Accessibility Metadata Semantic Markup Primer

Version 1.0.1, December 29, 2013

Table of Contents

About this Guide

This guide is a companion to the Accessibility Metadata Best Practices Guide. It provides a brief tutorial on how to use microdata and RDFa attributes in your HTML pages to include schema.org accessibility metadata. It is not designed or intended to be either a comprehensive guide to either technology, or to using schema.org properties, simply to get you up and running.

Schema.org

If you're reading this primer, you already know that semantic metadata plays a key role in discovering the accessibility of online resources. What undoubtedly seems foreign is the form the markup takes and how you go about adding it. Before jumping straight into those kinds of technical details, it will help to first understand schema.org, both the site and the concept.

The schema.org web site was created by Google, Microsoft, Yahoo! and Yandex in 2011 in an effort to standardize semantic markup for search engine optimization. The site does not add new tags or attributes to HTML, but defines types and properties. These types and properties don't do anything on their own (you won't find any magic if you visit the site), but are used with two metadata technologies — RDFa and microdata — to enrich your web pages. This guide will be detailing how this is done.

The idea of schema.org is much bigger than the rather mundane pages of type and property names you'll find on it: being able to extract meaning from the content of the web through a common vocabulary. In the case of accessibility metadata, this means that search engines will be able to compile information about the nature of the content that otherwise wouldn't be easily surfaceable.

The Accessibility Metadata Project proposal contributed a set of properties to an existing type in schema.org, but to understand what that means we now need to review the structure of schema.org.

(For more information about the accessibility properties, please see the Best Practices Guide.)

Types

Although schema.org can be conceived of as one big vocabulary for the web, it's actually a hierarchical collection of what are called types, each of which defines a micro-vocabulary (or schema, hence the site's name). These types are a lot like a taxonomic classification system, with groupings based on shared characteristics.

Like a taxonomy, the structure of schema.org starts with two generic root types: DataType and Thing. DataType defines various kinds of data expected when you use properties (e.g., text, dates, URLs), but more important to this discussion is Thing. Below the Thing type lies all of schema.org's major vocabularies. (Note: if this structure seems murky, it's best to have a look at the Full Hierarchy page.)

When it comes to the accessibility metadata, be aware that the key type under Thing to be aware of is CreativeWork, as this is the top-level type the accessibility metadata properties have been contributed to.

Another interesting fact about schema.org is that properties are inherited downward by the specialization subtypes. By leveraging this inheritance, the accessibility metadata is available for use whenever you define a page as containing a CreativeWork or any of the more specific types you find branching out from it.

What this means is that the metadata will be available for annotating a vast array of the web's content resources: articles, blogs, books, media objects and on.

But types are just names on their own. They indicate what type of thing you've bumped into, but don't tell you anything more specific about that thing than its name, and that's where properties come in.

[Note: Don't confuse the HTML `class` attribute when you think schema.org types, as there is no relation between the two. Schema.org metadata is not used in `class` attributes, and doing so will strip the semantics of their meaning (i.e., they will not get extracted/indexed).]

Properties

If a type defines what you're talking about, properties are what you use to make specific statements about it. A type on its own isn't very interesting without any defining characteristics, after all.

In other words, simply stating that you are going to describe a Book (the type), without actually indicating the title, author, publisher, ISBN or publication date (the properties), isn't terribly helpful to a search engine trying to index your page. It will know what your page is defining, but searching into the content is still reduced to keyword matching.

One of the easiest ways to think of properties is as defining "is a" or "has a" relationships between the type and its content (e.g., the book has a title). If you stop here for a moment and look at the Book type property list, you'll see quickly from the names of the properties how this is generally true (although not perfectly).

The same is true for accessibility: a resource like a book has an access mode, its display is possibly transformable to user needs, it might have additional media features of note, etc.

Every property in turn must have a value associated with it, otherwise it says nothing. For example, if you say a book has an access mode without providing the access mode, you've really not helped the search engine understand your content.

Property values often are simple text strings, but can take more complex forms, like dates, language codes or uniform resource locators (URLs).

In the case of schema.org generally, the properties of the type you're defining are usually part of the readable text of the document. The metadata framework may only exist in the underlying markup, but the content being augmented isn't normally hidden from view.

This model doesn't work so well for accessibility metadata, however. Although accessibility information is integral to understanding the nature of the content, it is not always in the content itself (especially in the case of visual media that doesn't include text). It's for this reason that you'll find a lot of accessibility metadata expressed using non-visible HTML `meta` tags in the body of the document.

It is a best practice to incorporate as much of the nature of the content into the rendered document as you can, however, as that allows anyone accessing the page, regardless of searching, to discover the same information.

The understanding implicit in this guide is that accessibility metadata cannot always be surfaced in the visible content, but do not read into the use of `meta` tags a requirement for the information to be non-visible.

But that's as far as we can go without actually moving out of the theoretical realm of semantic markup. The remainder of this section will now turn to look at how this metadata gets expressed practically within documents.

Enriching Markup

Overview

This section provides a quick introduction to using microdata to tag content. RDFa can also be used to implement schema.org metadata, but for simplicity is not handled in this primer. A brief introduction to RDFa, and how to map between the two technologies, is provided in Appendix A. (Note that if you are looking to add accessibility metadata to HTML4 or XHTML 1.1, you will need to use RDFa, as microdata is not valid outside of HTML5.)

Microdata

There are three key attributes in microdata you will find yourself using over and over to include accessibility metadata: `itemscope`, `itemtype` and `itemprop`. These attributes allow you to define your type (`itemscope` and `itemtype`) as well as its defining characteristics (`itemprop`).

Defining Types

Microdata uses two attribute to define types. The first, `itemscope`, provides the **item**'s **scope** (i.e., where in the markup you are talking about the type). The second, `itemtype`, the **type** of **item** (i.e., its name).

These attributes are almost exclusively found together, and in the case of applying accessibility metadata, they always go hand in hand. We'll look at the role each of these play in more detail in this section, and how they complement each other.
.
The `itemscope` attribute is attached to the HTML element that contains all the information about the resource you are describing. It doesn't define the type you're using, but identifies the extent of each type in the markup.

For example, the attribute could be added as follows to an HTML `div`:

```
<div itemscope="">
     ...
</div>
```

The attribute identifies here that any metadata that falls between the opening and closing `div` tags belongs together as a logical group, but without making any statement about *why* it belongs together.

As you might expect, the `itemtype` attribute is what you use to define the appropriate schema.org type. The value of this attribute always takes the form http://schema.org/xyz, where "xyz" represents the type name.

For example, using the CreativeWork type requires the `itemtype` value http://schema.org/CreativeWork:

```
<div ... itemtype="http://schema.org/CreativeWork">
```

Returning to the earlier itemscope example, we can now be more explicit that we're talking about a book within the `div` by adding its type:

```
<div itemscope="" itemtype="http://schema.org/Book">
     ...
</div>
```

To find the correct `itemtype` value for the type you want to use, navigate to its definition on the schema.org website. You can then copy the value from the browser address bar into the attribute.

Defining Properties

The `itemprop` attribute is used to define the unique characteristics of the type: the **item**’s **prop**erties. This attribute is by far the most ubiquitous you’ll be using, as each type typically has many different properties.

You can use any of the properties defined in the schema for the type you are describing. For example, a book’s title might be identified as follows using the `name` property:

```
<div itemscope=”” itemtype=”http://schema.org/Book”>
     <h1 itemprop=”name”>The Sound and The Fury</h1>
     ...
</div>
```

In the preceding example, the text value of the `h1` element represents the value, but as noted earlier, accessibility metadata is often not visible. To indicate that our book is a text document without tagging any text, the `accessMode` property can be added using a `meta` tag:

```
<div itemscope=”” itemtype=”http://schema.org/Book”>
     <meta itemprop=”accessMode” content=”textual”/>
     <h1 itemprop=”name”>The Sound and The Fury</h1>
     ...
</div>
```

The value of the property is now defined in the `content` attribute. (We’ll explore how values are established later in this guide, so don’t worry if this seems confusing at this point.)

But that’s all there is to defining properties. The only work involved is finding each bit of information that can be enriched and adding the appropriate semantic property to it. The more work you put into enriching your type by tagging properties, the more value can be extracted from your content by search engines.

Recap

To recap what we’ve just covered, to add microdata to an HTML5 page you first need to declare the scope and type on an element containing the properties:

```
<div itemscope=”” itemtype=”http://schema.org/Book”>
     ...
```

```
	</div>
```

This step never changes, only the value of the `itemtype` attribute.

Having declared the type of resource you're describing, you then tag each applicable property using the `itemprop` attribute:

```
	<div itemscope="" itemtype="http://schema.org/Book">
		<meta itemprop="accessMode" content="textual"/>
		...
	</div>
```

And those are the key microdata attributes you need to know, in a nutshell, to add accessibility metadata to your content.

How to Add Semantics to Markup

With a general idea of how to apply schema.org semantics, it's time to look more closely at how to make statements in your markup, as it's not always clear what you're saying with your metadata, or when you're saying it.

Text Content

When it comes to semantically tagging your text, the first thing to remember is that types always need properties, even when it seems like the text speaks for itself.

For example, the following markup provides no machine-identifiable accessibility information about the book, even though the text conveys the access mode to a reader:

```
	<div itemscope="" itemtype="http://schema.org/Book">
		<p>This resource is primarily textual in nature.</p>
		...
	</div>
```

You have to identify the relevant text and mark it up in order for a machine to be able to process it. Or, looking at the previous example, knowing that "textual" is the key statement being made, we can single out that one word by adding a `span` tag to it defining the `accessMode` property:

```
	<div itemscope="" itemtype="http://schema.org/Book">
		<p>This resource is primarily <span
	itemprop="accessMode">textual</span> in nature.</p>
```

```
		...
	</div>
```

A machine can now pick out the three key concepts: Book accessMode textual. It might sound like Frankenstein talking, but it's the kind of simple association that machines need to turn around and relay this page on to a user looking for books with a textual access mode.

It's critical to note here that the markup you use to tag text content needs to be as specific as possible to be effective. If the `itemprop` attribute had been added to the `p` tag in the preceding example, we'd be back to the search engine having to parse apart the sentence to understand what is being stated.

Meta Tags

As noted earlier, it's always better if the text content of your resource expresses its accessibility, but when that is not possible, the metadata can be expressed using "hidden" `meta` tags (`meta` tags do not add to the visible content of the page).

With `meta` tags, the `content` attribute carries the value you're expressing:

```
	<meta itemprop="accessMode" content="textual"/>
```

In HTML5, this element can be added anywhere within the scope of type you are describing. General best practice, however, is to include such non-visible accessibility metadata as close to where you've defined the `itemtype` attribute as you can. For example:

```
	<div itemscope="" itemtype="http://schema.org/Book">
		<meta itemprop="accessMode" content="textual"/>

		<p itemprop="name">The Sound and the Fury</p>
		<p itemprop="author">William Faulkner</p>
		...
	</div>
```

As this metadata is not visible when updating the page, it can easily be forgotten about, and dispersing it throughout the markup only makes it harder to locate and keep current later.

Multiple Values

A common question with semantic metadata is what to do if more than one property applies to a type. Books often have both textual and visual access modes, for example, as they combine text and images.

When it comes to semantic markup, the advice offered early on in this section about being discreet about what you tag applies. Then we were talking about only tagging the specific word(s) in a larger sentence or paragraph, but the same hold true for multiple semantics. Keep them separate so there is no ambiguity:

```
<meta itemprop=”accessibilityFeature” content=”transcript”/>
<meta itemprop=”accessibilityFeature” content=”captions”/>
```

Do not use a single tag:

```
<meta itemprop=”accessibilityFeature” content=”transcript
captions”/>
```

Although this approach seems simpler and involves less markup, it also introduces ambiguity about your intent. A search engine has to know how to when and how to split apart the value.

Simplicity is always a good thing when it comes to expressing your metadata.

Additional Reading

For more information, please refer to the following resources:

Schema.org
- Type Hierarchy - list of types
- Documentation - introduction to schema.org and semantic markup

Microdata
- Microdata Specification - W3C microdata specification

RDFa
- RDFa Core 1.1 - full RDFa specification
- RDFa Lite 1.1 - subset of RDFa that maps to Microdata

General
- Google Rich Snippets Documentation - various pages covering semantic technologies
- Google Structured Data Testing Tool - check your semantic markup

Appendix A — RDFa Mapping

The RDFa specification provides an even richer set of attributes for defining semantic markup than does microdata, as the specification is more deeply entrenched in the larger goal of a semantic web. But it's a myth that the richness of the specification makes it more complex to implement than microdata, as microdata is, in most respects, a subsetting of RDFa modified to work with the HTML document object model.

To use RDFa attributes with schema.org properties you only need to be aware of a small subset of the full specification also known as RDFa Lite.

The following table shows how to map the attributes for the two technologies:

Microdata	**RDFa**
`itemscope`	N/A
`itemtype`	`typeof`
N/A	`vocab`
`itemprop`	`property`

The most significant difference between the two technologies, at least at this minimal level, is in how you declare a schema.org type. The microdata `itemscope` attribute does not have an equivalent in RDFa, as it is not needed. The RDFa `typeof` attribute implies that the element it is attached to contains all the metadata (i.e., RDFa does in one step what microdata accomplishes in two).

The flip side of that coin is that RDFa does in two steps with its `vocab` and `typeof` attributes what microdata does in one with `itemtype`. In RDFa, the vocabulary you draw your properties from is defined in the `vocab` attribute, and the specific thing you are describing is defined in the `typeof` attribute.

For example, we saw the following microdata to declare the Book type earlier:

```
<div itemscope="" itemtype="http://schema.org/Book">
```

To translate this markup to RDFa you break out the Book type name from the schema.org URL as follows:

```
<div vocab="http://schema.org/" typeof="Book">
```

If you can master those two differences in defining the type, the rest is easy, as the microdata `itemprop` and RDFa `property` attributes function in exactly the same way:

```
<div vocab="http://schema.org/" typeof="Book">
     <meta property="accessMode" content="textual"/>
     <h1 property="name">The Sound and The Fury</h1>
     ...
</div>
```

To recap:

- Instead of adding the microdata `itemscope=""` every time you define a type, in RDFa you always add the attribute `vocab="http://schema.org/"`.
- Instead of putting the full URL of the page where the type is defined in the the microdata `itemtype` attribute, in RDFa you add just the type name in the `typeof` attribute.
- In place of `itemprop`, you use `property`.

Following is a side-by-side comparison:

Microdata	**RDFa**

`<div itemscope="" itemtype="http://schema.org/Book">`
`   <meta itemprop="accessMode" content="textual"/>`
`   <h1 itemprop="name">The Sound and The Fury</h1>`
`   ...`
`</div>`
	`<div vocab="http://schema.org/" typeof="Book">`
	`   <meta property="accessMode" content="textual"/>`
	`   <h1 property="name">The Sound and The Fury</h1>`
	`   ...`
	`</div>`

Although you'll inevitably find yourself using one or the other technology depending on your needs, it's helpful to understand these basic mapping steps, as it will allow you to easily read either markup, regardless of which you're more familiar with. And it is always worthwhile to check your metadata with a structured data testing tool.