Accessibility Metadata
Best Practices Guide

Draft V.5, May 14, 2013

For more information, the specification, examples and news,
go to the Accessibility Metadata Project website, http://a11ymetadata.org/

Table of Contents

About this Guide

Although no knowledge of schema.org or semantic tagging is necessary in order to read this guide, to get the most from it you should have a working knowledge of these technologies. Where you begin in this document will consequently depend on your level of familiarity.

The main body of this guide walks through the accessibility metadata and assumes a general understanding of microdata and schema.org. You can read through the body without any knowledge of the underlying technologies, but the language and examples may seem foreign.

For those completely new to these technologies, it is recommended that you start with the semantic markup primer provided in Appendix A. The primer covers only as much of the technologies as you need to know to, and without bogging you down in technical details. Once you're comfortable with the basic concepts, the tagging of the accessibility metadata will be much easier to understand.

Introduction

Semantic data and semantic markup are two concepts that go hand-in-hand. Semantic data is the end goal of everyone who maintains content on the web—rich, meaningful data that can be

processed by machine (particularly search engines)—while semantic markup is the means by which this information is layered into HTML content. Both are at the heart of making digital content accessible, as it is only through meaningful structure that otherwise arbitrary strings of text begin to assume meaning.

A common critique of HTML, however, is that it has weak semantics. Part of being the transport mechanism for information to the world is that the markup has been kept deliberately simple. You don't find a lot of specialized tags for specialized purposes. For example, when putting the information about a book online you can only add generic tags like numbered headings, paragraphs and spans to identify author names, contributors, copyright holders, the publisher, the edition and other important information about the book. What you haven't been able to do in a universally standardized way is say that those tags contain those unique bits of information.

The simplicity that underlies the Web also makes it difficult for machines to extract meaning, in other words. Search engines are adept at matching the text string you enter against text strings found in web content, but they're still a long way from inferring the meaning of those strings. Language is, put simply, complex. Complex even for humans.

The goal of semantic markup, then, is to enrich this content beyond just the HTML tag names and provide more nuanced information that can simplify processing and making connections. The machine still may not comprehend the text in the way a human does, but with the semantics available, the programmers behind those tools can present the information in meaningful ways.

And that's where semantics and accessibility intersect in the realm of internet searching. The richer the data is, the better it can be presented to users with differing needs and preferences. No more having to access each resource in order to discover its nature.

Not surprisingly, then, the goal of the Accessibility Metadata Project was to improve the results that search engines can provide by defining semantic properties within schema.org, the new framework for expressing search semantics (see the primer for more information). The project does not attempt to address the underlying inaccessibility of formats and protocols on the web; it only seeks to help users in the rapid discovery of resources that meet their needs.

By explicitly identifying access modes (the way that information is conveyed in the resource), media features (enhanced content to enable multiple modalities) and other key aspects of the content, search engines can accurately relay the nature of these resources to users. The ambiguity of searching by keywords—where the presence of search words on a page, regardless of context, determines relevance—will finally be taken out of the equation.

Accessibility Metadata

The Accessibility Metadata Project has contributed a set of nine new properties to the schema.org CreativeWork class:

- `accessMode`
- `mediaFeature`
- `hasAdaptation`
- `isAdaptationOf`
- `accessAPI`
- `ATCompatible`
- `displayTransformability`
- `controlFlexibility`
- `accessHazard`

By choosing to contribute the properties to this existing super-class, they will be inherited by all the specializations of CreativeWork when the proposal is accepted, making the properties instantly available and visible to creators of those works.

In this section of the guide we'll move through each of these properties in turn, explaining their uses and best practices for applying them. We'll also look at the additions to the `bookFormat` property.

Access Modes

Arguably the most important property in the accessibility metadata is `accessMode`, which is defined as follows:

> [a]n access mode through which the intellectual content of a described resource or adaptation is communicated

Unlike the other properties we'll be looking at, every resource has at least one access mode, as information is always conveyed by some discernible means: a novel might have only a textual mode, a cartoon purely visual, comics and manga may have textual and visual components to them, while videos typically have visual and auditory access modes.

This property includes the following predefined set of values that are used to indicate these modes whenever appropriate to your content:

- `auditory`
- `colorDependent`
- `tactile`
- `textOnImage`
- `textual`

- `visual`

We'll review each of these modes in turn as we move through this section.

Note that the access mode of a resource reflects only the primary means by which information is communicated. The addition of media features to enhance access does not change or add new modes. See the section on the effect of media features on access modes for more information.

Auditory

The `auditory` mode is set whenever aural content carries information necessary to the comprehension of the resource:

```
<dt>Access Mode:</dt>
<dd><span itemprop="accessMode">auditory</span></dd>
```

This mode does not apply in all cases where sound is used, however, only where it is fundamental to the comprehension of the resource.

For example, if a web page or ebook has "mood music" playing in the background, the auditory access mode does not need to be set, as someone who cannot hear that music can still consume the content without loss of information.

On the other hand, if a piece of music is included in the resource to provide context, an `auditory` access mode does need to be specified.

Likewise, a silent video would only have a `visual` mode, but one with an accompanying soundtrack would include an `auditory` mode.

Color-Dependent

The `colorDependent` mode indicates that information on the page requires the ability to distinguish color:

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

This mode is commonly set when pie charts, graphs and other visual representations of data convey information, and where the colors used are the only indication of meaning.

Note that if the content passes WCAG success criteria 1.4.1 and contains labels and other

recognition patterns that allow the information to be identified whether the colors used can be distinguished or not, this access mode does not need to be specified.

Tactile

The `tactile` mode indicates that information is conveyed through touch:

```
<dt>Access Mode:</dt>
<dd><span itemprop="accessMode">tactile</span></dd>
```

This mode could be set for print braille books listed in an online catalogue, for example. Electronic braille files, on the other hand, should not have this access mode set. They should instead be identified as `textual` and include the media feature `braille`. The reason is that these files do not have an exclusively tactile interface, it is only when read through a refreshable braille interface or printed that they become tactile. Some text-to-speech programs can voice the files.

If the work is a tactile graphic, or consists of a set of these graphics, this access mode should also be set. If tactile graphics are provided as an alternative to source images, the `tactileGraphics` media feature should instead be set.

This access mode is also applicable to files intended for force feedback devices that convey tactile sensations virtually. It can also be used to identify other physical print sources, such as “touch and feel” books.

Text On Image

When text is written into an image format such as a JPEG or PNG, that text is lost in the pixel data to anyone who cannot view the image or make out the text from the underlying image. As these kinds of raster image formats do not scale well, the text also typically becomes pixellated and unreadable the greater the image is zoomed, impeding comprehension by those who require magnification of the content. It also cannot be read aloud by people with print disabilities using text-to-speech tools.

If your resource includes text in this manner, set the `textOnImage` mode to alert users that the content may not be accessible to them:

```
<meta itemprop=”accessMode” content=”textOnImage”/>
```

This mode should be set even if the text is available in an `alt` attribute or as part of an SVG

image's underlying markup, as the mode always reflects the default nature of the content.

There are many types of works in which text can appear within images. Most notably are comics, graphic novels and manga, but any type of work where the content author has locked the font and size by drawing text graphically will have this access mode.

The content does not have to be drawn into an image using image editing software in order for this access mode to be set, either. A scanned book without optical character recognition performed to turn the page images into text content is another example where this value should be set. A white background behind the text does not change that the access mode is text on an image.

In some exceptional cases, you do not need to set the `textOnImage` mode even though an image may contain text. For non-essential graphics with text in them, such as a logo at the top of a web page containing the resource, this access mode should not be set, since the graphic is peripheral to the content.

Note, too, that this value represents a refinement of the `visual` access mode, so both values do not have to be set in the case of a single image with text, or if all images have text. If the resource combines a mix of visual imagery and imagery with text, both values can be set.

Textual

The `textual` mode is relatively straightforward compared to the others in that indicates that information is conveyed as text content:

```
<dt>Access Mode:</dt>
<dd><span itemprop="accessMode">textual</span></dd>
```

The addition of text alternatives, such as alt text and descriptions for images, should not be considered when setting the primary access mode of a work. These alternatives only affect the media features added.

Visual

The `visual` mode indicates that comprehension of the resource is heavily dependent on being able to see it, whether the visual media is an image, vector artwork, a scripted canvas, an animation or video:

```
<dt>Access Mode:</dt>
```

```
<dd><span itemprop="accessMode">visual</span></dd>
```

If a resource contains only decorative images that do not convey information (equivalent to the ARIA role presentation), this access mode should not be set.

If the images have text written into them, but no other useful visual information, the more precise `textOnImage` mode should be set instead (see above). If the work conveys information both via visual images and images with text, both of these access mode should be set.

Multiple Access Modes

You are not limited to picking just one access mode, as has been noted already. It's often the case that resources may require multiple modalities in order to fully comprehend them.

A typical web page, for example, will include text, image and video content. As all of these access modes are required to consume the content in its entirety, each needs to be set.

Include a separate property for each applicable access mode. For example, a video with a soundtrack would include the following two access modes:

```
<dt>Access Mode:</dt>
<dd>
      <span itemprop="accessMode">auditory</span>,
      <span itemprop="accessMode">visual</span>
</dd>
```

Note that the order in which you list the access modes is not significant.

Media Features

Media features complement the primary access modes and generally make a resource more accessible to a broader range of users. Two visual works are not the same, for example, if one includes alternative (alt) text and descriptions and the other does not.

The accessibility metadata proposal predefines many of the most common features that resources contain:

- `alternativeText` - alternative text is provided for visual content (e.g., the HTML `alt` attribute)
- `audioDescription` - audio descriptions are available (e.g., via the HTML5 `track` element)

- `braille` - braille content or alternative is available (e.g., eBraille or print braille)
- `captions` - captions are available for audio and video content
- `ChemML` - chemistry equations and formulas are provided in ChemML format
- `haptic` - resource facilitates access/control via haptic technologies
- `highContrast` - high contrast options are provided
- `largePrint` - resource is formatted for large print reading
- `latex` - math content is available in LaTeX format
- `longDescription` - descriptions are provided for image-based content and/or complex structures such as tables
- `MathML` - math content is available in MathML format
- `nemethBraille` - math content is available in Nemeth braille format
- `signLanguage` - sign language translation is available for aural content
- `tactileGraphics` - tactile graphics have been provided
- `transcript` - transcripts are provided for audio and video content

As you might expect, a single resource might contain many of these media features, and there is no limit to the number you can use, provided they apply.

When including multiple media features, tag each separately. For example, a textbook that includes alt text and descriptions for images, transcripts for audio and video content, and MathML equations, would include all of the following:

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

      <meta itemprop="mediaFeature" content="alternativeText"/>
      <meta itemprop="mediaFeature" content="longDescription"/>
      <meta itemprop="mediaFeature" content="transcript"/>
      <meta itemprop="mediaFeature" content="MAthML"/>

      <h1 itemprop="name">Introduction to Math</h1>
      ...
</div>
```

Differentiating from Access Modes

The access mode(s) of a work represent the author's primary design for that content—whether the work is the source or an adaptation—while media features represent additional access features.

When marking up the metadata, it can get confusing whether media features constitute a new

access mode if you try to view them as part of that original design. The answer is to keep each intent separate.

When marking up access modes, focus only on the way that information is primarily communicated.

After establishing this primary nature, review the content for media features that enhance it.

Adaptations

Sometimes the resource you're tagging may be an adaptation of another, or you might be aware of other adaptations that you want to make users aware of. The accessibility metadata includes two properties that define these cross-resource relationships: `hasAdaptation` and `isAdaptationOf`.

A source is the primary intellectual work created by a content author and/or publisher, and `hasAdaptation` is used to point from it to its adaptations. Adaptations, on the other hand, are alternative forms that have been created in order to make the original work more accessible to users with differing needs and preferences. The `isAdaptationOf` property, naturally enough, allows you to link back to the source.

An adaptation does not have to be designed and created specifically to address a source in order to be linked to, and vice versa. A website that includes a video of Hamlet, for example, could point to a free online version of the play's text as an adaptation, even though the two were resources were made independently of each other. It only matters that each end of the link chain represents the same content.

If you have control over both the source and adaptation, mark up both relationships. A user may reach the source and use the information to move to a more appropriate adaptation, but when navigating the web it's just as likely that the user will reach the adaptation and want to find the source. If both directions have not been set, one of these scenarios will be inhibited. Similarly, a search engine may only be able to provide information in one direction if both source and adaptation have not been crosslinked.

There are a number of different ways to tag these relationships, depending on whether the source or adaptation is part of the text and whether it is also available online:

- If linkable text is available and the source is reachable on the web, the HTML `a` tag can be used:

  ```
  Adapted from <a itemprop="isAdaptationOf"
  ```

```
href="http://example.com/bookstore/thePrintBook">The Print
Book</a>
```

- If no linkable text is available, but the source is reachable on the web, the link element can be used:

```
<link itemprop="isAdaptationOf"
href="http://example.com/bookstore/thePrintBook"/>
```

- If no linkable text is available, but a universal resource name (URN) is known, the meta element can be used:

```
<meta itemprop="isAdaptationOf"
content="urn:isbn:9781234567890"/>
```

When linking to sources and adaptations, always keep in mind that the accessibility metadata you're defining is unique to the document you're tagging. That a textual adaptation is available for a visual work does not mean that you indicate a textual access mode in the metadata for the visual work.

Note that the changing print/digital landscape confuses these concepts to some degree. If both print and ebook versions of a work are released at the same time, for example, which is the source is often not clear. Although both can technically be the source (and omitting a relationship is not incorrect), for accessibility purposes it is better to identify the ebook as an adaptation of the print as it is will often be more widely accessible to people with print disabilities.

Note, too, that different formats that represent identical access modes do not represent different adaptations of each other (e.g., EPUB and MOBI versions of the same ebook are not adaptations of each other). Work is ongoing in schema.org to define different manifestations of the same work, which, when finished, will better define these kinds of relationships.

Display Transformability

The `displayTransformability` property allows you to indicate that aspects of the resource's rendering can be controlled by the user.

In general, transformability on the Web relates to the ability to manipulate CSS styling. Multimedia resources such as images, videos and audio do not provide such measures for control (note that in the case of image-based content, the ability to zoom is not considered a transformability trait). A text-based page, on the other hand, can typically have all of its text and layout properties changed.

Because of the scope of what can be controlled by CSS, the `displayTransformability` property includes only the single predefined value `css`:

```
<meta itemprop="displayTransformability" content="css"/>
```

Having to list each possible CSS property that can be transformed would be cumbersome, which is what this shorthand alleviates.

You are not limited to this basic all-or-nothing value, however, as it also implies that the user can supply their own styles for the page (a skill that is often too technical for many users). If you want to indicate that only certain properties can be controlled (e.g., through a settings interface), you can instead list each of the main CSS properties that you expose.

For example, an online reading interface for books might list only a few text-control properties as follows:

```
<meta itemprop="displayTransformability" content="font-family"/>
<meta itemprop="displayTransformability" content="font-size"/>
<meta itemprop="displayTransformability" content="color"/>
<meta itemprop="displayTransformability"
content="background-color"/>
<meta itemprop="displayTransformability" content="line-height"/>
<meta itemprop="displayTransformability"
content="word-spacing"/>
```

As has been noted now a few times, include one tag for each configurable property.

Content Control

If you take the time to ensure that your content can be accessed effectively by users with different preferred modalities, you should also indicate such compatibility in your metadata. The `controlFlexibility` property enables the tagging of this information and has two predefined values: `fullKeyboardControl` and `fullMouseControl`.

A good measure for keyboard control is WCAG 2.0 Guideline 2.1. If your content is compliant to this guideline, you can safely set this control flexibility in your metadata:

```
<meta itemprop="controlFlexibility"
content="fullKeyboardControl"/>
```

A similar guideline for mouse accessibility does not exist, but the requirement is generally the same in terms of the user being able to effectively access and control your content by mouse

alone:

```
<meta itemprop="controlFlexibility" content="fullMouseControl"/>
```

If you are not certain if your content is accessible by keyboard and mouse, do not assume for web content that those values must be true. While it is generally the case for basic text and headings works, if your content has any scripted controls or dynamic features, you will need to test to ensure operability.

Note that the need to input text does not make content inaccessible to mouse-only interaction. Users who can only access a computer via a mouse are likely to have an on-screen keyboard available, so if data input is the only keyboard requirement, you can set `fullMouseControl`.

Hazards

Some digital media can be physically dangerous to those who access it. It is commonly known that rates of flashing faster than 3Hz (3 times per second) can cause seizures, but loud repetitive sounds can have the same effect. Nausea is another hazard that can be brought on by motion simulation in visual resources.

When a resource is known to contain such physical hazards, the user needs to discover their presence before accessing the content. Although adding warnings to the page may seem sufficient, such warnings are easily missed.

The `accessHazard` property allows these physiological dangers to be identified so that a search engine can report the hazard. It has three predefined values to account for the above-mentioned situations: `flashing`, `sound` and `motionSimulation`.

For example, a video could be identified as having a flashing hazard as follows:

```
<div itemtype="http://schema.org/VideoObject" itemscope="">
     <p>Warning: This video contains <span
itemprop="accessHazard">flashing</span> that may cause seizures in
some people.</p>
     <video src="rapidFire.webm"/>
</div>
```

Note that providing access to an alternative less hazardous adaptation does not change the need to indicate that the hazard exists.

Accessibility APIs

While you may be led to believe that accessibility metadata applies only to media objects (books, video, etc.), it also applies to programs, whether mobile apps, flash content, Java applets or HTML pages. Noting whether this content is compatible with accessibility APIs is important because it allows users to determine whether they will be able to interact with it via the device they're using:

```
<meta itemprop="accessAPI" content="MSAA"/>
```

This property includes the following predefined API values:

- `AndroidAccessibility`
- `ARIA`
- `ATK`
- `AT-SPI`
- `BlackberryAccessibility`
- `iAccessible2`
- `iOSAccessibility`
- `JavaAccessibility`
- `MacOSXAccessibility`
- `MSAA`
- `UIAutomation`

To create a more complete picture, you can also pair this property with the `operatingSystem` property from the SoftwareApplication class if the content is known to only work on certain systems that support the accessibility API:

```
<span itemprop="operatingSystem" itemscope="" itemtype="http://schema.org/SoftwareApplication">Windows 7</span>
```

In general, you should not need to set the OS-based APIs for HTML, but `JavaAccessibility` and `ARIA` should be set when appropriate.

This property is useful when you are describing a page that uses proprietary plugins, or when you are describing the accessibility of non-web formats, where such formats may only work on certain platforms.

AT Compatibility

It is also possible to use the accessibility metadata to indicate basic compatibility with assistive technologies:

```
<meta itemprop="ATCompatible" content="true"/>
```

This property does not require compliance to a specific set of guidelines, but the following WCAG 2.0 checkpoints are a useful measure for web content:

- 1.1.1 Non-text Content
- 1.3.1 Info and Relationships
- 1.3.2 Meaningful Sequence
- 2.4.4 Link Purpose (In Context)
- 3.1.1 Language of Page
- 3.1.2 Language of Parts
- 3.3.2 Labels or Instructions
- 4.1.1 Parsing
- 4.1.2 Name, Role, Value

If these WCAG guidelines cannot be applied to the content format, another measure of compatibility with assistive technologies will be necessary. The accessibility metadata does not attempt to provide compatibility criteria for every type of resource that can be represented.

Book Formats

The accessibility metadata proposal has also requested that the BookFormatType enumeration be expanded to include the following formats:

- DAISY202
- DAISY3
- EPUB2
- EPUB3

These formats provide information that might otherwise not be easily obtainable. EPUB files are not distinguishable from their file extensions, but require looking into the packaging to find out which version the file conforms to. DAISY books often are zipped up for online distribution, again making it hard to determine their nature without loading them or looking into the content.

An online library could, for example, indicate that a resource is a DAISY 2.02 book by setting this property as follows:

```
<meta itemprop="bookFormat" content="DAISY202"/>
```

Note that at this time it is *not* recommended that the schema.org URL syntax be used for these formats. Properties in the BookFormatType enumeration are normally identified like this:

```
<link itemprop="bookFormat" href="http://schema.org/EBook"/>
```

But until the accessibility metadata proposal is formally accepted into schema.org, these URLs do not exist for the requested formats. Due to the way that schema.org classes are defined using the similar URLs, best practice at this time is to add the above values as plain text strings. Doing so does not impact on the accessibility information made available, but avoids the risk of future conflicts until the proposal is finalized.

If the format name is a part of the text content, it can be tagged directly:

```
<dl>
      <dt>Format:</dt>
      <dd itemprop="bookFormat">DAISY3</dd>
</dl>
```

Migration Path

Having covered all of the available properties, you might be feeling daunted by the breadth of information that you can express. The natural question at this point is: “Do I have to implement it all now?”

Ideally, the answer is yes; but realistically, even small steps will go a long way to making the nature of your content more easily discovered. You may also find that much of the information needed for this metadata is information that you’d had locked away in your databases that you’ve used for content catalogs and organization, but have not had any way to express this as semantic information on the web. Your information was dumbed down for the web: microdata gives you a way to take the accessibility metadata that you’ve been using internally and make this available to search engines and users to more easily discover and leverage.

If you are looking for a simpler path to begin enhancing your content, the single most important step you can take is to ensure that all your resources identify their primary access mode(s) (`accessMode`). This information instantly conveys to a user much of what they can expect if they try to access the resource.

The logical complement to the access mode is to also list all of the supplied media features (`mediaFeature`), to ensure that users know about all the ways you’ve enhanced your content.

Although they may not be common to many types of resources, the hazards the content presents are just as important as the means of access and available features because of the

potential for real physical harm to users.

These properties in combination will give readers a wealth of information about the nature of the resource and the accommodations available, allowing them to make informed decisions without having to navigate to the content.

Next in importance are sources and adaptations. Of course, there are going to be many times when the source document you're tagging has no adaptations, but if you are aware of relationships they are important to tag as they allow a search engine to find and present the alternatives to the user. This information could also be used in the future to allow direct linking of web pages for the source and the adaptations.

The remaining four properties carry information that is no less valuable, but represents refinements to the way the content is accessed that are not as critical. With the exception of `displayTransformability`, these properties also require more specialized understanding of accessible best practices and implementation techniques, so as your understanding grows it will become easier to correctly implement these.

Or to break the properties down by overall importance:

Importance	**Properties**
Critical	accessMode mediaFeature accessHazard
High	hasAdaptation isAdaptationOf
Normal	displayTransformability controlFlexibility ATCompatible accessAPI

Frequently Asked Questions

1) Can I use the accessibility metadata now?

Yes. Even though the metadata has not been officially included in schema.org as of writing, that does not prevent you from using it to enhance your content. You may receive warnings from

validation tools, like Google's Structured Data Testing Tool, that the properties are not a part of the class you are using, but these warnings can be ignored.

2) Can I use other values with the metadata?

You are strongly encouraged to use the predefined values whenever they are applicable, as it is only through standardization that intelligent processing and presentation of the information can be done. But there is no reason you can't use other values if they are applicable.

If you find values missing from the predefined set, you are encouraged to contact the Accessibility Metadata Project working group to have them considered for inclusion.

Appendix A – Semantic Markup Primer

schema.org

Knowing that semantic metadata plays a key role in discovering the accessibility of online resources, the logical question you're probably asking yourself is "what form does this markup take and how does one go about adding it?" Before jumping straight into the technical details of semantic markup, it helps to first understand schema.org, both the site and the concept.

Schema.org, the web site, was created by four of the major search providers in an effort to standardize semantic markup for search engine optimization, and was launched in 2011 by Google, Microsoft, Yahoo! and Yandex.

The site does not add new tags or attributes to HTML, but hosts semantic vocabularies that can be used with two other web technologies: RDFa and microdata. These technologies provide the mechanism by which you enrich your content and schema.org provides the means.

Although schema.org was originally created by search providers, it is not an exclusive club. The site was intended to be developed and expanded on by third parties interested in enriching content on the web, and anyone can make proposals to define new vocabularies or enhance existing ones.

The Accessibility Metadata Project developed one such proposal for adoption into schema.org, as detailed in the main body of this guide. The proposal contributes a new set of properties to an existing class in schema.org, but to understand what this means we first need to review the structure of schema.org.

Classes

Although schema.org can be conceived of as one big vocabulary for the web, it's actually a hierarchical collection of what are called classes, each of which defines a micro-vocabulary (or schema, hence the site's name). These classes 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 classes: DataType and Thing. The DataType class defines the kinds of data expected when you use properties, but more important to this discussion is Thing. Below the Thing class lies all of schema.org's major vocabularies. (Note: if this structure seems murky still, it's best to have a look at the Full Hierarchy page.)

When it comes to the accessibility metadata, be aware that the key class under Thing to be aware of is CreativeWork, as the accessibility metadata properties have been contributed to it.

Another interesting fact about schema.org is that properties can be inherited downward by the specialization subclasses. By leveraging this inheritance, the accessibility metadata will be 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 classes 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 classes, 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 class defines what you're talking about, properties are what you use to make specific statements about it. A class 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, without actually indicating the title, author, publisher, ISBN or publication date, isn't terribly helpful to a search engine trying to index your page. It knows what your page is defining (a book), 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 class and its content: the book has a title, this person is an author, etc. If you stop here for a moment and look at the Book class 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 universal resource locators (URLs).

In the case of schema.org generally, the properties of the class 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 the 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 B. (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 class (in the case of `itemscope` and `itemtype`) as well as its defining characteristics (`itemprop`).

Defining Classes

Microdata uses two attribute to define classes. The first, `itemscope`, provides the **item**'s **scope** (i.e., where in the markup you are talking about the class). The second, `itemtype`, the **type** of **item** (i.e., the class 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 class you're using, but identifies the extent of each class 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.

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

For example, using the CreativeWork class 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 class:

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

To find the correct `itemtype` value for the class 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 class: the **item**'s **prop**erties. This attribute is by far the most ubiquitous you'll be using, as each class typically has many different properties.

You can use any of the properties defined in the schema for the class 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 class 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 class 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 classes 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 class 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.

Linked Resources

Sometimes your metadata will identify a relationship to another document. The `hasAdaptation` and `isAdaptationOf` properties are two that operate in this way: one used to identify the the location or identifier of an adaptation and the other to identify that the current resource is an adaptation of another.

In both these cases, a URL typically identifies the source or adaptation. And, as you're probably already well aware, URLs are most often found in HTML hyperlinks (i.e., in the `href` attribute attached to `a` tags).

The `href` attribute value overrides the text value of the tag, so you only have to attach these properties to an element with an `href` to make the association. You don't have to include the URL as text content, in other words.

Or, going by example, you could create the property value by including the full URL as a text string on your page like this:

```
This ebook was adapted from <span
itemprop="isAdaptationOf">http://example.com/ebook/MyVisualEbook
</span>
```

But unlinked URLs are an uncommon occurrence on the web. More typically, you'll be able to leverage the `href` attribute on an `a` tag:

```
This ebook was adapted from <a itemprop="isAdaptationOf"
href="http://example.com/ebook/MyVisualEbook">http://example.com
/ebook/MyVisualEbook</a>
```

And, to be clear, it's not the text content that matters when an `href` attribute is present. The following markup is exactly equivalent to the previous, at least from a semantic association point of view, even though the book name will be the visible hyperlink label:

```
This ebook was adapted from <a itemprop="isAdaptationOf"
href="http://example.com/ebook/MyVisualEbook">My Visual
Ebook</a>
```

If the document doesn't not contain linkable text, the `link` element can be used instead:

```
<link itemprop="isAdaptationOf"
href="http://example.com/ebook/MyVisualEbook"/>
```

Once again, the `href` attribute makes the association, and this markup is semantically equivalent to both of the preceding examples that used the `a` tag.

Note that you should never use empty `a` tags to express these kinds of property associations (`<a itemprop="isAdaptationOf" href="..."/>`) as these can cause issues for screen readers (e.g., unwanted announcement of the `href` value).

Multiple Values

A common question with semantic metadata is what to do if more than one property applies to a class. 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="accessMode" content="textual"/>
<meta itemprop="accessMode" content="visual"/>
```

Do not use a single tag:

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

Although this approach seems simpler and involves less markup, it also introduces ambiguity about your intent. Are you defining two separate properties here, or have you created a third? To get the right answer, a machine has to know which properties to break apart, and how to split them.

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

More Information

This section has, by design, provided only a very basic introduction to semantic markup aimed at getting you started with applying the accessibility metadata in schema.org. For more information, please refer to the following resources:

Schema.org

- Type Hierarchy - list of Classes
- 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 B – 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 bigger 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 you only need to be aware of a small subset of the full specification. The following table shows how the two technologies map attributes:

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

These attributes are also referred to as RDFa Lite, and they are all you need to know to use schema.org metadata in your content.

The most significant difference between the two technologies, at least at this minimal level, is in how you declare a schema.org class. 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.

So, for example, the following microdata to describe a book:

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

gets translated to RDFa by breaking out the Book class 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 class, 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>
```

So, to recap:

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

Following is a side-by-side comparison:

Microdata	RDFa

  <meta itemprop="accessMode" content="textual"/>
  <h1 itemprop="name">The Sound and The Fury</h1>
  ...
 	
	  <meta property="accessMode" content="textual"/>
	  <h1 property="name">The Sound and The Fury</h1>
	  ...

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.

Appendix C – Quick-Start Templates

The following basic templates can be used to quickly apply the accessibility metadata to some common creative works.

Note that the use of `meta` tags is not required, but done for simplicity. If the accessible nature of the content can be exposed as text content, the meta tags should be reformulated to a more appropriate form.

Video

The only accessibility property that must be set for all videos is the `visual` access mode. All other metadata is conditional.

```
<body itemscope="" itemtype="http://schema.org/VideoObject">
    <meta itemprop="accessMode" content="visual"/>

    <!-- if there is a soundtrack to the video: -->
    <meta itemprop="accessMode" content="auditory"/>

    <!-- if captions have been provided: -->
    <meta itemprop="mediaFeature" content="captions"/>

    <!-- if a transcript is available: -->
    <meta itemprop="mediaFeature" content="transcript"/>

    <!-- if an audio description for the soundtrack: -->
    <meta itemprop="mediaFeature" content="audioDescription"/>
```

```
      <!-- if sign language transcription provided: -->
      <meta itemprop=”mediaFeature” content=”signLanguage”/>

      <!-- if a keyboard accessible player: -->
      <meta itemprop=”controlFlexibility”
content=”fullKeyboardControl”/>

      <!-- if a mouse accessible player: -->
      <meta itemprop=”controlFlexibility”
content=”fullMouseControl”/>

      <!-- if a flashing hazard in the video: -->
      <meta itemprop=”accessHazard” content=”flashing”/>

      <!-- if a motion hazard in the video: -->
      <meta itemprop=”accessHazard” content=”motionSimulation”/>

      <!-- if a sound hazard in the video: -->
      <meta itemprop=”accessHazard” content=”sound”/>

      <video .../>
</body>
```

Audio

The only accessibility property that must be set for all audio clips is the `auditory` access mode. All other metadata is conditional.

```
<body itemscope=”” itemtype=”http://schema.org/AudioObject”>
      <meta itemprop=”accessMode” content=”auditory”/>

      <!-- if captions have been provided: -->
      <meta itemprop=”mediaFeature” content=”captions”/>

      <!-- if a transcript is available: -->
      <meta itemprop=”mediaFeature” content=”transcript”/>

      <!-- if an audio description for the soundtrack: -->
      <meta itemprop=”mediaFeature” content=”audioDescription”/>

      <!-- if sign language transcription provided: -->
```

```
      <meta itemprop=”mediaFeature” content=”signLanguage”/>

      <!-- if a keyboard accessible player: -->
      <meta itemprop=”controlFlexibility”
content=”fullKeyboardControl”/>

      <!-- if a mouse accessible player: -->
      <meta itemprop=”controlFlexibility”
content=”fullMouseControl”/>

      <!-- if a sound hazard: -->
      <meta itemprop=”accessHazard” content=”sound”/>

      <audio .../>
</body>
```