**

A highly subjective criticism. Some wish the angle brackets were round.

### **Namespaces**

These are controversial as an extension mechanism to some.

### **No built-in datatypes, validation or support for URIs**

As a data format, it is pretty weak offering a tree of untyped string nodes that cannot have hyperlinks to external objects or intrinsically record anything that is not an ordered hierarchy.

### **SGML Legacy**

DTDs, Entities and Processing Instructions probably should be thrown away.

For many of XML's main uses, it is totally appropriate due to the widespread tool support and it's mature features of markup, Unicode and validation. In many situations, where documents or data needs transferring or encoding, not choosing XML is a mistake.

## Web Textual Formats

The primary goals for textual formats are very different from XML-based solutions and the main two goals by far are:

1. Human readable and writable
2. Simple

But there are a variety of uses for textual formats beyond just data and documents. The main types of uses today are:

### Application protocols

Examples: HTTP and SMTP. The most successful application protocols by far are the textual ones HTTP for the web and SMTP for email. Both were developed based on IETF principles of being liberal on input and conservative on output with clear resulting ease of use and widespread interoperability. Protocols have been said to need binary or compact forms for efficiency but above the TCP/IP layer of Internet Protocol technologies, textual has won out as an application protocol and when bandwidth or size is an issue, a negotiated application-level compression has proved sufficient.

### Generic data formats

Examples: JSON [CRO06], YAML[YAML], Serialized PHP. A data object serialization format can turn an internal data structure into something “on the wire” with likely a small size and with a short lifetime. The data structure has a core fixed set of data types that are supported and a few structural types such as sequences, records and associative arrays. XML alone has no built-in datatypes apart from strings and only provides sequences and records. XML schema does add datatypes to XML but the resulting schema language that uses it: W3C XML Schema (WXS) is too complex for the average developer to understand and really only suitable for machines to generate and consume.

### Application-specific formats

Examples: iCalendar, vCard. Formats such as these are for specific uses such as iCalendar for describing events and vCard for digital business cards. The two are widely used but are rather old designs, and are often used with many vendor-specific extensions. They typically need their own entirely separate tool chains to use and are not intrinsically extensible.

### Markup formats for writing HTML

Examples: Wiki, Markdown, Textile. The main two styles are the wiki-formats that come in many varieties and the ones based on blogging that are intended to allow quick writing of blog entries or comments without any angle brackets such as Markdown and Textile.

### Data encoded in HTML

Examples: Microformats. Specially written programs or mappings can read “visible metadata” written in HTML with microformats and interpret it into some application-specific model or other format. The mapping may be defined explicitly via some declaration such as a GRDDL mapping. The textual part is that there are no additional angle-brackets for the data beyond those that already there, the meaning of existing HTML tags are overloaded to indicate regions with certain semantics.

### Query languages

Examples: SQL, XQuery, SPARQL. Database query languages such as SQL, XML query language XQuery and RDF query language SPARQL all are in text since they are intended to be written by humans. XQuery does have an XML version XQueryX but it is specifically intended for machines to manipulate.

### Schema languages

Examples: RelaxNG Compact. WXS is a verbose and hard to read schema language, something that has the feel of a machine-generated description. RelaxNG is both a better schema language and along with its textual equivalent RelaxNG Compact, easy for people to read and write and with the pluses that it can generate both WXS and DTDs (with some caveats).

### **Programming and scripting languages**

Examples: Perl, PHP, Python, Java, C++, ... These languages are read and written and have to be general purpose, terse as well as precise – if well designed – have some combination of the features above with typically more punctuation symbols *sigils* than data or markup formats tend to have. SQL and XQuery can also be considered programming languages as well as query languages although their design-point is to make querying easiest to use than general programming.

### **Mixed data and markup formats**

Examples: DBPedia, Semantic Wikipedia. DBPedia maps wiki markup data into multiple formats (Turtle/N3, RDF/XML, HTML) and semantic Wikipedia allows creating explicit data sections as well as HTML via extended wiki markup. These are primarily textual based but using the same data model across the XML and the textual formats, possibly more common with semweb applications that have a strong data-model centre.

## **Trends in Textual Formats**

### **Supporting Web Applications**

The most obvious recent trend in use of the web is in *Web Applications* where the user application has migrated from running on the desktop to run inside the web browser. This means that from a user's perspective it runs like an interactive application and it is expected to respond quickly to interactions.

The novelty extends to the implementation in that the browser client itself is now an integral part of what was “the web site”, and it takes on more than just a rendering role, but also includes substantial application logic and is more part of a client-server system with the backend web site, which instead of operating like an HTML-generating system, is delivering live data in a request/response with the browser as client.

This trend is typically described by the DHTML, Web2.0 or AJAX monikers which are about the underlying technologies in use – HTML, Javascript and XML operating asynchronously and invisibly between the browser and web site

However it has been found that the performance of these client-server operations needs to be very efficient so that the end-user interactions have a good response, to operate like an interactive application. This has meant that XML has sometimes been found to be too large and also harder to parse than really necessary in order to just communicate some objects between client and server.

Instead of XML, JSON – which by design is also a legal subset of JavaScript – is more often used and can be natively parsed in JavaScript (subject to some security checks). Serialized PHP is also generated for similar reasons.

### **Open Data for HTTP Web Services**

Data on the web can be easy to use as a (lowercase) web service, one where you can HTTP GET the information you want, rather than have to go via some complex *WS-Deathstar* negotiation over SOAP, that you cannot cache. Textual formats have been substantially used with the GET -web services, to supplement returning XML, which pretty much of all of them do also.

The textual formats are both for supporting web applications as described in the previous section but also generally seen as useful formats to return, compact and easy to read (by humans). It is common for web services to generate XML and at least one or two other textual formats such as JSON, or in the semwebby applications, Turtle and N-Triples. The Flickr API[FLICKR], Google or Yahoo! Maps and Search APIs are examples of common web application services that generate multiple textual results formats as well as XML ones.

These formats enable fast development of the style of data integration application called “mashups” where data from multiple web services is joined or integrated. It also suits other innovative applications that visualise and allow manipulation of data held on other web sites.

The low barrier to entry to allow people to use and re-use web services is one of the main reasons that textual formats help here.

## Summary of trends

- Small sized, short-lived data.
- Exchanging mainly application data, not markup.
- Goal: Make data human readable.
- Goal: Enable fast development.

## When Textual Formats go bad

Textual formats are not a panacea and they do come with their own downsides that have to be contrasted with the plus points raised above.

### Unicode support

Most textual formats, especially programming languages, started off with very little or very poor Unicode support and had to add it later, possibly as an option. XML had it from the start and it was always required, so consequently can be relied upon. This particularly affects longer-lived formats such as programming languages where the default encoding was set to 7-bit, ASCII or US-ASCII at some point in the past and now people want to use Unicode in the language while remaining compatible. This tends to be solved with default encodings that can be overridden which can cause more problems.

### Extending and Retrofitting

Especially with features such Unicode, as described above, textual formats have problems with extending the format. The need to extend formats occasionally has tended to make some of the formats (YAML, wiki) and programming languages (Perl) grow a larger and larger set of syntax detail possibilities along with use of a lot of special symbols (Perl again). Wiki formats also tend to evolve to supporting more and more of HTML by adding new ways to encode things over time.

The X in XML stands for eXtensible so it was a major design point to allow adding attributes or elements. XML remains the same at the core (elements and attributes) but you extend by inventing new names and/or using namespaces.

Just allowing adding more keys/fields is not the end all of extensibility. There is often little thought put in the textual formats to adding more datatypes, new types of structures or even putting in the format a mechanism to allow new things to be added later. Some languages such do have this such as Python's `from future import ...` which has successfully been used to add language features.

### Proliferation of similar alternatives

There are dozens, if not hundreds, of similar Wiki markup languages and blog markup languages such as Markdown and Textile, all with little differences. This might be considered an indication for the need for standardisation in the area, but that goes counter to the ethos of these light-weight and ever evolving systems, to stop and call a format complete.

### Used beyond their sweet spot

*"The sweet spot for JSON is serializing simple data structures for transfer between programming languages.*

*If you need more complex data structures (maybe with some kind of schema for validation), use XML"* Simon Willison, <http://simonwillison.net/2006/Dec/20/json/>

### Not thinking about big problems

Once you get beyond recording simple local data structures, and start to want to share and connect, you need to have ways to disambiguate the same term used in different places. This particularly affects mashups and aggregations. Textual formats tend to have no namespaces, modules or packages (except for the programming languages) so you will have to invent your own naming convention and hope either somebody else didn't use it (X-Foo headers or fields) or try to come up with something unique. Reusing class names in microformats is an

## <XML/> Without the X The Return of *Textual* Markup

example of just grabbing public names and hoping nobody else uses them. XML at least has namespace, if you want to use them.

### Validation

Love it or hate it, some people want it and textual formats rarely have even the concept of validation, nevermind tools support for this, unless it is intrinsic to the format (Relax NG compact) or you have to do a syntax check anyway (programming languages, query languages). If the problem is schema-driven, having no way to validate against it for the data in hand can be a problem, especially if the schema changes, custom code has to be updated if there is no programatic way to validate against the schema.

### Tools

Each textual format needs it's own tools for at the very least, reading and writing the format. Some of the rules for textual formats can be baroque to deal with due to lots of syntax detail and whitespace rules (YAML, Wiki, vCalendar, vCard, programming languages). XML as a generic syntax with it's existing tool support beats this easily. There are (for example) no JSON databases, query languages, out of the box APIs sitting in the standard SDKs for all major programming languages and operating systems. XML has all of that.

### Performance: Speed and Size

Despite all the issues with XML's background and oddities and verbosity, there are optimising, streaming XML parsers with 10+ years of development, deployed everywhere and this can often beat quick hacks with a textual format. The XML tools have also often been tested with large datasets (gigabytes), much bigger than are ever typically thrown through textual formats.

### Whitespace has semantics

What is this, the 1960s when where you punched a hole in a card column number mattered? Did nobody figure out when designing Python and YAML why this was a bad idea after the problems with Fortran and `make(1)`? XML does not make a virtually invisible part of the markup something that has a special meaning.

## Case Study 1: The Turtle RDF format

In [Bec04] I described multiple problems with RDF/XML as a syntax for writing RDF in XML mainly in having too many alternatives and being an early XML language. In it I also proposed a strawman replacement RXR (*Regular XML RDF*) as the simplest triple-encoding in XML. Since then I have considered the textual route for a syntax, intended to meet the needs of people – human readable and simple – the core purposes for text formats as given above.

RDF/XML is sufficient for machines to exchange RDF, with all the usual XML pro-s and con-s albeit as an early XML format, with additional issues of it's own. However, in parallel, semantic web developers needed a more “scribble-able” format for writing triples quickly in email and messaging, and since 2000 Berners-Lee developed *Notation3* (N3) [B-L98] as a handy format for writing down triples, and experimenting with beyond-RDF ideas.

During the RDF Core[RDFCORE] standardising work of 2002-2004, a way was need to consisely write down triples for test cases so a strict subset of N3 was developer called *N-Triples*[GRA04]. It was very strict in that it had only 1 way to write down any triple; think of it as the textual true opposite of RDF/XML. N-Triples looks like this (this is one triple on one line):

```
<http://www.dajobe.org/foaf.rdf#i> <http://xmlns.com/foaf/0.1/homepage> <http://purl.org/net/dajob
```

N-Triples was a success and is still used ongoing in test cases for OWL, SPARQL, GRDDL and other semantic web systems, as a definitive, if verbose, way to write down any RDF set of triples, or RDF graph. Indeed it can write down some RDF graphs that cannot be written in any XML syntax due to XML restrictions on the allowed Unicode characters at least in XML 1.0, although XML 1.1 removes some of these restrictions.

However, N-Triples was still too verbose with all the visible long URIs so in 2004 as part of the Semantic Web Advanced Development Europe (SWADE) project, I began to develop what I initially called *N-Triples Plus* that added some of the abbreviations that N3 provided with the following goals:

1. Add to the syntax only what was useful for people

## 2. Remain within the RDF data model

This new syntax was named *Turtle – the Terse RDF Triple Language*[BEC06] since it was focused on making writing RDF more compact. The set of abbreviations added addressed verbosity and duplication in the syntax plus some additions for convenience to make short forms for common or complex forms.

### Prefixes Names (Verbosity)

Making the URIs shorter, importing the XML-style QName or qualified-name for declaring a short prefix (`@prefix`) and then using it in the place of the long `<URI>` form. This solves 90% of the verbosity problem.

### Abbreviate repeated triple subjects (Verbosity, Duplication)

Allowing lists of triple (predicate, object) term pairs separated by ‘;’

### Abbreviate repeated triple subjects and predicates (Verbosity, Duplication)

Allowing lists of triple (object) terms separated by ‘,’s.

### Blank Nodes (Duplication)

Reduce duplication of blank nodes by allowing the Turtle parser to generate the names using a ‘[]’ form as a placeholder.

### RDF Collections (Verbosity, Convenience)

A way to format `rdf:parseType="Collection"` lists using a set of terms in ‘(..)’ without seeing all the infrastructural triples or detail.

### Common Data Types (Convenience)

Allow integers, doubles, decimals and booleans to be typed natively

### Common Predicate (Convenience)

Abbreviate the commonly used `rdf:type` predicate with ‘a’

Turtle itself has been a success in terms of wide use and implementation across major RDF systems. It has slowly added abbreviations from N3 until it has reached a fairly stable state by 2005/2006. Turtle in 2007 looks like this, encoding 3 triples taken from <http://www.dajobe.org/foaf.rdf> :

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<http://www.dajobe.org#i>
  a foaf:Person;
  foaf:name "Dave Beckett";
  foaf:homepage <http://purl.org/net/dajobe/>;
...
```

During the SPARQL development the triples language was also influenced by N3/Turtle design and the triple pattern format in the `WHERE` clause is today, very close to Turtle with minor technical differences.

Turtle has hit a sweet-spot of functionality, precision, ease of use that has been demonstrated by its uptake, even while it remains outside the RDF standards track, today. The slow evolution process to add syntax and review the feedback proved a success compared to a top-down, clean-sheet design of a new format.

## Case Study 2: Creating a new format – RDF-JSON

The goal here is to develop a JSON encoding for RDF triples, so that it is suitable for web-browser applications reading JSON and wanting to process triples about some subject.

As described above, JSON is an object serializing format, and it can easily record as set of triples as a set of (3 tuple) objects, each of which is a uri, literal or blank node types. This is relatively straightforward and you end up with something like this, using the style of the *Serializing SPARQL Query Results in JSON* [SPARQLJSON], as the syntax:

```
{
  "subject" : {
    "value" : "http://www.dajobe.org/foaf.rdf#i",
    "type" : "uri"
  }
}
```

## <XML/> Without the X The Return of *Textual* Markup

```
},
"predicate" : {
  "value" : "http://xmlns.com/foaf/0.1/homepage",
  "type" : "uri"
},
"object" : {
  "value" : "http://purl.org/net/dajobe/",
  "type" : "uri"
}
},
```

This **Triples-Centric Style** RDF-JSON has a similar verbosity problem as N-Triples and also it is hard to see all the information about a subject. Taking that as a hint, we can apply the same grouping of triples about the same subject, abbreviating long URIs, using something like QNames and allowing multiple objects using a JSON sequence to give something more compact and record-like:

```
{
  "prefixes" : {
    "rdf" : "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    "foaf" : "http://xmlns.com/foaf/0.1/",
    ...
  },

  "http://www.dajobe.org/foaf.rdf#i" : {
    "rdf:type" : [ { "value" : "http://xmlns.com/foaf/0.1/Person", "type" : "uri" } ] ,
    "foaf:homepage" : [ { "value" : "http://purl.org/net/dajobe/", "type" : "uri" } ] ,
    "foaf:name" : [ { "value" : "Dave Beckett", "type" : "literal" } ] ,
    ...
  },
}
```

This is a more **Resource-Centric Style** RDF-JSON and might be the best place to stop abbreviating. It forms a regular structure that can be navigated and looked-up by applications from the interesting items – the resource URIs – it is compact and complete. It has gained a lot more JSON punctuation sigils but it is necessary to add the sequence of values. Since JSON cannot be extended to natively understand URI datatypes or other RDF terms, it is required to reify the RDF terms such as a URI *u* to be a JSON object:

```
{ "value" : "u", "type" : "uri" }
```

There have been other forms of JSON seen that encoding RDF-like information, but not quite in the same style, such as with Exhibit tool[EXHIBIT] from the MIT SIMILE project for exchanging semwebby data between data-extracting and browser-based data visualising applications. The data model given in Exhibit Database splits the JSON content into items (with ID and label), types, properties and property values. These roughly correspond to triple subjects (or resources) with a `rdfs:label` triple, a `rdf:type` triple, triple predicates and triple objects. All predicates have the same value type.

## Conclusions

There have to be good reasons to make a textual format instead of an XML one.

Making a format available for people to easily read/write is a very good reason.

Just take care with going too far, as XML can provide a lot of functionality and in particular, has a lot mature support, that a new textual format will not have for years.

## Bibliography

[BEC03] D. Beckett, A retrospective on the development of the RDF/XML Revised Syntax, 4 June 2003, ILRT Research Report Number: 1017, University of Bristol, UK, [http://www.ilrt.bris.ac.uk/publications/researchreport/rr1017/report\\_html?ilrtyear=2003](http://www.ilrt.bris.ac.uk/publications/researchreport/rr1017/report_html?ilrtyear=2003)

*<XML/> Without the X The Return of `Textual` Markup*

- [BEC04] D. Beckett, Modernising Semantic Web Markup, 20 April 2004, XML Europe 2004, Amsterdam, [http://www.idealliance.org/papers/dx\\_xmle04/papers/03-08-03/03-08-03.html](http://www.idealliance.org/papers/dx_xmle04/papers/03-08-03/03-08-03.html)
- [BEC06] D. Beckett, Turtle Terse RDF Triple Language, 4 April 2006, <http://www.dajobe.org/2004/01/turtle/>
- [B-L98] T. Berners-Lee (ed.), Notation 3, 1998, Technical report, World Wide Web Consortium (W3C), <http://www.w3.org/DesignIssues/Notation3>
- [CRO06] D. Crockford, JSON, the fat-free alternative to XML, Dec 2006, Proceedings of XML 2006, <http://www.json.org/fatfree.html>
- [EXHIBIT] Exhibit software, SIMILE project, MIT, <http://simile.mit.edu/wiki/Exhibit>
- [FLICKR] Flickr Services API Documentation, <http://www.flickr.com/services/api/>
- [GRA04] J. Grant and D. Beckett (eds.), RDF Test Cases, February 2004, World Wide Web Consortium (W3C), W3C Recommendation, <http://www.w3.org/TR/rdf-testcases/>
- [RDFCORE] W3C RDF Core Working Group, February 2001, World Wide Web Consortium (W3C), <http://www.w3.org/2001/sw/RDFCore/>
- [SPARQLJSON] K. G. Clark, L. Feigenbaum, E. Torres (eds.), Serializing SPARQL Query Results in JSON, W3C Working Group Note, 4 October 2006, <http://www.w3.org/TR/rdf-sparql-json-res/>
- [YAML] O. Ben-Kiki, C. Evans, B. Ingerson (eds.), YAML Ain't Markup Language (YAML™) Version 1.1, Working Draft, 2 December 2004, <http://www.yaml.org/>