Structured Metadata in HTML

This post deals mostly with Google's implementation of JSON-LD structured data.

While the schema.org effor was undertaken in 2011/2012 by all major search engines at the time (Google, Bing, Yahoo, and Yandex) not all of them have adopted a common platform. Bing doesn't support JSON-LD, Yandex has its API behind email registration and I don't know if Yahoo has any resources to validate JSON-LD for their search engine (particularly post acquisition).

So, while in theory this should work in all search egines, only Google provides examples and freely available validation tools.

Also note that this post only covers how to use JSON-LD with Schema.org types to support search engine result discoverability. It does not discuss how to build APIs with JSON-LD.

One thing I've always found intriguing is how to markup data to make it appear like it does in Google's search results. There are several different structured markup formats. 3 of the most widely used are Microdata, RDFa Lite and JSON-LD. I've chosen to work with JSON-LD because Microdata is not supported accross browsers (Webkit removed the feature because "the feature never gained any traction and was eventually removed to clean up the codebase"). Support for the Microdata API was also removed from Blink (Google Chrome). Removal of the feature from a browser also shows us a likely future for Microdata, which is less and less support.

What Is Linked Data

Before jumping straight to JSON-LD (JSON for Linked Data) it would help to understand what Linked Data is and how it relates to the web and other resources.

In computing, `linked data` (often capitalized as Linked Data) is a method of publishing structured data so that it can be interlinked and become more useful

through semantic queries. It builds upon standard Web technologies such as HTTP, RDF and URIs, but rather than using them to serve web pages for human readers, it extends them to share information in a way that can be read automatically by computers.

Tim Berners-Lee, director of the World Wide Web Consortium (W3C), coined the term in a 2006 design note about the Semantic Web project.[1]

Tim Berners-Lee outlined four principles of linked data in his Linked Data note of 2006, paraphrased along the following lines:

1. Use URIs to identify things
2. Use HTTP URIs so that these things can be found and dereferenced
3. Provide useful information about the object a name identifies when it's looked up, using open standards such as RDF, SPARQL, etc
4. Refer to other things using their HTTP URI-based names when publishing data on the Web

What is JSON-LD?

JSON-LD is a lightweight Linked Data format. It is easy for humans to read and write. It is based on the already successful JSON format and provides a way to help JSON data interoperate at Web-scale.

Core Markup

In the following example we can see some of the basic attributes of a JSON-LD document. I'm working primarily with `context` and `@type`

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "Book",
```

```
"accessibilityAPI": "ARIA",
"accessibilityControl": [
  "fullKeyboardControl",
  "fullMouseControl"
],
"accessibilityFeature": [
  "largePrint/CSSEnabled",
  "highContrast/CSSEnabled",
  "resizeText/CSSEnabled",
  "displayTransformability",
  "longDescription",
  "alternativeText"
],
"accessibilityHazard": [
  "noFlashingHazard",
  "noMotionSimulationHazard",
  "noSoundHazard"
],
"aggregateRating": {
  "@type": "AggregateRating",
  "reviewCount": "0"
},
"bookFormat": "EBook/DAISY3",
"copyrightHolder": {
  "@type": "Organization",
  "name": "Holt, Rinehart and Winston"
},
"copyrightYear": "2007",
"description": "NIMAC-sourced textbook",
"genre": "Educational Materials",
"inLanguage": "en-US",
"isFamilyFriendly": "true",
"isbn": "9780030426599",
"name": "Holt Physical Science",
"numberOfPages": "598",
"publisher": {
  "@type": "Organization",
  "name": "Holt, Rinehart and Winston"
```

```
  }
}
</script>
```

Basic JSON-LD concepts

This is a basic (and incomplete) overview of JSON-LD as I've used it in schema.org-based search engine optimization markup. It is not complete nor it's meant to be. If you want more detailed information look at the latest syntax specification from the JSON for Linking Data Community Group.

@context

In JSON-LD, a context is used to match terms in the document to Internationalized Resource Identifiers (IRIs), internationalized URIs/URLs.

The Web uses IRIs for unambiguous identification, a given IRI will match one and only one resource. The idea is that these terms mean something that may be of use to other developers and that it is useful to give them an unambiguous identifier. They will also help prevent collisions if mor than one author decides to use the same name: the contexts will be different so they will not collide.

In this example we are using schema.org as the context. All future work will be bound to it.

```
{
  "@context": "http://schema"http://schema.org""@type": "Book"
}
```

@type

The type attribute tells us what type of resource it is in the given context. For example using the following code:

```
{
  "@context": "http://schema"http://schema.org""@type": "Book"
```

```
}
```

Tells whoever is reading the JSON file that this is defining a book using the context from schema.org.

@language

The `@language` attribute tells JSON-LD what language to use for the specified resource. There are multiple ways to use it. In the first example we set it up on the root cotnext as the default language (unless it's overwritten)

```
{
  "@context": "http://schema"http://schema.org""@type": "Book",
  "@language": "en"
}
```

You can also incorporate the attribute wherever you use the `@type` attribute, which will override the default for that particular resource only.

Square Brackets

Square brackets exist for situations where there are multiple values for an item property. In the example below, both `accessibilityControl` and `accessibilityFeatures` have multiple values, like saying there are multiple accessibility controls and features.

> This is different than formal nesting, discussed in the next section.

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "Book",
  "accessibilityAPI": "ARIA",
  "accessibilityControl": [
    "fullKeyboardControl",
    "fullMouseControl"
```

```
  ],
  "accessibilityFeature": [
    "largePrint/CSSEnabled",
    "highContrast/CSSEnabled",
    "resizeText/CSSEnabled",
    "displayTransformability",
    "longDescription",
    "alternativeText"
  ]
}
</script>
```

Nesting

Where square brackets give you the option to nest multiple values for the same attribute nested elements use curly brackets `{}` to insert a completely different but related element.

Or a more complex example of nested elements taken from Google's Fact Check Structured Data

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "http://schema.org""@type""datePublished": "2": ""datePublished": "http
  "itemReviewed":
  {
   "http://example.com/news/science/worldisflat.html""itemReviewed":
  {
    "@type"  "name": "Square Worl": ""name""author""sameAs": "htt": ""@typ
      "name": "Square World Society",
      "sameAs"lished": "2016-06-20"
  },
  "claimReviewed": "The world is flat",
  "author":
": ""datePublished": "2016-06-20"
  },
```

```
  "claimReviewed": "The world is flat",
  "author":
  {
    "@type"type": "Rating",
    "": ""name"e": "1",
    "bestRating": "5",
    "": ""reviewRating":
  {
    "@type"me" : "False"
  " : ""ratingValue": "1",
    "bestRating": "5",
    "worstRating": "1",
    "alternateName" : "False"
  }
}</script>
```

- **JSON-LD nesting checklist**
  - Must use the item property (specific to the item type)
  - The value lives in curly braces
  - You MUST identify the item type of that property. The JSON-LD parser must know what type of object you nested
  - Attribute/value properties for the type must be included
  - Follow proper JSON syntax

@id

The `@id` element gives a unique IRI/URI referencing the element it's used in. Where there is more than one reference for the same resource (different formats for the same video, or different size for the same image) don't use `@id`, just name the resource to follow the schema.org guidelines.

```
{
  "@context": "http://schema"http://schema.org""@type": "NewsArticle",
  "mainEntityOfPage"type": "": ""@type"   "@id": "https:": ""@id": "https
  }
}
```

Commonly used types

Schema.org provides vocabularies for some common types of markup. This is a subset of the material supported by Google and it's a good starting as it provides a comprehesive crosslist of all the material supported for a given type.

These type descriptions are not like Google's Search Gallery in that they don't provide full examples but a list of attributes that you can use for a given content type.

- Creative works:
    - CreativeWork
    - Book
    - Movie
    - MusicRecording
    - Recipe
    - TVSeries
- Embedded non-text objects:
    - AudioObject
    - ImageObject
    - VideoObject
- Event
    - Health and medical types: notes on the health and medical types under MedicalEntity
    - Organization
    - Person
    - Place
    - LocalBusiness
    - Restaurant
- Product
    - Offer
    - AggregateOffer
- Review
    - AggregateRating
- Action

Tagging Examples

Now that we know what Linked Data is, some basics about how to write JSON-LD

and where to find more information about it, let's look at some examples of how Google recommends you write JSON-LD to work with its search engine. We'll look at both structural and content markup examples.

> You can find more fully realized examples in the Google Search Gallery

Article

This examples shows the markup for a news article. Notice how it breaks author and publisher into different types and how it can extend those types to get as rich as we need to.

```
{
  "@context": "http://schema"http://schema.org""@type": "NewsArticle",
  "mainEntityOfPage"type": "": ""@type"   "@id": "https:": ""@id"com/artic
  },
  "headline": "Artic": ""headline"  "image": [
    "https://example.com/photos/1x1/photo.jpg",
    "https://example.com/photos/4x3/photo.jpg",
    "https://example.com/photos/16x9/photo.jpg"
  ],
  "datePublished": "2015"https://example.com/photos/1x1/photo.jpg""datePub
  "author": {
    ""author""dateModified""name": "John Doe"
  },
  "publis": ""author""@type": "Organ": "ion",
    "name": "Goo": ""name""logo": {
      "@ty"logo""publisher": {
    "@type" "https://google.com/logo.jpg"
    }
  },
  "https://google.com/logo.jpg""logo": {
      "@type": "ImageObject",
      "url": "https://google.com/logo.jpg"
    }
  },
  "description": "A most wonderful article"
}
```

Video Object

The video object describes a video that you may put up on your web site as marketing collateral or in a Youtube channel to try to monetize and grow your audience.

```
{
  "@context": "http://schema"http://schema.org""@type"ject",
  "name": "T": ""name"description": "Video descr": "on",
  "thumbnailUrl": "h": ""thumbnailUrl""thumbnailUrl".jpg",
  "uploadDate": "2015-02-05T08:00:00+0": ""uploadDate"on": "PT1M33S",
  "publisher": {
": ""duration": "PT1M33S",
  "publisher" "Example Publisher",
    "logo": {
 "Example Publisher""name"t",
      "url": "https://e": ""url""logo": {
      "@type": "ImageObject",
      "url"
    }
  },
  "contentUrl": "https://www": ""width": 600,
      "height"mbedUrl": "https:/": ""contentUrl": "https://www.example.com
  "embedUrl": "https://www.example.com/videoplayer.swf?video=123",
  "interactionCount": "2347"
}
```

Use the Force, Luke

I hate reinventing the wheel. This is where the Google Search Gallery comes in. They provide full markup examples of the most frequently used types.

The library also includes what they call enhancements likke breadcrumbs, search boxes and carousels. These may not be visible in the page that links to the JSON-LD but will allow new functionality in the SERP (Search Engine Results Page) where they appear.

Available examples from the gallery:

- Content types
    - Article
    - Book
    - Course
    - Dataset
    - Event
    - Fact Check
    - Job Posting
    - Local Business
    - Music
    - Occupation
    - Paywalled Content
    - Podcast
    - Product
    - Recipe
    - Review
    - TV and Movie
    - Video
- Enhancements
    - Breadcrumb
    - Corporate Contact
    - Carousel
    - Logo
    - Sitelinks Searchbox
    - Social Profile

Process to create JSON-LD data

So what does it take to create a JSON-LD document for a page? I've adapted this list from JSON-LD For Beginners. It has been slightly edited for style.

It's important to note that the first two steps are planning what you will markup and why do you want to do it. Experimenting is ok but marking up content just because you can't is not, or at least it shouldn't be.

1. Mentally answer:
    1. What do you want to mark up?
        1. **Goal:** Determine that you can mark up the content with the Schema.org vocabulary. Some things may make sense conceptually, but are not available within the vocabulary

    2. Why do you want to mark it up?
        1. **Goal:** Determine whether there is a business case for what you want to markup. If you're not doing it for business reasons or as an experiment you shouldn't mark content up just for the sake of marking it up
        2. You want to mark up content that will help search engines understand the most vital information on your page and maximize your ability to demonstrate that you are the best resource for users
        3. Look at Google's resources on structured data, how they use them, and examples of different types of supported elements and how they recommend using them
2. If you're using a markup that Google explicitly supports (i.e., JSON-LD, Microdata or RDFa), open the specific documentation page and any relevant examples
    1. Don't feel like you have to create your markup from scratch. You should understand JSON-LD and the Schema.org vocabulary
    2. If Google or search engine results provide examples of the type of markup you want to use then use it!
3. Open up the Schema.org item type page
    1. Especially when you're starting off with Schema.org, review the Schema.org technical documentation page to get an idea of the item type; what it entails, and its properties. This can help you better understand an example or make it easier to create your own content
4. Copy/paste the immutable elements
    1. Copying and pasting the required elements saves time and mental energy. If you're using an external example the immutable elements should already be present
    2. Occasionally an examples may leave out the `script` tags. Please note that they are vital; Browsers will not parse JavaScript outside `script` tags or properly linked scripts
5. Add desired item type you're marking up as the value of `@type`:
6. List item properties and values
    1. This step doesn't require syntax and is more of a mental organization exercise. Concentrate on what you want to markup — don't sweat the details yet.
    2. You may have ideas about what you want to mark up, but may not necessarily know if it's possible within the vocabulary or how it's nested
7. Add JSON-LD syntax, nesting where required/appropriate
    1. The step where you finish up your model, nest it, and put markup

   together
8. Test with the Structured Data Testing Tool
    1. The tool is your linter for grammar and syntax. Confirm that the elements and attributes you use are well formed and pass validation. When in doubt, check the Google documentation and the corresponding entry in schema.org
9. Determine strategy for adding to the webpage
    1. You must inline the JSON-LD data in a script tag in the head of the page
    2. If you're working with dynamic content or through a CMS, work with your dev team to add the script to the page

Review the Guidelines

Once you have the markup that you want to use and have validated it using Google's validation tool, make sure to check their Structured Data General Guidelines, particularly their quality guidelines. This will save you time and potential headaches since **Google does not guarantee that your structured data will show up in search results, even if your page is marked up correctly according to the Structured Data Testing Tool** and this mey be caused by a vaariety of reasons.

But if you've created good content and marked up your JSON-LD appropriately the search results should become a much better experience for the people using search engines.

Resources

- General Information and Tutorials
    - schema.org
    - Introduction to Structured Data
    - Google Structured Data Policies
    - Build, Test, and Release Your Structured Data
    - A Guide to JSON-LD for Beginners
- Structured Data Formats
    - JSON-LD
    - RDFa
    - Microdata
- Tools and Examples

- Google Search Gallery
- Structured Data Testing Tool